QNAP2
version 9.3
Reference Manual

July 1996

Contents

Chapter 1

DECLARE command

ANY	Highest level type. Root of all other types.
BOOLEAN	Declaration of a boolean variable.
CLASS	Declaration of a customer class object.
CUSTOMER	Customer object type.
EXCEPTION	Declaration of an exception object.
FILE	Declaration of a file object.
FLAG	Declaration of a flag object.
FUNCTION	Declaration of a user-defined function.
INTEGER	Declaration of an integer variable.
OBJECT	Declaration of an object type.
PROCEDURE	Declaration of a user-defined procedure.
QUEUE	Declaration of a queue object.
REAL	Declaration of a real variable.
REF	Declaration of a pointer to an object.
STRING	Declaration of a character string variable.
TIMER	Declaration of a timer object.
WATCHED	Declaration of a user statistical variable.

NAME

ANY - Highest level type. Root of all other types.

SYNTAX

REF ANY *id* ;
REF ANY *id1*, *id2*, ...;
REF ANY *id* (*integer*) ;

DESCRIPTION

Declaration of pointers to a fictitious object type representing the root of the object types tree.

Such a pointer can be assigned a reference to whatever object type (predefined or not).

EVALUATION

During compilation.

WARNING

- Such an object can only be considered as a reference. It is not possible to create instances of that type (neither statically, nor dynamically).
- A pointer of that type cannot be assigned a variable of a scalar type (INTEGER, REAL, BOOLEAN and STRING).
- Only the 8 first characters of the identifier are considered.

SEE ALSO

:: - OBJECT - REF

EXAMPLE

```
/DECLARE/ QUEUE OBJECT machine ;
            REAL duration ;
          END ;
          REF  ANY rany ;

/STATION/ NAME  = *machine ;
          SERVICE  =  BEGIN
                        ...
                        rany := CUSTOMER ;
                        rany := QUEUE::machine ;
                        rany := CLASS ;
                        PRINT (rany::CLASS.ICLASS) ;
                        ...
                      END ;
```

BOOLEAN

NAME

BOOLEAN - Declaration of a boolean variable.

SYNTAX

BOOLEAN *id* [= *boolean*] ;
BOOLEAN *id* (*size*) [= *list-of-booleans*] ;
BOOLEAN *id1* [= *boolean1*], *id2* [= *boolean2*], ... ;

DESCRIPTION

Declaration of boolean variables or arrays.

A boolean variable can only have two values: TRUE or FALSE . The size of a boolean array is an integer value (type INTEGER) which can be either a constant or an expression.

Variables, resp. arrays, can be statically initialised by using the "=" assignment operator, followed by a boolean value, resp. a list of boolean values.

EVALUATION

During compilation.

NOTES

The default value of a non-initialised boolean variable is FALSE.

WARNING

Only the 8 first characters of the identifier are considered.

EXAMPLE

```
/DECLARE/  BOOLEAN b ;
           BOOLEAN b1 = TRUE  ;
           BOOLEAN b2 (10) ;
           BOOLEAN b3, b4 ;
           INTEGER i1 = 18 ;
           BOOLEAN b5 (i1) ;
           BOOLEAN b6 (3) = (TRUE, FALSE, TRUE) ;
           BOOLEAN b7 (10) =  (TRUE REPEAT 10) ;
```

NAME

CLASS - Declaration of a customer class object.

SYNTAX

CLASS *id* ;
CLASS *id* (*size*) ;
CLASS *id1, id2, ...* ;

DESCRIPTION

Declaration of customer class object variables or arrays. A customer class gathers an homogeneous population of customers statistically having the same behaviour.

size is an integer value (type INTEGER) which can be either a constant or an expression.

EVALUATION

During compilation.

NOTES

Before the solution starts, a customer class can also be dynamically created using the NEW function.

WARNING

Only the 8 first characters of the identifier are considered.

SEE ALSO

NEW - NMAX

EXAMPLE

```
/DECLARE/  CLASS c ;
           CLASS c1 (10) ;
           REF CLASS @_class;
           INTEGER i1 = 18;
           CLASS c4 (i1) ;

....

/EXEC/ BEGIN
         @_class := NEW (CLASS);
         SIMUL;
         ...
       END;
```

CUSTOMER

NAME

CUSTOMER - Customer object type.

SYNTAX

CUSTOMER *type id* ;
CUSTOMER *type id* (*size*) ;

DESCRIPTION

Within a /DECLARE/ block, the CUSTOMER keyword can only be used to extend the CUSTOMER object type.
All further created customers will have the attributes added to the predefined type.

size is an integer value (type INTEGER) which can be either a constant or an expression.

EVALUATION

During compilation.

NOTES

It is not allowed to create customer objects in a /DECLARE/ section.

WARNING

Only the 8 first characters of the identifier are considered.

SEE ALSO

NEW - INIT (station)

EXAMPLE

```
/DECLARE/ CUSTOMER REAL t(3) ;
          CUSTOMER REF QUEUE source ;
          REF CUSTOMER @_cust;
          ...
          QUEUE q1,q2;

/STATION/ NAME = q;
          INIT = 1;             & Creation of the first customer
          SERVICE = BEGIN
                      @_cust := NEW (CUSTOMER); & Creation of a son customer
                      TRANSIT (@_cust, q2);
                      ...
                    END;
```

NAME

EXCEPTION - Declaration of an exception object.

SYNTAX

EXCEPTION id (*size*) ;
EXCEPTION *except1, except2, ...* ;
EXCEPTION (*handler_procedure*) except 3 ;

DESCRIPTION

Declaration of exception objects in a QNAP2 model.

Such objects allow to trigger user-defined procedures in case of a particular event.

Captured events can be:

- Asynchronous interruption signal sent by the machine operating system.
- Events related to the simulation: start and end of simulation, fixed statistical precision reached.

Exceptions are handled through calls to the appropriate SETEXCEPT:keyword procedure.

The following exception objects are predefined in QNAP2:

- SIMSTART, exception raised whenever the simulation is activated. Its default handler is void. Note that when the REPLICATION confidence intervals computation method is used, each new replication activates the simulation.
- SIMSTOP, exception raised whenever the simulation ends. As noted above, when using the REPLICATION method, the end of each replication is considered as the end of a simulation. The default handler associated to this execption, and also forced after a user-defined handler, will stop the simulation or the replication.
- SIMACCUR, exception raised whenever the requested statistical precision is reached on all concerned variables and results. The default handler stops the simulation. As opposed to the SIMSTOP handler, it is not forced after a user-defined handler and thus the user should call the STOP procedure to stop the simulation. If the user-defined handler is missing such a call, simulation continues and further activation of the exception is possible.

The exception object type has a predefined attribute, named EXCEPTION:DEFHANDLER, which is a pointer to the triggered procedure. This procedure can then be specified when the exception is created or declared, or later using the SETEXCEPT:HANDLER:CONNECT procedure.

EVALUATION

During compilation.

NOTES

When an exception is raised, if needed its associated processing is activated (*after all current events have been processed*). The processing is a simple procedure without argument called the HANDLER of the exception. During its execution, the current exception raised is available through the implicit pointer EXCEPTION. All simulation operations are available inside a handler procedure.

SEE ALSO

SETEXCEPT:CONNECT - SETEXCEPT:DISCONNECT - SETEXCEPT:MASK - SETEXCEPT:UNMASK - SETEXCEPT:HANDLER - SETEXCEPT:LAUNCHTIMER - SETEXCEPT:CANCELTIMER - SIMSTART - SIMSTOP - SIMACCUR - SETSTAT:PRECISION

EXAMPLE

```
/DECLARE/ PROCEDURE HANDLER; &  Handler of the exception
          BEGIN
            PRINT ("AN INTERRUPTION ARRIVED");
          END;

          EXCEPTION INTERRUPT;  & Declararation of an exception object

/EXEC/ BEGIN
        SETEXCEPT:CONNECT (INTERRUPT, "signal", HANDLER); & Connection
                                                          & of the handler
                                                          & to the exception
        SIMUL ;
      END;
```

NAME

FILE - Declaration of a file object.

SYNTAX

FILE *id* ;
FILE *id* (*size*) ;
FILE *id1, id2, ...* ;

DESCRIPTION

Declaration of file object variables or arrays.
A file is a logical entity to which can be associated a physical file.

size is an integer value (type INTEGER) which can be either a constant or an expression.

Predefined files in QNAP2 are:

- FSYSINPU : implicit input of the model. This file is assigned and opened. It cannot be neither reassigned, nor closed.
- FSYSOUTP : implicit output of results and error messages. This file is assigned and opened. It cannot be neither reassigned, nor closed.
- FSYSTERM : terminal input. This file is assigned and opened. It cannot be neither reassigned, nor closed. It can instead be interactively used when typing commands from keyboard.
- FSYSGET : implicit file for reading operations (GET and GETLN). It can be assigned before its first use; that allows to use it as an ordinary file. If not, it is automatically assigned and cannot then be neither reassigned nor closed.
- FSYSPRIN : implicit file for writing operations (PRINT, WRITE and WRITELN). It can be used the same way as the FSYSGET file.
- FSYSTRA : implicit file for simulation trace. It can be used the same way as the FSYSGET file.
- FSYSLIB : implicit file for save/restore operations. It can be used the same way as the FSYSGET file. It is not formatted, as opposed to other usual files of QNAP2.

EVALUATION

During compilation.

NOTES

A file object can also be dynamically created using the NEW function.

WARNING

- Only the 8 first characters of the identifier are considered.
- Only ASCII (or EBCDIC) sequential files can be treated by QNAP2.

SEE ALSO

FILASSIGN

EXAMPLE

```
 /DECLARE/  FILE f;
            FILE f1 (10);
            FILE f2, f3;
            INTEGER i1 = 18, i2;
            FILE f4 (i1);

/EXEC/ BEGIN
         FILASSIGN (f, "file.dat");
         OPEN (f, 1);
         i2 := GETLN (f, INTEGER);
         CLOSE (f);
       END;
```

NAME

FLAG - Declaration of a flag object.

SYNTAX

FLAG *id* ;
FLAG *id* (*size*) ;
FLAG *id1, id2, ...* ;

DESCRIPTION

Declaration of flag object variables or arrays. A flag object is used for synchronisation purpose.

The flag status is either passing or blocking.

size is an integer value (type INTEGER) which can be either a constant or an expression.

EVALUATION

During compilation.

NOTES

A flag object can also be dynamically created using the NEW function.

The default status of a flag object is blocking.

WARNING

Only the 8 first characters of the identifier are considered.

SEE ALSO

NEW - SET - RESET - WAIT - WAITAND - WAITOR - FREE - STATE - FLINKCUS - FLISTCUS

EXAMPLE

```
 /DECLARE/  FLAG f;
            FLAG f1 (10);
            FLAG f2, f3;
            INTEGER i1 = 18;
            FLAG f4 (i1);

/STATION/  NAME=q;
           SERVICE = BEGIN
                       WAIT (f); & wait for flag status to become passing
                       ...
                     END;
```

FUNCTION

NAME

FUNCTION - Declaration of a user-defined function.

SYNTAX

type FUNCTION *func-name* (*arg1, ... ,argn*);
[VAR] *type11 arg1*; & Declaration of arguments
[VAR] *type12 arg2*; & Declaration of arguments
.....
BEGIN
... & algorithmic code
END;

or:
type FUNCTION *func-name* (*arg1, ... ,argn*);
[VAR] *type11 arg1*; & Declaration of arguments
[VAR] *type12 arg2*; & Declaration of arguments
.....
GENERIC;

In case the function returns a reference rather than a value the declaration is:
REF *type* FUNCTION *func-name* (*arg1, ... ,argn*);
[VAR] *type11 arg1*; & Declaration of arguments
[VAR] *type12 arg2*; & Declaration of arguments
.....
BEGIN
...
END;

DESCRIPTION

The FUNCTION keyword allows the definition of a user-defined function. The result of such a function should be unique and cannot be an array. The body of the function is described using the algorithmic language of QNAP2.

type: is the type of the returned result. It can be any scalar type: INTEGER, REAL, BOOLEAN, STRING (size), or a reference to an object of any type (types like REF ANY, REF to procedure or REF to function are also allowed).

(arg1, arg2,...): is the formal arguments list. They are declared the same way as for a procedure. They should be immediatly declared after the function header. Like for procedures, VAR arguments are allowed, they provide a way to return a value through the returned result as well as through the arguments.

Inside the body of the function, a reserved local variable named RESULT is automatically declared by QNAP2 with the type of the returned result. It will be used to hold the returned result, and should therefore be assigned a value. It is initialised according to its type (zero, NIL or "").

Like for a procedure, a function can be declared as GENERIC. Such a declaration defines a "pointer to function" type. An object of that type can then be assigned a real function, by giving the identifier of the function followed by the ADDRESS keyword, or another pointer of the same type. A real function is compatible with a generic one if, and only if, both returned type and formal arguments lists are equal (number of arguments are equal, type and passing mode are equal for each argument). When calling a function through a pointer to generic function, an actual arguments list must be catenated to the pointer expression, allowing distinction between the value of the pointer and the call to the referenced function.

EVALUATION

During compilation.

NOTES

- User-defined functions can be recursive. Whatever simulation operations or synchronisations the function uses in its body, it can be shared between different simulated customers during their service.
- If the formal arguments list is empty, parenthesis are useless.
- A function can be called within any algorithmic code (/EXEC/ section, /REBOOT/ section, service description, ENTRY command, EXIT command or TEST command).

WARNING:

As a restriction of the actual version, a call to a function is not allowed in a boolean expression to select items within a list using the WITH operator (not to be confused with the ''WITH object DO'' statement, inspired from the Pascal programming language).

SEE ALSO

PROCEDURE - REF

EXAMPLE

```
/DECLARE/ INTEGER FUNCTION SUM (A,B);
          INTEGER A,B;
          BEGIN
            RESULT := A+B;
          END;

/EXEC/ BEGIN
         PRINT (SUM (3,4));      & Print 7
       END;

/DECLARE/ INTEGER FUNCTION PTRFNC (A,B);
          INTEGER A,B;
          GENERIC;

          REF PTRFNC sumbis;

/EXEC/ BEGIN
          sumbis := SUM ADDRESS;
          PRINT (sumbis (3,4));   & Print 7 like above
       END;
```

NAME

INTEGER - Declaration of an integer variable.

SYNTAX

INTEGER *id* [= *integer*] ;
INTEGER *id* (*size*) [= *list-of-integers*] ;
INTEGER *id1* [= *integer1*], *id2* [= *integer2*] , ... ;

DESCRIPTION

Declaration of integer variables or arrays.

size is an integer value (type INTEGER) which can be either a constant or an expression.

Variables, resp. arrays, can be statically initialised by using the "=" assignment operator, followed by an integer value, resp. a list of integer values.

EVALUATION

During compilation.

NOTES

The default value of a non-initialised integer variable is 0.

WARNING

Only the 8 first characters of the identifier are considered.

SEE ALSO

WATCHED

EXAMPLE

```
/DECLARE/  INTEGER i;
           INTEGER i1 = 2;
           INTEGER i2 (10);
           INTEGER i3, i4;
           INTEGER i5 (i1);
           INTEGER i6 (3) = (1, 3, 6);
           INTEGER i7 (10) = (1 REPEAT 10);
```

OBJECT

NAME

OBJECT - Declaration of an object type.

SYNTAX

OBJECT *id* [(*par1*, *par2*,...)] ;
type1 id1 [, *id2*, ...] ;
type2 id3 [, *id4*, ...] ;
...
END ;
type OBJECT *id* [(*par1*, *par2*, ...)] ;
type1 id1 [, *id2*, ...] ;
type2 id3 [, *id4*, ...] ;
...
END ;

DESCRIPTION

The OBJECT keyword allows for definition of new object types, which can be either inherited types, or completely new types. The types hierarchy can be represented as a tree which root is the fictitious type ANY.

Properties of inherited types, also called subtypes, are:

- instances of the subtype have the same characteristics as those of the supertype,
- adding attributes to a supertype is no longer possible as soon as a subtype has been defined.

Entities between the object type identifier and the END keyword are called the *attributes* of the object. An attribute can be a simple variable, an array, an object instance or a pointer to an object. It can be initialised when declaring an object instance if it is specified as a parameter of the type, that is in the parameter list following the identifier of the object type. Each parameter in the list should after appear in the attributes section.

Creation of an instance can be achieved either statically within a /DECLARE/ section, or dynamically in an algorithmic sequence using the NEW function.

EVALUATION

During compilation.

NOTES

- In case of a QUEUE subtype, or if an attribute of a subtype is a QUEUE, it is possible to build a virtual description of the corresponding station. In such a case, the description should occur before the creation of the first instance of the subtype (refer to the definition of the NAME parameter of the /STATION/ command).

- Access to a subtype object through a reference to a supertype is achieved by the :: operator.

WARNING

- Forward references to yet undeclared types are allowed providing that the definition of the referenced type will occur within the same /DECLARE/ section.
- Only the 8 first characters of the identifier are considered.

SEE ALSO

ANY - NAME - ::

EXAMPLE

```
 /DECLARE/  OBJECT PROCESSOR ;
            QUEUE cpu, mem ;
            REAL mips ;
            END ;

            OBJECT PROGRAM (nb_jobs) ;
             INTEGER nb_jobs ;
             REF PROCESSOR rproc (nb_jobs) ;
            END;

            QUEUE OBJECT ENTRY ;
            WATCHED REAL inter ;
            END ;

            ENTRY ent1, ent2;   & static creation of entry objects
            PROCESSOR p1;       & static creation of a processor
            PROGRAM (3) pg1 ;   & static creation of program of 3 jobs
            REF ENTRY re ;      & declaration of a pointer to an entry
            REF PROCESSOR rp ;  & declaration of a pointer to a procesor
            REF PROGRAM rpg ;   & declaration of a pointer to a program
            ...

/STATION/ NAME= *PROCESSOR.mem ;& virtual definition of mem stations
          ...

/EXEC/
BEGIN
  re := NEW (ENTRY) ;       & dynamic creation of an entry object
  rm := NEW (PROCESSOR) ;   & dynamic creation of a processor object
  rg := NEW (PROGRAM, 4) ; & dynamic creation of a program object
                            & of 4 jobs
```

```
END ;
...
```

NAME

PROCEDURE - Declaration of a user-defined procedure.

SYNTAX

```
PROCEDURE proc-name [(par1, par2, ... ]) ;      Normal procedure
[VAR] type1 id1 [, id2, ... ] ;                 Declaration of arguments
[VAR] type2 id3 [, id4, ... ] ;                 Declaration of arguments
...
type3 id5 [, id6, ... ] ;                       Local variables declaration
type4 id7 [, id8, ... ] ;                       Local variables declaration
...
BEGIN
...                                             Body of the procedure: algorithmic code
END ;

PROCEDURE proc-name [(par1, par2, ...]) ;       Forward procedure
[VAR] type1 id1 [, id2, ... ] ;                 Declaration of arguments
[VAR] type2 id3 [, id4, ... ] ;                 Declaration of arguments
...
FORWARD ;

PROCEDURE proc-name ;                           Definition of the forward procedure
[type3 id5 [, id6, ... ] ;                      Local variables declaration
type4 id7 [, id8, ... ] ; ]                     Local variables declaration
...
BEGIN
...                                             Body of the procedure: algorithmic code
END ;

PROCEDURE proc-name [(par1, par2, ...]) ;       Generic procedure
[VAR] type1 id1 [, id2, ... ] ;                 Declaration of arguments
[VAR] type2 id3 [, id4, ... ] ;                 Declaration of arguments
GENERIC;
```

DESCRIPTION

Three different types of procedure can be declared within a /DECLARE/ section:

- normal procedures,
- forward procedures and their corresponding body,
- generic procedures.

Arguments are given as a list following the procedure identifier. They should also be declared before the body of the procedure, like any local variable of the procedure.

Arguments can be passed by reference if the VAR keyword appears before the argument type. They are otherwise passed by value.

Locality is preserved within a procedure and thus local variables of a procedure can have the same identifier as a variable of another procedure, a global variable or an attribute of an object.

A forward procedure declaration contains the procedure identifier, the arguments list followed by the arguments declaration. Local variables declaration can also appear in the forward declaration provided that it does not appear in the definition of the procedure. The body of the procedure should not appear in the forward declaration, but it must be specified in the current DECLARE bloc.

The definition of a forward procedure contains the identifier of the procedure, **without the arguments list**, eventually followed by the local variables declaration (*if it wasn't specified in the forward declaration*), and the body of the procedure.

A generic procedure declaration contains the procedure identifier, the arguments list followed by the arguments declaration. As it is only a "type of procedure" declaration, no body is allowed there.

EVALUATION

- During compilation for declarative parts.
- During execution of the procedure for its body.

NOTES

- As a generic procedure defines a type, a pointer to a generic procedure is declared as any other pointer, with the keyword REF.
- Assignment of a normal procedure to a pointer to generic procedure is achieved by a simple assignment operation with the ":=" operator. Arguments list of the normal procedure is then matched against the generic procedure one to check if they are both equal (number of arguments are equal, type and passing mode are equal for each argument).
- A procedure can be called within any algorithmic code (/EXEC/ section, /REBOOT/ section, service description, ENTRY command, EXIT command or TEST command).

WARNING

- QNAP2 does not allow to declare procedures inside other procedures, e.g., like in Pascal.
- When the procedure is called inside the service of a virtual station, no implicit reference to the customer being served are available, *as opposed to the body of the service.*
- A procedure using customer manipulation operations can only be used within simulation.
- The body of a procedure defined in FORWARD **must** be specified in the current DECLARE bloc.

SEE ALSO

REF

EXAMPLE

```
/DECLARE/ QUEUE q;
          INTEGER m;
          PROCEDURE  p1;        & procedure without arguments
          BEGIN
           PRINT ("p1");
           ...
          END;                  & end of procedure

/EXEC/ BEGIN
        p1;
        ...
       END;

/DECLARE/ PROCEDURE p2 (i,j);   & procedure with arguments
          VAR INTEGER i;        & i is passed by reference
          INTEGER  j;           & j is passed by value
          BEGIN
           i := j + 3;
           ...
          END;                  & end of procedure

/STATION/ NAME     = q;
          SERVICE = p2 (m, 3);  & call to procedure p2
                                & within a service
          ...

/DECLARE/ PROCEDURE  p3 (i);    & forward declaration of a procedure
          INTEGER i;
          FORWARD;              & end of forward declaration
          PROCEDURE  p4;
          INTEGER k;
          BEGIN
             ...
             p3 (k);            & call to a procedure wihtout body
             ...
          END;
          PROCEDURE  p3;        & definition of the forward procedure
                                & description of its body


          BEGIN
            ...
```

```
            p4;
            ...
          END;


/CONTROL/ TEST = p4;              & call to a procedure within a TEST command


/DECLARE/ PROCEDURE p5 (i,j);      & generic procedure
         VAR INTEGER i;
         INTEGER j;
         GENERIC;                 & end of generic procedure

         REF p5 rp;               & pointer to a generic procedure
         INTEGER i;
         ...
 /EXEC/
  BEGIN
     rp := p1;                    & assignment to a pointer to procedure
     rp (i, 4);                   & call to a procedure through a pointer
  END;
```

QUEUE

NAME

QUEUE - Declaration of a queue object.

SYNTAX

QUEUE *id* ;
QUEUE *id* (*size*) ;
QUEUE *id1, id2, ...* ;

DESCRIPTION

Declaration of queue object variables or arrays.

A queue is an elementary entity. It enqueues customers with a particular ordering policy.

size is an integer value (type INTEGER) which can be either a constant or an expression.

EVALUATION

During compilation.

NOTES

- A queue can be associated to a station description (see all the instructions concerning the /STATION/ command). This is not mandatory as a queue can also be passive.
- Before the solution starts, a queue can also be dynamically created using the NEW function.

WARNING

Only the 8 first characters of the identifier are considered.

SEE ALSO

NEW - NAME - OBJECT

EXAMPLE

```
/DECLARE/  QUEUE q;
           QUEUE q1 (10);
           QUEUE q2, q3;
           OBJECT GENERAL;
            QUEUE q4, q5;
           END;

/STATION/ NAME = q;
          ...
```

NAME

REAL - Declaration of a real variable.

SYNTAX

REAL *id* [= *real*] ;
REAL *id* (*size*) [= *list-of-reals*] ;
REAL *id1* [= *real1*], id2 [= *real2*], ... ;

DESCRIPTION

Declaration of real variables or arrays.

size is an integer value (type INTEGER) which can be either a constant or an expression.

Variables, resp. arrays, can be statically initialised by using the "=" assignment operator, followed by a real value, resp. a list of real values.

EVALUATION

During compilation.

NOTES

The default value of a non-initialised real value is 0.0.

WARNING

Only the 8 first characters of the identifier are considered.

SEE ALSO

WATCHED

EXAMPLE

```
/DECLARE/ REAL r;
          REAL r1 = 2.8;
          REAL r2 (10);
          REAL r3, r4;
          INTEGER i1 = 18;
          REAL r5 (i1);
          REAL r6 (3)   = (1E + 4, 3.65, 6.9);
          REAL r7 (10) = (1.2 REPEAT 10);
```

REF

NAME

REF - Declaration of a pointer to an object.

SYNTAX

REF *type id* ;
REF *type id (size)* ;
REF *type id1, id2, ...* ;

DESCRIPTION

Declaration of pointer to object variables or arrays.

type can be any predefined type (QUEUE, CUSTOMER, CLASS, FLAG, FILE and ANY), as well as any user-defined type.

A pointer to an object behaves exactly like an object, except that it can be assigned either an object or a pointer to an object.

size is an integer value (type INTEGER) which can be either a constant or an expression.

EVALUATION

During compilation.

NOTES

- If the pointer is used as an attribute, the type it references may not be already defined as QNAP2 allows forward references within a /DECLARE/ section.
- Pointers and objects allow to manipulate data structures like chained lists, trees, ...

WARNING

Within a /DECLARE/ section, static creation of customers is not allowed, whereas declaration of pointers to customer (REF CUSTOMER) is allowed.

SEE ALSO

OBJECT - QUEUE - CUSTOMER - CLASS - FLAG - FILE - ANY - PROCEDURE - FUNCTION

EXAMPLE

```
/DECLARE/  REF QUEUE rq;
           REF CLASS rclist (10);
           REF CUSTOMER rc1, rc2;
           REF QUEUE rq1, rq2 (5);

           OBJECT object1;
             INTEGER iobj;
           END;

           REF object1 robj;

           OBJECT object2;
             REF QUEUE rqobj;
           END;

           OBJECT object3;
             REF object4 robj4;
           END;

           OBJECT object4;
             REF object3 robj3;
           END;

           OBJECT object5;      & chained list
             INTEGER iobj;
             REF object5 robj5;
           END;
           QUEUE q;

/STATION/ NAME = q;
          SERVICE = BEGIN
                      robj := NEW ( object1 );
                      rq := q;
                      rq1 := rq2;
                      TRANSIT (rc1, q);
                      TRANSIT (rq1);
                      ...
                    END;
```

STRING

NAME

STRING - Declaration of a character string variable.

SYNTAX

STRING [*length*] *id* [=*string*] ;
STRING [*length*] *id* (*size*) [=*list-of-strings*] ;
STRING [*length*] *id 1*[= *string1*], *id2* [= *string2*] ,... ;

DESCRIPTION

Declaration of character string variables or arrays.

size and *length* are integer values (type INTEGER) which can be either constants or expressions.

length is the number of characters of the created string. The default length is 80 characters. Variables, resp. arrays, can be statically initialised by using the "=" assignment operator, followed by a string value, resp. a list of string values.

EVALUATION

During compilation.

NOTES

The default value of a non-initialised string variable is "".

If the " character is to be stored in a character string, it should be written as "". For instance """" is the string character representing ".

WARNING

- Only the 8 first characters of the identifier are considered.
- *length* should be within 0 and 256.

SEE ALSO

STRLENGTH - STRMAXL

EXAMPLE

```
/DECLARE/ STRING b ;  & b is a 80 characters long string (default)
          STRING b1 = "TRUE" ;
          STRING b2 (10) ;
          STRING b3, b4 ;
          STRING (120) b5 ;  & b5 is a 120 characters long string
          INTEGER i1 = 18 ;
          STRING b6 (i1) ;
          STRING b7 (3) = ("TRUE", "FALSE", "TRUE") ;
          STRING b8 (10 )= ("TRUE" REPEAT 10) ;
```

NAME

TIMER - Declaration of a timer object.

SYNTAX

TIMER *t1* (*size*) ;
TIMER *t2, t3, ...* ;

DESCRIPTION

Declaration of timer object variables or arrays. Such objects introduce simulation events not related to customers.

Predefined timer objects are available during simulation, they concern:

- simulation start, TSTART option of the /CONTROL/ section, measurement period in some case (spectral method for confidence intervals computation on standard results);
- total simulation time (TMAX option of the /CONTROL/ section).

Timer objects are handled either:

- by means of calls to the appropriate SETTIMER:keyword procedure;
- automatically: predefined timers are handled automatically. A few user actions are allowed.

A timer object is attached a procedure, handler, called whenever the timer expires. All simulation operations are available inside a handler procedure, except delay operations and synchronisations.

TIMER object attributes are:

- ACTIARG : Last activation argument used for a timer.
- HANDLER : Handler procedure associated to the timer.
- STATE : Activation mode of the timer.
- TIMPRIOR : Timer priority.

EVALUATION

During compilation.

NOTES

Some timer objects are predefined in QNAP2:

- TSYSTMAX corresponds to the total simulation time. The user can change its activation time. When it expires, simulation ends.
- TSYSPERI corresponds to the user period (PERIOD option of the /CONTROL/ section). The user is free to completely manage it, overwriting then the implicit behaviour.

- TSYSTSTR corresponds to the TSTART option of the /CONTROL/ section. The user is free to completely manage it, overwriting then the implicit behaviour.
- TSYSTRACE corresponds to the TRACE option of the /CONTROL/ section. The user is free to completely manage it, overwriting then the implicit behaviour.
- TSYSSPCT corresponds to the measurement period for confidence intervals computation by the spectral method. The user can only read this timer. It was introduced for debugging purpose.
- TSYSWDOG stops the simulation as a last resort (*with an error status*). The user cannot manipulate this timer.

SEE ALSO

SETTIMER:CANCEL - SETTIMER:ABSOLUTE - SETTIMER:RELATIVE - SETTIMER:CYCLIC - SETTIMER:TRACKTIME - SETTIMER:SETPROC - TIMPRIOR - STATE - ACTIARG - HANDLER

EXAMPLE

```
/DECLARE/ TIMER MYTIMER;
          PROCEDURE TIMHANDLE;
          BEGIN
            PRINT (" TIMER ACTIVED AT TIME", TIME);
          END;
          ...

/STATION/ NAME = ...
          SERVICE = BEGIN
                      SETTIMER:ABSOLUTE (MYTIMER, 10.0);
                      & Activation of the timer within 10 units of time
                      ....
                      SETTIMER:CANCEL (MYTIMER);
                      & Cancel previous call to SETTIMER:ABSOLUTE
                      ...
                    END;

/ EXEC / BEGIN
          ...
          SETTIMER:SETPROC (MYTIMER, TIMHANDLE);
          & connection of the timer to its handler procedure
          ...
         END;
```

NAME

WATCHED - Declaration of a user statistical variable.

SYNTAX

WATCHED INTEGER *id1* ;
WATCHED REAL *id2* ;

DESCRIPTION

Declaration of a user real or integer statistical variable. Such a variable allows the user to automatically obtain statistical results on a real or integer variable, object attribute or array element.

The variable will be set as discrete by the SETSTAT:DISCRETE (..) procedure, or as continuous by the SETSTAT:CONTINUE (..) procedure.

Statistical results will be provided by calls to the GETSTAT:keyword functions.

EVALUATION

During compilation.

WARNING

A WATCHED variable can be passed by reference to a procedure or function only if the corresponding formal argument was defined as WATCHED.

SEE ALSO

GETSTAT:MEAN - SETSTAT:CONTINUE - SETSTAT:DISCRETE - SETSTAT:ON - SETSTAT:OFF - SETSTAT:SAMPLE - SETSTAT:CANCEL

EXAMPLE

```
/DECLARE/ WATCHED INTEGER I;
          PROCEDURE service(j);
            VAR WATCHED INTEGER j;
            INTEGER K;
            BEGIN
              FOR K:=1 STEP 1 UNTIL 10 DO
                j:=K*10;
            END;
          QUEUE Q;

/STATION/ NAME=Q;
          INIT=1;
          SERVICE=service(I);
          TRANSIT=OUT;

/CONTROL/ TMAX=1;

/EXEC/
          BEGIN
            SETSTAT:DISCRETE(I);
            SIMUL;
            PRINT("I ",I);
            PRINT("MEAN ",GETSTAT:MEAN(I));
            PRINT("MIN ",GETSTAT:MINIMUM(I));
            PRINT("MAX ",GETSTAT:MAXIMUM(I));
            PRINT("VAR ",GETSTAT:VARIANCE(I));
          END;

/END/
```

Chapter 2

STATION command

CAPACITY	Limited capacity queue.
COPY	Station description by copying the characteristics of another station.
FISSION	Customer fission.
FUSION	Customers fusion.
INIT	Initial number of customers in a station.
MATCH	Customers fusion.
NAME	Name of station parameter.
PRIOR	Priority level parameter.
QUANTUM	Service quantum allocation parameter.
RATE	Server speed parameter.
REJECT	Reject processing policy parameter.
SCHED	Queue ordering and server allocation policy parameter.
SERVICE	Service description parameter.
SPLIT	Customer splitting.
TRANSIT	Customer routing parameter.
TYPE	Station type parameter.

NAME

CAPACITY - Limited capacity queue.

SYNTAX

CAPACITY = *integer* ;
CAPACITY (*class1*, *class2*, ...) = *integer* ;

DESCRIPTION

This parameter allows to define a limited capacity queue.

With the first syntax, the capacity is limited to the number specified by *integer*, whatever the customer class is.

With the second syntax, the limited capacities apply to the specified customer classes.

If both syntax are used within a station description, the most restrictive of both specified capacities for a customer class applies.

When a customer enters a queue whose capacity has been reached, two different processings are possible:

- When there's no possible preemption, the customer is rejected, it remains in the queue from where it was attempting to transit.
- If preemption is possible (SCHED=PREEMPT option) and the customer is considered as having a higher priority than another customer in the queue (higher priority level or SCHED=LIFO, PREEMPT), it replaces a customer having a lower priority than itself and which will be rejected.

The processing of a rejected customer is specified by the REJECT parameter of the station description.

class1, *class2*, ... are identifiers of declared objects of type CLASS.

integer is an expression or a constant evaluating to positive INTEGER.

EVALUATION

At initiation time.

WARNING

A customer rejected with no reject policy specified will end the solution time.

SEE ALSO

REJECT - SKIP - CAPACITY - QREJECT - CLREJECT - CPREEMPT

EXAMPLE

```
/DECLARE/ CLASS X, Y, Z;
/STATION/ NAME = ...;
          CAPACITY = 5;
          CAPACITY (X, Y) = 4;
          CAPACITY (Z) = 3;
          REJECT = SKIP;
```

The global queue capacity is 5. And the queue can contain no more than 4 customers of class X and of class Y, and no more than 3 customers of class Z.

When a customer enters the queue, the most restrictive of these three specifications applies.

NAME

COPY - Station description by copying the characteristics of another station.

SYNTAX

COPY = *queue*;

DESCRIPTION

This syntax allows to define a new station whose characteristics are copied from those of another station specified by *queue*.

queue is an indentifier of a declared object of type QUEUE.

EVALUATION

During compilation.

NOTES

If no other parameter is specified for the station, then the result is equivalent to having the name of the current station added to the name list of the NAME parameter of the reference station.

Such a specification can also be used to describe a station which is quite identical to another one but not really. The parameters which differ may then be redefined.

EXAMPLE

```
/DECLARE/ QUEUE q1, q2, q3;
          CLASS c1, c2;

/STATION/ NAME = q1;
          INIT = 2;
          SERVICE = EXP (4);
          TRANSIT = q3;

/STATION/ NAME = q2;
          COPY = q1;
          INIT (c1) = 1; & only difference compared to station q1
          ...
```

FISSION

NAME

FISSION - Customer fission.

SYNTAX

```
FISSION [(class1, class2, ...)] =
        (queue11,class11,nb11, ... ,queue1n,class1n,nb1n) [prob1,
        (queue21,class21,nb21, ... ,queue2n,class2n,nb2n) prob2,
        ...
        (queuep1,classp1,nbp1, ... ,queuepn,classpn,nbpn) probp];
        ...
```

DESCRIPTION

This parameter describes, in place of a TRANSIT parameter, the behaviour of a customer which finishes its service. In this case, the customer is splitted into several pieces specified by the *(queue, class, nb, ...)* list indicating the destination queue, the destination class and the number of pieces generated. The number of pieces for each queue/class can be omitted and is assumed to be 1. The class specification can be omitted and is assumed to be equal to the class of the origin customer.

As opposed to a SPLIT operation, resulting customers don't keep the information of their common origin. This disables the possibility to later group the customers together by a MATCH. They have no father customer, as opposed to customers created by calls to the NEW(CUSTOMER) function. They can only be joined together with other customers by a FUSION operation.

class1, *class2*, ... specify the customer classes to which the fission applies. All classes for which no particular fission is specified are concerned if the class list is omitted.

Several fission policies can be specified and their related probabilities, *prob1*, *prob2*, ..., allow to choose between them. The probability values are normalised if their sum is not equal to 1.

When the fission is completed, the initial customer is destroyed, after executing a possible trace action.

Each piece can also be splitted further, either by a FISSION or a SPLIT operation.

queue11, *queue12*, ... are identifiers of declared objects of type QUEUE.

class1, *class2*, ..., *class11*, *class12*, ... are identifiers of declared objects of type CLASS.

nb11, *nb12*, ... are expressions or constants evaluating to positive INTEGER.

prob1, *prob2*, ... are expressions or constants evaluating to REAL.

EVALUATION

At initiation time.

NOTES

The FISSION parameter can only be used in simulation.

WARNING

As a FISSION is equivalent to a TRANSIT or a SPLIT, these parameters cannot be combined within the same /STATION/ command.

SEE ALSO

SPLIT - MATCH - FUSION

EXAMPLE

```
/DECLARE/ QUEUE inter1, inter2, inter3;
          CLASS c1,c2;

/STATION/ NAME = inter1;
          FISSION  = (inter2,2,inter3,2);
          SERVICE = EXP (1.);
```

FUSION

NAME

FUSION - Customers fusion.

SYNTAX

```
FUSION = (cl11,nb11,...,cl1n,nb1n)resultclass1:PRIOR(integer1),
         (cl21,nb21,...,cl2n,nb2n)resultclass2:PRIOR(integer2),
         ...
         (clp1,nbp1,...,clpn,nbpn)resultclassn:PRIOR(integern);
```

or:

```
FUSION = (cl11,nb11,...,cl1n,nb1n)resultclass1:WEIGHT(real1),
         (cl21,nb21,...,cl2n,nb2n)resultclass2:WEIGHT(real2),
         ...
         (clp1,nbp1,...,clpn,nbpn)resultclassn:WEIGHT(realp);
```

DESCRIPTION

The FUSION operation allows to fusion several customers into one, wherever the customers come from: initialisation customers, dynamically created customers (NEW(CUSTOMER)) or customers resulting of a FISSION operation (or a SPLIT operation even though it is not adviced as a FUSION operation does not maintain the origin information of a customer).

Customers joined by a FUSION are immediatly destroyed (i.e., sent to OUT) and a new customer is created with the specified class. This customer can now execute a service.

(clp1,nbp1,...,clpn,nbpn) specifies the joining set. It is a list of class, number of customers couples. All couples must be present simultaneously to enable the join.

resultclass specifies the class of the resulting customer.

Different join policies can be specified by the FUSION parameter. PRIOR(*integer*) or WEIGHT(*real*) are then used to decide which one to apply, based upon different priority levels or different probabilities. If neither PRIOR, nor WEIGHT are specified, the first completed join is applied.

cl11, *cl12*, ..., *resultclass1*, *resultclass2*, ... are identifiers of declared objects of type CLASS.

nb11, *nb12*, ..., *integer1*, *integer2*, ... are expressions or constants evaluating to positive INTEGER.

real1, *real2*, ... are expressions or constants evaluating to REAL.

EVALUATION

At initiation time.

WARNING

The FUSION parameter can only be used in simulation.

Customers waiting for a FUSION are not considered as being in the waiting queue of the station and thus are not considered for statistics computation. Instead, customers joined are placed in the waiting queue.

SEE ALSO

SPLIT - FISSION - MATCH

EXAMPLE

```
/DECLARE/ QUEUE exit, inter1;
          CLASS c1, c2;

/STATION/ NAME = inter1;
          INIT = 1;
          FISSION  = (exit,c1,2,exit,c2,2);
          SERVICE = EXP (1.);

/STATION/ NAME = exit;
          TRANSIT = OUT;
          FUSION = (c1,2,c2,2) c1:WEIGTH(2.),(c1,2,c2,2) c2:WEIGTH(1.);
          SERVICE = EXP (2.);
```

INIT

NAME

INIT - Initial number of customers in a station.

SYNTAX

INIT = *integer* ;
INIT [(*class1*, *class2*, ...)] = *integer*;

DESCRIPTION

The initial number of customers in a station before solution time can be specified either for all classes or for a list of classes by the INIT parameter. It can appear several times within the same /STATION/ command, and the one whose class list is empty applies to all classes for which no INIT parameter is defined.

The order of creation of the customers depends on the following criteria given in decreasing order :

- priority of the classes in the queue (highest priority first)
- classes declaration order (first declared class on head)

class1, *class2*, ..., are identifiers of declared objects of type CLASS.

integer is an expression or constant evaluating to INTEGER.

EVALUATION

Class list is evaluated at compile time, whereas the *integer* value is evaluated at initiation time.

NOTES

- This parameter is not mandatory.
- No initial customer is placed in the queue if no INIT parameter is defined in the station description.

WARNING

This parameter cannot be used for stations of type SEMAPHORE, RESOURCE or SOURCE (refer to the TYPE parameter description).

EXAMPLE

```
/DECLARE/ QUEUE q1, q2, q3;
          CLASS c1, c2, c3;
          INTEGER i = 3;

/STATION/ NAME = q1;
          INIT = 2;              & Customers in the queue
                                 &  2 customers of class c1
                                 &  2 customers of class c2
                                 &  2 customers of class c3
                                 & Order: c1, c1, c2, c2, c3, c3

/STATION/ NAME = q2;
          INIT (c1) =1;          &  1 customer of class c1
          INIT = i;              &  i customers of class c2
                                 &  i customers of class c3
                                 & Order: c1, c2, c2, c2, c3, c3, c3
         ...

/STATION/ NAME = q3;
          INIT (c2, c3) = 2;     &  2 customers of class c2
                                 &  2 customers of class c3
          INIT = 1;              &  1 customer of class c1
                                 & Order: c1, c2, c2, c3, c3
         ...
```

MATCH

NAME

`MATCH` - Customers fusion.

SYNTAX

```
MATCH = (origin):(cl11,nb11,...,cl1n,nb1n)resultclass1[:PRIOR(integer1),
        (origin):(cl21,nb21,...,cl2n,nb2n)resultclass2:PRIOR(integer2),
        ...
        (origin):(clp1,nbp1,...,clpn,nbpn)resultclassn:PRIOR(integern)];
```

or:

```
MATCH = (origin):(cl11,nb11,...,cl1n,nb1n)resultclass1[:WEIGHT(real1),
        (origin):(cl21,nb21,...,cl2n,nb2n)resultclass2:WEIGHT(real2),
        ...
        (origin):(clp1,nbp1,...,clpn,nbpn)resultclassn:WEIGHT(realp)];
```

DESCRIPTION

The `MATCH` operation allows to fusion several customers into one, with respect to their origin, i.e., they should come from the same intitial customer which was splitted by one or many `SPLIT` operations. The origin customer information is therefore necessary to fusion customers with the `MATCH` operation. It is maintained over several `MATCH` operations allowing to fusion customers in several steps.

origin specifies the origin queue, queue/class or queue/class-list, globally surrounded by parentheses and followed by the ":" character. It references the queue where the `SPLIT` operation was performed, and possibly the class(es) of customers for which the `SPLIT` was specified.

The *(clp1,nbp1,...,clpn,nbpn)* set specifies the joining set. It is a list of classes, number of customers couples. All couples must be present simultaneously to enable the join.

Customers of different initial customer will wait separately for their own.

Customers joined by a `MATCH` are immediatly destroyed (i.e., sent to `OUT`) and a new customer is created with the specified class (*resultclass*). This customer can now execute a service.

Different join policies can be specified by the `MATCH` parameter. `PRIOR(`*integer*`)` or `WEIGHT(`*real*`)` are then used to decide which one to apply, based upon different priority levels or different probabilities. If neither `PRIOR`, nor `WEIGHT` are specified, the first completed join is applied.

origin is an identifier of a declared object of type `QUEUE`.

cl11, *cl12*, ..., *resultclass1*, *resultclass2*, ... are identifiers of declared objects of type `CLASS`.

nb11, *nb12*, ..., *integer1*, *integer2*, ... are expressions or constants evaluating to positive INTEGER.

real1, *real2*, ... are expressions or constants evaluating to REAL.

EVALUATION

At initiation time.

WARNING

If a customer created by a SPLIT operation and awaited in a MATCH is unfortunately sent to OUT, the MATCH will never be completed and other customers concerned by this match will wait forever.

Customers waiting for a MATCH are not considered as being in the waiting queue of the station and thus are not considered for statistics computation. Instead, customers joined are placed in the waiting queue.

SEE ALSO

SPLIT - FISSION - FUSION

EXAMPLE

```
/DECLARE/ QUEUE inter1, inter2, inter3;
          CLASS c1, c2;

/STATION/ NAME = inter1;
          SPLIT (c1) = (inter2,c2,2,inter3,c2,2) 0.4,
                       (inter2,c1,2,inter3,c1,2) 0.6;
          SERVICE = EXP (1.);

/STATION/ NAME = inter2;
          TRANSIT = OUT;
          MATCH = (inter1):(c1,4) c1;
          SERVICE = EXP (2.);

/STATION/ NAME = inter3;
          TRANSIT = inter2;
          SERVICE = EXP (3.);
```

NAME

NAME - Name of station parameter.

SYNTAX

NAME = *queue1* [,*queue2*, ...] ; & for a list of queues
NAME = *queue* (*lower-bound* STEP *step* UNTIL *upper-bound*) & for an array subset
NAME = ∗*object-id*[.*queue*] ; & for a virtual station

DESCRIPTION

This parameter defines the name of a station. The name must be a declared identifier of a variable (simple, list or array), or an attribute (simple, list or array) of an object, of type:

- QUEUE or any of its subtype,
- reference to a QUEUE object , or any of its subtype,

or even an indentifier of a QUEUE subtype.

In case of an object attribute, the object type identifier, *object-id*, is used to prefix the attribute identifier as shown in the third syntax.

In case of an identifier of a QUEUE subtype, the identifier is prefixed by the "*" character.

lower-bound, *step* and *upper-bound* are expressions or constants evaluating to INTEGER.

EVALUATION

During compilation.

NOTES

QUEUE objects can be passive and therefore are not always associated to a station.

WARNING

- This parameter is mandatory and must appear first in the station description.
- If the queue identifier is a reference to a queue, it must be assigned a queue when the NAME parameter is evaluated.

EXAMPLE

```
/DECLARE/ QUEUE q, q1, q2, q3(10), q4, q5 ;
          REF QUEUE rq1, rq2 ;
          QUEUE OBJECT qo ;
             INTEGER qoi ;
          END ;

          OBJECT o ;
            QUEUE oq ;
          END ;

/STATION/ NAME = q ;
...
/STATION/ NAME = q1, q2 ;
...
/STATION/ NAME = q3 (1 STEP 1 UNTIL 5) ;
          ...

/EXEC/ BEGIN
         rq1 : = q4 ;
         rq2 : = q5 ;
       END ;

/STATION/ NAME = rq1, rq2 ;
          ...

/STATION/ NAME = *qo ;
          ...

/STATION/ NAME = *o.oq ;
```

PRIOR

NAME

PRIOR - Priority level parameter.

SYNTAX

PRIOR [(*class1*, *class2*, ...)] = *integer*;

DESCRIPTION

This parameter defines the priority level of customers entering the station. It can be specified globally or for a particular class list. The global specification applies to all classes for which no class specification exists. Several PRIOR specifications are allowed within the same /STATION/ command.

Customers of classes for which no priority level is defined keep the priority level they have before entering the station.

class1, *class2*, ... are identifiers of declared objects of type CLASS.

integer is an expression or constant evaluating to a positive INTEGER.

EVALUATION

- During compilation for the class list.
- At initiation time for the priority level.

NOTES

By default, customers entering the queue keep their priority level.

SEE ALSO

PRIOR (Procedure) - TRANSIT (Procedure) - AFTCUST - BEFCUST

EXAMPLE

```
/DECLARE/ QUEUE q1, q2, q3;
          CLASS c1, c2, c3;
          INTEGER i;

/STATION/ NAME = q1;
          PRIOR = 2;          & Priority level 2 for all customers
                              & entering the station
          ...

/STATION/ NAME = q2;
          PRIOR = 2*i;        & Priority level 2*i for customers of classes
                              & c2 and c3 (computed at initiation time)
                              &
          PRIOR (c1) = 1;     & Priority level 1 for customers of class c1
          ...

/STATION/ NAME = q3;
          INIT (c2, c3) = 2; & Priority level 2 for customers of class c2
                              & and c3, customers of class c1 keep their
          ...                 & priority level
```

QUANTUM

NAME

QUANTUM - Service quantum allocation parameter.

SYNTAX

QUANTUM [(*class1*, *class2*, ...)] = *real*;

DESCRIPTION

This parameter allows to define the service quantum allocation to each class.

Several QUANTUM specifications are allowed within the same /STATION/ command. If no class is specified, the definition applies to all classes for which no service quantum allocation is defined within the same /STATION/ command.

This parameter is meaningful if the SCHED policy of the station is set to QUANTUM (cf. the SCHED parameter).

class1, *class2*, ... are identifiers of declared objects of type CLASS.

real is an expression or constant evaluating to REAL.

EVALUATION

- During compilation for the class list.
- At initiation time for the quantum value.

SEE ALSO

SCHED

EXAMPLE

```
/DECLARE/ QUEUE q1, q2, q3, q4;
          CLASS x, y, z;
          REAL r;

/STATION/ NAME = q1;
          SCHED = QUANTUM;
          QUANTUM = 10.0;         & quantum for all classes: 10.0
          ...

/STATION/ NAME = q2;
          SCHED = QUANTUM (2.0); & quantum for class z: 2.0
          QUANTUM (x, y) = 3.0;  & quantum for classes x and y: 3.0
          ...

/STATION/ NAME = q3;
          SCHED = QUANTUM;
          QUANTUM = 2.0;          & quantum for class z: 2.0
          QUANTUM (x) = 3.0;      & quantum for class x: 3.0
          QUANTUM (y) = 3*r;      & quantum for class y: 3*r
          ...
```

RATE

NAME

RATE - Server speed parameter.

SYNTAX

RATE (*class1*, *class2*, ...) = *real* | *real-array* | *real1*, *real2*, ...

DESCRIPTION

The RATE parameter defines the speed of the station server(s). It is a ratio representing the number of work units performed during each time unit.

Several RATE specifications are allowed within the same /STATION/ command. If no class is specified, the RATE parameter defines the nominal rate of the server(s). It applies directly to all classes for which no rate is defined within the same /STATION/ command. If a class specification and a nominal rate specification exist in the same station, they must be multiplied together to obtain the real rate for that class.

The service rate can be a single real value, *real*, representing a constant service rate. The service time of a customer is obtained by dividing the work demand for that customer by the real service rate.

The service rate can also be an array or list of real values, *real-array* or *reali*, representing the instantaneous service rate which depends on the number of customers in the queue of the station. The last value in the array/list applies also if the number of customers in the queue of the station is greater than or equal to the number of values in the array/list.

class1, *class2*, ... are identifiers of declared objects of type CLASS.

real, *real1*, *real2*, ... are expressions or constants evaluating to positive REAL. *real-array* is an array of expressions or constants evaluating to positive REAL.

EVALUATION

- During compilation for the class list.
- At initiation time for the rate value(s).

NOTES

The default service rate is 1.

SEE ALSO

SERVICE

EXAMPLE

```
/DECLARE/ QUEUE q1, q2, q3;
          CLASS c1, c2, c3;
          INTEGER i;

/STATION/ NAME = q1;
          SERVICE = CST (1);
          RATE = 2, 4;       & Service time for 1 customer: 1/2 = 0.5
                             & Service time for 2 customers or more:
                             &       1/4 = 0.25
/STATION/ NAME = q3;
          SERVICE = CST (1);
          RATE = 2, 4;
          RATE (c1) = 5, 1;
          ...
          & Service time will be:
          & - for class c2 or c3 customers:
          &     1/2 = 0.5 for 1 queueing customer
          &     1/4 = 0.25 for 2 or more queueing customers
          & - for class c1 customers:
          &     1/(2*5) = 0.1 for 1 class c1 queueing customer
          &     1/(4*5) = 0.05 for 2 or more queueing customers with
          &                     only 1 class c1 customer
          &     1/(4*1)=0.25 for 2 or more queueing customers with
          &                  2 or more class c1 customers

/STATION/ NAME = q2;
          SERVICE = CST (1);
          RATE (c1) = 5;     & Service time for class c1 customers:
                             &   1/(2*5) = 0.1
          RATE = 2;          & and for class c2 or c3 customers:
                             &   1/2 = 0.5
```

REJECT

NAME

REJECT - Reject processing policy parameter.

SYNTAX

REJECT = *statement-list*
REJECT (*class1*, *class2*, ...) = *statement-list*

DESCRIPTION

This parameter specifies the processing of customers that are rejected due to the limited capacity (cf. CAPACITY parameter) of the station. No implicit processing is defined by QNAP2.

Several REJECT specifications are allowed within the same /STATION/ command. If no class is specified, the definition applies to all classes for which no reject processing is defined within the same /STATION/ command.

The processing is described by an algorithmic code (*statement-list*).

class1, *class2*, ... are identifiers of declared objects of type CLASS.

EVALUATION

- During compilation for the class list.
- During execution for the statement list.

NOTES

The processing of a rejected customer due to the limited capacity of its destination station is specified in the destination station, whereas its current queue (CQUEUE attribute of the customer) remains its origin station. Attributes functions like CUSTOMER, QREJECT, CLREJECT or CPREEMPT allows to obtain useful information about the current reject.

WARNING

If no reject processing is defined and a customer is rejected due to the limited capacity of the station, an error message is issued by QNAP2 and solution time ends.

If the model is to be solved by an analytical solver or the Markovian solver, the only statement allowed is a call to the SKIP procedure.

To ensure a proper fate to the rejected customer, the reject processing should include at least one of the following:

- a work demand procedure, in order to wait for a free place in the station; this is considered as a continuation of the service in the origin station;
- a *blocking* synchronisation operation (e.g., P, WAIT);

- a successful TRANSIT to another station or to the same station after making room.

Should none of these be satisfied, causing a new reject to the same customer, an error message is issued by QNAP2 and the simulation stops.

SEE ALSO

CAPACITY parameter - CAPACITY attribute - SKIP - QREJECT - CLREJECT - CPREEMPT

EXAMPLE

```
/DECLARE/ CLASS X, Y;

/STATION/ NAME = ...
          CAPACITY = 5;
          REJECT (X) = BEGIN
                         PRINT ("A class X customer was rejected");
                         TRANSIT (CUSTOMER, OUT);
                       END;
          REJECT = BEGIN
                       PRINT ("A customer of class other than X was rejected");
                       TRANSIT (CUSTOMER, OUT);
                   END;
```

SCHED

NAME

SCHED - Queue ordering and server allocation policy parameter.

SYNTAX

SCHED = FIFO | LIFO | PS | QUANTUM (*real*) | EXCLUDE *concurrency-set*[, ...] | RESEQUENCE (*queue*[, *class*, ...])

```
SCHED = [ FIFO, ] PRIOR
SCHED = [ FIFO, ] PRIOR, PREEMPT
SCHED = LIFO, PRIOR
SCHED = LIFO, PRIOR, PREEMPT
SCHED = LIFO, PREEMPT
SCHED = [FIFO|LIFO][FEFS,] EXCLUDE concurrency-set[, ...]
```

DESCRIPTION

This parameter describes both the queue ordering and server allocation policy of the station. It determines the order the customers will be placed in the queue when they enter the station and hence the order they will be served.

Some options determine the order the customers will be placed in the queue when they enter the station. They are:

- FIFO: customers are ordered according to their order of arrival, this is the default SCHED value;
- LIFO: customers are ordered according to the reverse order of their arrival;
- FIFO, PRIOR: customers are ordered according first to their priority level (highest priority first), and then, for equal priority customers, according to their order of arrival;
- LIFO, PRIOR: customers are ordered according first to their priority level (highest priority first), and then, for equal priority customers, according to the reverse order of their arrival;

Other options are more specific to the server allocation and scheduling policy. They are:

- QUANTUM[(*real*)]: a server is allocated for fixed length period defined by the *real* argument or by the QUANTUM parameter(s) of the station;
- PS: a server is allocated for variable length period determined by the current number of customers in the queue and the service time; this is equivalent to a QUANTUM policy except that the period is recomputed each time a new customer enters the station;
- PREEMPT: any arriving customer placed *before* a customer being served preempts the service of the one being served, which is in turn placed back in the queue;
- EXCLUDE *concurrency-set*[, ...]: this option applies to multiple server stations or infinite stations; it specifies that two customers belonging to the same concurrency set cannot be served simultaneously (see below the definition of concurrency sets);
- FEFS, EXCLUDE *concurrency-set*[, ...]: unlike the simple EXCLUDE option, this one only

concerns multiple server stations; it specifies that a customer waiting in the queue can seize a free server if it is declared "eligible" (according to the specification of the EXCLUDE policy presented above), even if other "non eligible" customers are waiting before itself (without a FEFS specification, this customer would remain waiting and the server would stay free); this option is mandatory for analytical solvers and the Markovian solver;

- RESEQUENCE(*queue*[, *class*, ...]): customers are served in the very order that they departed from a reference station (and possibly with the specified class or classes). When a customer passes several times through the reference station, only the first time cares.

concurrency-set can be defined as follows:

- (*class1* [, *prob1*], *class2* [, *prob2*], ..., *classn* [, *probn*]): the concurrency set is defined by customers of classes *class1*, ..., *classn*, possibly with their respective probability (allowing a customer class to belong to different concurrency sets with different probabilities); customers whose class is not cited in the class list of all concurrency sets, or whose class has a sum of related probabilities lower than 1 belong to no concurrency set, this is allowed only with simulation.
- (*prob1*, *prob2*, ..., *probn*): this syntax defines several concurrency sets based on a probabilistic selection of customers. If the sum of the probabilities is lower than 1, a customer can belong to no concurrency set, this is allowed only with simulation.

The concurrency set to which belongs a customer is determined when the customer enters the station. It can be accessed by the CONCSETN function of the CUSTOMER type.

real, *prob1*, ..., *probn* are expressions or constants evaluating to positive REAL.

queue is an indentifier of a declared object of type QUEUE.

class, *class1*, ..., *classn* are identifiers of declared objects of type CLASS.

EVALUATION

- During compilation for class values.
- At initiation time for queue and real values.

WARNING

When using the PS policy with the simulation, the service description cannot contain any synchronisation operation which can affect the service time, like P, JOIN, JOINC, WAIT, WAITAND, WAITOR.

SEE ALSO

SERVICE - QUANTUM

EXAMPLE

```
/DECLARE/ QUEUE q1, q2, q3, q4, q5, q6;
          CLASS c1, c2, c3;

/STATION/ NAME = q1;              & default policy: FIFO
          ...

/STATION/ NAME = q2;
          SCHED = LIFO;
          ...

/STATION/ NAME = q3;
          SCHED = PRIOR, PREEMPT;
          ...

/STATION/ NAME = q4;
          SCHED = RESEQUENCE (q1,c1);
          SERVICE =  EXP(1) ;
          TRANSIT = OUT;
          ....

/STATION/ NAME = q5 ;
          TYPE = MULTIPLE (2) ;
          SCHED = FIFO, FEFS,  EXCLUDE (0.1, 0.2, 0.05, 0.35) ;
          ....

/STATION/ NAME = q6 ;
          TYPE = MULTIPLE (3) ;
          SCHED = FEFS,  EXCLUDE (c1, 0.1, c2, 0.2, c3, 0.05) ;
          ....
```

NAME

SERVICE - Service description parameter.

SYNTAX

SERVICE [(*class1*, *class2*, ...)] = *statement-list*

DESCRIPTION

This parameter describes the service of a station, that is the processing of a customer when it seizes a server of the station.

Several SERVICE specifications are allowed within the same /STATION/ command. If no class is specified, the definition applies to all classes for which no service is defined within the same /STATION/ command.

The processing is described by an algorithmic code (*statement-list*), which can be either a single statement (e.g., call to a work demand procedure) or a complex algorithmic sequence, and can contain:

- assignments,
- conditional and loop statements: IF, FOR, WHILE, ...,
- calls to work demand procedures: CST, EXP, ...,
- calls to modelling procedures and functions, i.e., user defined procedures and functions, customer manipulation procedures and functions, synchronisation procedures and functions, ...,
- calls to I/O procedures and functions,
- calls to character string manipulation procedures and functions,
- calls to graphical procedures and functions,
- call to the save procedure SAVERUN.

class1, *class2*, ... are identifiers of declared objects of type CLASS.

EVALUATION

- During compilation for the class list.
- During execution for the statement list with simulation or the Markovian solver.
- At initiation time for the statement list with analytical solvers.

WARNING

- This parameter should not be defined for stations of type SEMAPHORE and RESOURCE.
- If it is planned to solve the model with an analytical solver, the statement list should only contain a single call to a work demand procedure.
- If it is planned to solve the model with the Markovian solver, the statement list should contain no more than one call to a work demand procedure and no call to a synchronisation

procedure or function.

- As the association of a class to a customer is applied only during the transit of the customer, a SOURCE station should contain only one service description without class specification.

SEE ALSO

Algorithmic language - SAVERUN - SOLVE - SIMUL - MARKOV

EXAMPLE

```
/DECLARE/ QUEUE q1, q2, q3, s;
          CLASS c1, c2, c3;
          REAL r;

/STATION/ NAME = q1;
          SERVICE = EXP (3);  & same service for all classes, wait
                              & for a random period exponentially
                              & distributed around 3

/STATION/ NAME = q2;
          SERVICE = BEGIN     & service of class c1 customers
                      P (s);
                      PRINT ("Semaphore taken");
                      CST (r);
                    END;
          SERVICE (c2, c3) = BEGIN   & service of class c2 and c3 customers
                               SAVERUN ("model");
                               IF CUSTNB (q1) > 3 THEN
                                  TRANSIT (q2)
                               ELSE
                                  TRANSIT (q1);
                             END;
          ...
```

NAME

SPLIT - Customer splitting.

SYNTAX

```
SPLIT [(class1, class2, ...)] =
       (queue11,class11,nb11, ... ,queue1n,class1n,nb1n) [prob1,
       (queue21,class21,nb21, ... ,queue2n,class2n,nb2n) prob2,
       ...
       (queuep1,classp1,nbp1, ... ,queuepn,classpn,nbpn) probp];
```

DESCRIPTION

This parameter describes, in place of a TRANSIT parameter, the behaviour of a customer which finishes its service. In this case, the customer is splitted into several pieces specified by the *(queue, class, nb, ...)* list indicating the destination queue, the destination class and the number of pieces generated. The number of pieces for each queue/class can be omitted and is assumed to be 1. The class specification can be omitted and is assumed to be equal to the class of the origin customer.

As opposed to a FISSION operation, resulting customers keep the information of their common origin, which is necessary for further MATCH of the customers.

class-list specifies the customer classes to which the split applies. All classes for which no particular split is specified are concerned if *class-list* is omitted.

Several split policies can be specified and their related probabilities, *prob1*, *prob2*, ..., allow to choose between them. The probability values are normalised if their sum is not equal to 1.

When the fission is completed, the initial customer is destroyed, after executing a possible trace action.

Each resulting customer can also be splitted further, either by a FISSION or a SPLIT operation. They will maintain their origin information along several SPLIT or FISSION.

queue11, *queue12*, ... are identifiers of declared objects of type QUEUE.

class1, *class2*, ..., *class11*, *class12*, ... are identifiers of declared objects of type CLASS.

nb11, *nb12*, ... are expressions or constants evaluating to positive INTEGER.

prob1, *prob2*, ... are expressions or constants evaluating to REAL.

EVALUATION

At initiation time.

NOTES

As opposed to the FISSION parameter, the SPLIT one can also be used with the analytical solvers.

WARNING

As a SPLIT is equivalent to a TRANSIT or a FISSION, these parameters cannot be combined within the same /STATION/ command.

SEE ALSO

FISSION - MATCH - FUSION - SPLITMAT

EXAMPLE

```
/DECLARE/ QUEUE inter1, inter2, inter3;
          CLASS c1,c2;

/STATION/ NAME = inter1;
          SPLIT(c1) = (inter2,c2,2,inter3,c2,2) 0.4,
                      (inter2,c1,2,inter3,c1,2) 0.6;
          SERVICE = EXP (1.);

/STATION/ NAME = inter2;
          TRANSIT =OUT;
          MATCH = (inter1):(c1,2) c1;
          SERVICE = EXP (2.);
```

NAME

TRANSIT - Customer routing parameter.

SYNTAX

TRANSIT = *routing* ;
TRANSIT (*class1*, *class2*, ...) = *routing* ;

Where *routing* can be:

- *queue* [, *class*]
- *queue1* [, *class1*], *proba1*, *queue2* [, *class2*], *proba 2*, ..., *queuen* [, *classn*]
- *queue1* [, *class1*], *weight1*,*queue2* [, *class2*], *weight2*, ..., *queuen* [, *classn*], *weightn*

DESCRIPTION

This paramater describes the routing policy of customers which have finished their service in the station.

Several TRANSIT specifications are allowed within the same /STATION/ command. If no class is specified, the definition applies to all classes for which no transit policy is defined within the same /STATION/ command.

The routing is determined either by a single queue or queue/class specification, or by a list of queues, or queues/class, and probability or weight associations. If the sum of all probabilities is greater than 1, weight are assumed. Otherwise, the last probability value can be omitted as it will be computed as 1 minus the sum of all other probabilities. In case of weights, all values are necessary and probabilities are computed by dividing the weight by the sum of all weights.

class1, *class2*, ... are identifiers of declared objects of type CLASS.

queue1, *queue2*,... are identifiers of declared objects of type QUEUE.

prob1, *prob2*,..., *weight1*, *weight2*,... are expressions or constants evaluating to REAL.

NOTES

- The OUT predefined queue can be used to specify the exit of the model.
- As the association of a class to a customer is applied only during the transit of the customer, a SOURCE station must specify class associations if the model is multi classes, i.e., more than one class are declared, and there should be only one TRANSIT parameter without class specification on the left-hand side of the parameter.

EVALUATION

- During compilation for the class list.
- At initiation time for the routing description.

WARNING

- This parameter should not be defined for stations of type SEMAPHORE and RESOURCE.
- As the routing description is evaluated at initiation time, reference to QUEUE must be assigned a QUEUE at that time if it is used in the routing description, and the routing will statically refer to the value of the reference at initiation time.
- If no service is specified in the station, the TRANSIT command is ignored.

SEE ALSO

SERVICE - TRANSIT (Procedure) - MOVE - BEFCUST - AFTCUST

EXAMPLE

```
/DECLARE/ QUEUE q1, q2, q3;
          CLASS c1, c2, c3;
          REAL p1, p2;

/STATION/ NAME = q1;
          TRANSIT = q2; & routing toward q2 with the same class
          ...

/STATION/ NAME = q2;
          TRANSIT (c1) = q2, c2;      & routing back with class c2 for
                                      & class c1 customers
          TRANSIT = q1, 1, q3, c3, 1; & routing toward q1 with same class
                                      & or toward q3 with class c3
                                      & with the same probability
                                      & for class c2 or c3 customers
          ...

/STATION/ NAME = q3;
          TRANSIT = q1, p1, q2, p2, OUT; & routing toward q1 with
                                         & probability p1, toward q2
                                         & with probability p2 and
                                         & toward OUT with
                                         & probability 1-p1-p2
          ...
```

NAME

TYPE - Station type parameter.

SYNTAX

TYPE = SERVER [, SINGLE | MULTIPLE (*integer*) | INFINITE]
TYPE = RESOURCE [, SINGLE | MULTIPLE (*integer*) | INFINITE]
TYPE = SEMAPHORE [, SINGLE | MULTIPLE (*integer*)]
TYPE = SOURCE ;
TYPE = SINGLE | MULTIPLE (*integer*) | INFINITE

DESCRIPTION

This parameter specifies the type of the station. Available types are:

- SERVER: a processing station made of one or several units:
  - SINGLE: the default type, a single unit,
  - MULTIPLE(...): several units,
  - INFINITE: an infinity of units;
- RESOURCE: passive station made of resource units managed by P, V and FREE procedures. The number of resource units is determined by:
  - SINGLE: a single unit,
  - MULTIPLE(...): several units,
  - INFINITE: an infinity of units;
- SEMAPHORE: passive synchronisation station made of a passing counter managed by P, V and FREE procedures. The counter value is determined by:
  - SINGLE: counter = 1,
  - MULTIPLE(...): counter = multiplicity argument;
- SOURCE: customer creation station.

integer is an expression or constant evaluating to INTEGER.

EVALUATION

At initiation time.

NOTES

- This is not a mandatory parameter. The default type of a station is SERVER, SINGLE.
- If the type is not specified, it is assumed to be SERVER.
- If the multiplicity is not specified, it is assumed to be SINGLE.

SEE ALSO

P(resource) - P(semaphore) - V(resource) - V(semaphore)

EXAMPLE

```
/DECLARE/ QUEUE q1, q2, q3, q4, q5;
          INTEGER i;

/STATION/ NAME = q1;                          & single server station
          ...

/STATION/ NAME = q2;                          & infinite server station
          TYPE = INFINITE;                    & customers never wait

/STATION/ NAME = q3;
          TYPE = RESOURCE, MULTIPLE (2);      & 2 resource units station
          ...

/STATION /NAME = q4;
          TYPE = SEMAPHORE, MULTIPLE (2*i); & semaphore station, counter = 2*I
          ...

/STATION /NAME = q5;
          TYPE = SOURCE;                      & source station
          ...
```

Chapter 3

CONTROL command

ACCURACY	Definition of queues and classes for which confidence intervals are computed during simulation.
ALIAS	Definition of alias names.
CLASS	Definition of queues for which results per class are requested.
CONVERGENCE	Definition of mathematical solvers control parameters.
CORRELATION	Definition of queues and classes for which auto-correlation functions are requested.
ENTRY	Definition of the algorithmic code sequence which will be executed before resolution starts.
ESTIMATION	Selection of the method used for confidence intervals computation.
EXIT	Definition of the algorithmic code sequence which will be executed after each resolution.
MARGINAL	Definition of queues and classes for which marginal probabilities are computed.
NMAX	Definition of the maximum number of classes in the network.
OPTION	Definition of options to control execution.
PERIOD	Definition of the test sequence activation period.
RANDOM	Definition of the pseudo-random generator seed.
STATISTICS	Definition of periodic results type.
TEST	Definition of the algorithmic code sequence which will be executed periodically during the simulation.
TMAX	Definition of the maximum simulation run length.
TRACE	Definition of the trace activation period and format.
TSTART	Definition of the measurements starting date in simulation.
UNIT	Definition of input-output standard files.

NAME

ACCURACY - Definition of queues and classes for which confidence intervals are computed during simulation.

SYNTAX

ACCURACY = *queue* [, *class*] ;
ACCURACY = *queue1* [, *class1*], *queue2* [,*class2*], ... ;
ACCURACY = ALL QUEUE [,ALL CLASS] ;
ACCURACY = NIL ;

DESCRIPTION

This parameter allows to specify a list of queues and classes for which confidence intervals will be computed during the statistical measurements period of a simulation.

Performance criteria and functions to access results are the following:

- service time (CSERVICE)
- number of customers (CCUSTNB)
- busy percentage of servers (CBUSYPCT)
- response time (CRESPONSE)
- throughput (CTHRUPUT)
- blocked time (CBLOCKED)

The elements of list can be variables or expressions. A general request of confidence intervals computation is done using ALL QUEUE both with ALL CLASS if results per class are requested. To remove all the requests registered during a previous simulation, this parameter must be used with the NIL value.

The confidence intervals estimation method can be chosen using the ESTIMATION parameter. The spectral method is chosen by default.

queue, queue1 and *queue2* are expressions representing QUEUE type entities.

class, class1 and *class2* are expressions representing CLASS type entities.

EVALUATION

At the beginning of the resolution.

WARNING

- This parameter is not taken into account by the analytical solvers and the markovian solver.
- A (*queue, class*) couple can be requested only if results per class have been requested for the queue (CLASS *parameter*).

SEE ALSO

CLASS - ESTIMATION - SIMUL

EXAMPLE

```
/DECLARE/ QUEUE q1, q2, q3 ;
          CLASS c1, c2, c3 ;
  ...
/CONTROL/ ACCURACY = q1 ;
/CONTROL/ ACCURACY = q1, ALL CLASS ;
/CONTROL/ ACCURACY = q1, c2 ;
/CONTROL/ ACCURACY = q1, c1, q2, c2, q3 ;
/CONTROL/ ACCURACY = ALL QUEUE ;
/CONTROL/ ACCURACY = ALL QUEUE, c1 ;
/CONTROL/ ACCURACY = ALL QUEUE, ALL CLASS ;
```

NAME

ALIAS - Definition of alias names.

SYNTAX

ALIAS = (*new_name*, *old_name*) ;
ALIAS = (*new_name1*, *old_name1*), (*new_name2*, *old_name2*), ... ;

DESCRIPTION

This parameter allows to add new names to already existing identifiers and keywords. New names thus defined take inevitably the *old_name* initial meaning.

EVALUATION

During the compilation.

NOTES

Several aliases may be assigned to a given identifier or keyword.

WARNING

- Aliases are not possible with reserved keywords: functions, procedures or predefined attributes.
- An alias cannot be defined using an alias name itself.

SEE ALSO

Reserved keywords.

EXAMPLE

```
/DECLARE/ QUEUE q1, q2 ;
...
/CONTROL/ ALIAS = (NAME,NAME),(GESTION,SCHED),(PAPS,FIFO),(DADS,FIFO),(Q,q1);
...
/CONTROL/ ALIAS = (ROUTAGE,TRANSIT);
...
/STATION/
        NAME = Q ;            & <==> NAME = q1
        GESTION = PAPS ;      & <==> SCHED = DADS <==> SCHED = FIFO
        ROUTAGE = q2 ;        & <==> TRANSIT = q2
        ...
/STATION/
         NAME = q2 ;           & <==> NAME = q2
         GESTION = DADS ;     & <==> SCHED = PAPS <==> SCHED = FIFO
         ...
```

NAME

CLASS - Definition of queues for which results per class are requested.

SYNTAX

CLASS = *queue* ;
CLASS = *queue1*, *queue2*, ... ;
CLASS = **ALL QUEUE**;
CLASS = **NIL**;

DESCRIPTION

This parameter allows to request results per class for a list of queues.

These results per class are not computed by default. A general request for all the network queues is done using ALL QUEUE. To remove requests registered during a previous simulation, the parameter CLASS must be used with the NIL value.

queue, queue1 and *queue2* are expressions representing QUEUE type entities.

EVALUATION

At the beginning of the resolution.

WARNING

This parameter is now replaced by the SETSTAT:CLASS procedure and will no longer be maintained.

SEE ALSO

SETSTAT:keyword

EXAMPLE

```
/DECLARE/ QUEUE q1, q2, q3, q4 ;
          CLASS c1, c2 ;

/CONTROL/ CLASS = q1 ;        & results per class request
...                           & for q1 queue only
/CONTROL/ CLASS = q2, q3 ;    & results per class request
...                           & for q2 and q3 queues
/CONTROL/ CLASS = ALL QUEUE;  & results per class request for all queues
/CONTROL/ CLASS = NIL;        & remove the results per class request
```

NAME

CONVERGENCE - Definition of mathematical solvers control parameters.

SYNTAX

CONVERGENCE = *integer1, real1, integer2, real2*;

DESCRIPTION

This parameter allows to modify default values of some mathematical solvers parameters (SOLVE and MARKOV).

The meaning of these four parameters are the following:

- *integer1* : maximum number of iterations,
- *real1* : precision of the convergence test,
- *integer2* : number of test vectors used in the markovian solver,
- *real2* : ratio between the number of non-null entries in the states tansition matrix and the number of states in the markovian solver.

integer1 and *integer2* are expressions representing INTEGER type entities.

real1 et *real2* are expressions representing REAL type entities.

EVALUATION

At the beginning of the resolution.

NOTE

Default values are CONVERGENCE = 100, 1E-6, 10, 5.0 ;

EXAMPLE

```
/DECLARE/ INTEGER i = 5 ;
          REAL r = 1E-6 ;

/CONTROL/ CONVERGENCE = 500, r, i, 5.0 ;
```

CORRELATION

NAME

CORRELATION - Definition of queues and classes for which auto-correlation functions are requested.

SYNTAX

CORRELATION = *queue* [, *class*] [, *integer*] ;
CORRELATION = *queue1* [, *class1*] [,*integer1*], *queue2* [,*class2*] [, *entier2*]... ;
CORRELATION = **ALL QUEUE** [,**ALL CLASS**] [, *integer*] ;
CORRELATION = **NIL**;

DESCRIPTION

This parameter allows to specify a list of queues and classes for which auto-correlation functions will be computed in order to estimate the validity of confidence intervals generated by the regeneration method (especially the independence assumption made on these measurements).

A general request is done using ALL QUEUE both with ALL CLASS if results per class are requested. To remove all the requests registered during a previous simulation, this parameter must be used with the NIL value. The maximum order of the auto-correlation functions may be specified by an integer expression. It must be lower than 21.

queue, queue1 and *queue2* are expressions representing QUEUE type entities.

class, class1 and *class2* are expressions representing CLASS type entities.

integer, integer1 and *integer2* are expressions representing INTEGER type entities.

EVALUATION

At the beginning of the resolution.

NOTES

- No correlation is required by default (CORRELATION = NIL).
- The default value for the auto-correlation order is 5.

WARNING

- This parameter is not taken into account by the analytical solvers and the markovian solver.
- This parameter is available only when the confidence intervals are computed using the regeneration method.
- Queues and classes in the list must also appear in the ACCURRY parameter definition (request of confidence intervals).
- This parameter is now replaced by the SETSTAT:CORRELATION procedure and will no longer be maintained.

SEE ALSO

ESTIMATION - SIMUL - SAMPLE - ACCURACY - SETSTAT:keyword

EXAMPLE

```
/DECLARE/ QUEUE q1, q2, q3 ;
          CLASS c1, c2, c3 ;
          INTEGER i ;
  ...

/CONTROL/ ESTIMATION = REGENERATION ;
          PERIOD = 100 ;
          TEST = IF CUSTNB (q1) + CUSTNB (q2) = 0 THEN
                 BEGIN
                   SAMPLE ;
                   OUTPUT;
                 END;
          CORRELATION = q1 ;
...
/CONTROL/ CORRELATION = q1, c2, i ;
...
/CONTROL/ CORRELATION = q1, q2, c3 ;
...
/CONTROL/ CORRELATION = ALL QUEUE ;
```

ENTRY

NAME

ENTRY - Definition of the algorithmic code sequence which will be executed before resolution starts.

SYNTAX

ENTRY = *statement* ;
ENTRY =
BEGIN
statements sequence
END ;
ENTRY = ;

DESCRIPTION

This parameter allows to specify a QNAP2 algorithmic sequence which will be executed before each network analysis (just before resolution starts). The algorithmic sequence may contain variable initialisations.

To remove previous definitions of the ENTRY parameter, it must be used without any argument.

EVALUATION

At the beginning of the resolution.

NOTES

- No algorithmic sequence is executed by default.
- If the replication method is used to compute confidence intervals, the sequence is executed once only at the beginning of the first replication.

WARNING

The sequence cannot contain work requests, synchronisations or modeling mechanisms. Especially, SET and RESET operations on FLAG objects are forbidden.

SEE ALSO

EXIT - TIMERS - EXCEPTION

EXAMPLE

```
/DECLARE/ QUEUE q1, q2, q3 ;
          INTEGER i ;
          REAL r = 3.0 ;

  ...

/STATION/ NAME = q1;
          TYPE = SOURCE ;
          INIT = i ;
  ...
/CONTROL/ ENTRY = IF r > 10 THEN i := 2
                           ELSE i := 1 ;
...
/EXEC/ SIMUL ;                                & Value for i will be 1
                                              & (r = 3.0 by default)
...
/EXEC/ FOR r : = 5.0, 15.0 ,30.0 DO SOLVE ;   & Value for i will be
...                                           & successively 1, 2 and 2
```

ESTIMATION

NAME

ESTIMATION - Selection of the method used for confidence intervals computation.

SYNTAX

ESTIMATION = SPECTRAL [(*real*)] ;
ESTIMATION = REGENERATION ;
ESTIMATION = REPLICATION (*integer*) ;

DESCRIPTION

This parameter allows to define the method to use for confidence intervals computation during simulation.

The spectral method is used for steady state studies. The real parameter defines the first measurements period. Its value may be an expression or a constant. Its default value is TMAX/512.

The regeneration method is used for steady state studies. Explicit calls to the SAMPLE procedure in a service description or in the TEST sequence must appear to specify a regeneration point.

The replication method is used for transient state studies. The integer parameter specifies the number of replications (replicated simulations). The duration of each replication is defined by the TMAX parameter.

real is an expression representing a REAL type entity.

integer is an expression representing an INTEGER type entity.

EVALUATION

At the beginning of the resolution.

NOTES

- This parameter is mandatory to get the confidence intervals.
- Each replication produces a specific behavior because the seed of the pseudo-random numbers generator is not reinitialized between each replication.

WARNINGS

- This parameter is not taken into account by the analytical solvers and the markovian solver.
- Regeneration points defined by the SAMPLE procedure must delimitate independent samples.

SEE ALSO

CLASS - ESTIMATION - SIMUL - RANDOM

EXAMPLE

```
/DECLARE/ REAL r ;
  ...
/CONTROL/ ESTIMATION = SPECTRAL ;
 ...
/CONTROL/ ESTIMATION = SPECTRAL (r) ;
 ...
/CONTROL/ PERIOD = 100 ;
          TEST = IF CUSTNB (q1) + CUSTNB (q2) = 0 THEN SAMPLE ;
          ESTIMATION = REGENERATION ;
 ...
/CONTROL/ ESTIMATION = REPLICATION (5) ;
```

EXIT

NAME

EXIT - Definition of the algorithmic code sequence which will be executed after each resolution.

SYNTAX

EXIT = *statement* ;
EXIT =
BEGIN
statements sequence
END ;
EXIT = ;

DESCRIPTION

This parameter allows to specify the QNAP2 algorithmic sequence which will be executed after each resolution. The algorithmic sequence may compute additional results (using standard results) or perform some printing.

To remove previous definitions of EXIT parameter, it must be used without any argument.

EVALUATION

At the beginning of the resolution.

NOTES

- No algorithmic sequence is executed by default.
- If the replication method is used to compute confidence intervals, the sequence is executed once only at the end of the last replication.

WARNING

The sequence cannot contain work requests, synchronisations or modeling mechanisms.

SEE ALSO

ENTRY - TIMERS - EXCEPTION

EXAMPLE

```
/DECLARE/ QUEUE q1, q2, q3 ;
          REAL r ;
  ...
/CONTROL/ EXIT =
          BEGIN
            PRINT ("End of resolution") ;
            PRINT ("q1 response time : ", 100*MRESPONSE (q1)) ;
          END ;
...
/EXEC/ SIMUL ;
...
/EXEC/ FOR r : = 5.0, 15.0, 30.0 DO SOLVE ;
...
& Messages defined in the sequence will be displayed four times
& for each  description.
```

MARGINAL

NAME

MARGINAL - Definition of queues and classes for which marginal probabilities are computed.

SYNTAX

MARGINAL = *queue* [, *order*] ;
MARGINAL = *queue1* [, *order1*], *queue2* [,*order2*], ... ;
MARGINAL = **ALL QUEUE**;
MARGINAL = **NIL**;

DESCRIPTION

This parameter allows to specify queues and classes for which marginal probabilities are computed. The marginal probabilities are the probabilities that these queues contain exactly 0, 1, 2 ,..., n customers.

Marginal probabilities per class are computed only if results per class have been requested for the queue (see CLASS parameter).

The optional integer defines the order (maximum number of customers) up to which these probabilities have to be computed. The default value for the order is 5.

queue, queue1 and *queue2* are expressions representing QUEUE type entities.

order, order1 and *order2* are expressions representing INTEGER type entities.

EVALUATION

During the compilation.

NOTES

- No marginal probabilities are computed by default of this parameter.
- Marginal probabilities intervene in one of the mean customers number formula computation: $MCUSTNB(q) = \sum_{i=0}^{\infty} i.p(i)$

WARNING

This parameter is now replaced by the SETSTAT:MARGINAL procedure and will no longer be maintained.

SEE ALSO

MCUSTNB - PCUSTNB - CLASS - SETSTAT:keyword

EXAMPLE

```
/DECLARE/ QUEUE q1, q2 ;
          CLASS c1, c2 ;
  ...
/CONTROL/ MARGINAL = ALL QUEUE ;    & marginal probabilities computed
...                                 & for all queues up to the 5th order

/CONTROL/ MARGINAL = q1, 3, q2 ;    & marginal probabilities computed
...                                 & for q1 up to the 3rd order and
                                    & for q2 up to the 5th order

/CONTROL/ CLASS = ALL QUEUE ;
          MARGINAL = q1 ;           & marginal probabilities computed
                                    & per class for q1 up to the 5th order
```

NAME

NMAX - Definition of the maximum number of classes in the network.

SYNTAX

NMAX = *integer* ;

DESCRIPTION

This parameter allows to specify the maximum number of classes in the network.

integer is an expression representing INTEGER type entity. *integer* must be grater than zero.

EVALUATION

During the compilation.

NOTES

The maximum number of classes is set to 20 by default.

WARNING

This parameter must appear before any queue or class declaration.

SEE ALSO

CLASS

EXAMPLE

```
/DECLARE/ INTEGER i = 30 ;
  ...
/CONTROL/ NMAX = i ;                & the maximum number of classes is 30

/DECLARE/ CLASS c1 (25), c2 (5) ;   & declarations of queues and classes
          QUEUE q1, q2, q3 ;        & take place after the NMAX parameter
...
```

NAME

OPTION - Definition of options to control execution.

SYNTAX

OPTION = *option* ;
OPTION = *option1*, *option2*, ... ;

where option can be one of the following:

- DEBUG | NDEBUG
- SOURCE | NSOURCE
- RESULT | NRESULT
- TRACE | NTRACE
- VERIF | NVERIF
- WARN | NWARN
- SIMPAR | NSIMPAR

DESCRIPTION

The control of a QNAP2 execution is specified using options manipulated as ON/OFF switches. A default ON/OFF value is associated to each available option.

The requested action will be effective from the next QNAP2 statement up to an opposite request (else up to the end of the execution).

Options have the following meanings:

- DEBUG, NDEBUG: Compilation (or not) of the algorithmic code allowing to use the QNAP2 debugging tool.
- SOURCE, NSOURCE: Echo (or not) of the QNAP2 program lines during compilation and of data entered using FSYSGET.
- RESULT, NRESULT: Printing (or not) of standard results. Note that these results can be accessed using functions.
- TRACE, NTRACE: Printing (or not) of intermediate results in the case of mathematical methods; printing (or not) of the simulation event trace.
- VERIF, NVERIF: Activation (or not) of the consistency checks during execution. NVERIF may be used with tested models to reduce execution times.
- WARN, NWARN: Printing (or not) of QNAP2 warning messages.
- SIMPAR, NSIMPAR: Activation (or not) of the parallelization of replications (if it is allowed). Refer to QNAP2 User's Guide to use it.

EVALUATION

During the compilation.

NOTES

Defaults options are the following:

- NDEBUG
- SOURCE
- RESULT
- NTRACE
- VERIF
- WARN
- NSIMPAR (if parallelization is allowed)

To make the debugger available, a break-point must be inserted in the algorithmic code of the model. Note that the DEBUG option has no effect if no break-point is defined.

WARNINGS

- The DEBUG option must be used in simulation only.
- The debugger can only be used in interactive mode.
- Refer to the User's Guide to use OPTION=SIMPAR;

SEE ALSO

TRACE - SETTRACE - FSYSGET - HALT

EXAMPLE

```
/CONTROL/ OPTION = NSOURCE,  & During compilation, following lines
          NRESULT;           & will not appear. Standard results
                             & will not be displayed,
/DECLARE/ QUEUE q ;
...
/CONTROL/ OPTION = SOURCE ;  & During compilation, following lines
                             & will appear.
...
```

NAME

PERIOD - Definition of the test sequence activation period.

SYNTAX

PERIOD = *real* ;
PERIOD = ;

DESCRIPTION

This parameter allows to specify the period delay between two TEST sequence activations. The QNAP2 test sequence code is defined using the TEST parameter.

real is an expression representing a REAL type entity. The value must be greater than zero. If the value is equal to zero, the test sequence will be executed at each event occurence. If the parameter is specified without value, then the test sequence will never be run during simulation.

EVALUATION

During the resolution.

NOTES

By default of this parameter, the test sequence will never be executed.

WARNING

This parameter is not taken into account by the analytical solvers and the markovian solver.

SEE ALSO

TEST - TIMER - SIMUL

EXAMPLE

```
/DECLARE/ REAL r ;
  ...
/CONTROL/ PERIOD = 100 ;
          TEST = IF CUSTNB (q1) > 100 THEN STOP ;

/EXEC/ SIMUL ;
...
/CONTROL/ PERIOD = r ;
...
```

RANDOM

NAME

RANDOM - Definition of the pseudo-random generator seed.

SYNTAX

RANDOM = *integer* ;

DESCRIPTION

Numbers generated by mathematical functions are produced using a pseudo-random numbers generator.

The seed is the initial value from which all the random drawings will be based.

integer is an expression representing an INTEGER type entity.

EVALUATION

At the beginning of the resolution.

NOTES

- The seed default value is 413.
- The generator is reinitialized at the beginning of each /EXEC/ block. As a consequence, if several simulations are launched within one /EXEC/ block, the generator continues from the state resulting of the previous simulation.

SEE ALSO

RANDU - EXP - HEXP - ERLANG - NORMAL - COX - UNIFORM - RINT - DRAW - DISCRETE - HISTOGR

EXAMPLE

```
/CONTROL/ RANDOM = 100 ;
/EXEC/ SIMUL ;
...
/DECLARE/ INTEGER i = 556 ;
/CONTROL/ RANDOM = i ;

/EXEC/ BEGIN
         SIMUL ;  & the both simulations will have different behaviours
         SIMUL ;  & because the generator is not reinitalized
       END ;      & between them.
...
```

NAME

STATISTICS - Definition of periodic results type.

SYNTAX

```
STATISTICS = GLOBAL;
STATISTICS = PARTIAL;
```

DESCRIPTION

This parameter allows to specify the type of periodic results.

The GLOBAL keyword means that periodic computed results are produced from measurements starting date (TSTART parameter) up to the last regeneration point.

The PARTIAL means that periodic computed results are produced from measurements issued of the last regeneration period.

After the end of the simulation, results are produced according to the GLOBAL keyword definition. Results can be accessed using the OUTPUT procedure (request the display of standard results) or the results access functions.

EVALUATION

At the beginning of the resolution.

NOTES

By default of these parameters, results are produced according to the GLOBAL keyword definition.

WARNING

- This parameter is only available when the regeneration method is requested to compute confidence intervals.
- This parameter is not taken into account by the analytical solvers and the markovian solver.
- This parameter is now replaced by the SETSTAT:PARTIAL procedure and will no longer be maintained.

SEE ALSO

ESTIMATION - SIMUL - SAMPLE - OUTPUT - TSTART - SETSTAT:keyword

EXAMPLE

```
   ...
/CONTROL/ ESTIMATION = REGENERATION ;
          PERIOD = 100 ;
          TEST = IF CUSTNB (q1) + CUSTNB (q2) = 0 THEN BEGIN
                                                         SAMPLE ;
                                                         OUTPUT ;
                                                       END ;
          STATISTICS = PARTIAL ;

/EXEC/ SIMUL ;  & each call to OUTPUT produces results
...             & concerning the last regeneration period
...             & (between the two last calls to SAMPLE)
...
/CONTROL/ STATISTICS = GLOBAL ;

/EXEC/ SIMUL ; & each call to OUTPUT produces results
...            & concerning the  period between measurements starting
...            & date (TSTART = 0 by default) and the last
...            & regeneration point
```

NAME

TEST - Definition of the algorithmic code sequence which will be executed periodically during the simulation.

SYNTAX

TEST = *statement* ;
TEST =
BEGIN
statements sequence
END ;
TEST = ;

DESCRIPTION

This parameter allows to specify an QNAP2 algorithmic code sequence which will be executed periodically during the simulation. Period duration is defined using the PERIOD parameter.

The algorithmic code may contain:

- the definition of a regeneration point (call to SAMPLE),
- the request to intermediate results display (call to OUTPUT or to the results access functions),
- a simulation duration control statement (call to STOP).

All these actions may be conditionned by tests on the current network state.

To remove the effect of a test sequence previously defined, the TEST parameter must be used without any argument.

EVALUATION

At the end of the period.

NOTES

- During markovian resolution, the test sequence is activated at each state change.
- This parameter is not taken into account by the analytical solvers.

WARNINGS

- The sequence cannot contain work requests, synchronisations or modeling mechanisms.
- During simulation, the test sequence cannot contain calls to TRANSIT, MOVE, BEFCUST and AFTCUST procedures

SEE ALSO

PERIOD - TIMER - SETTRACE - EXCEPTION - SIMUL - MARKOV

EXAMPLE

```
   ...
/CONTROL/ PERIOD = 100 ;
          TEST = IF CUSTNB (q1) > 100 THEN STOP ;

/EXEC/ SIMUL ;
...
/CONTROL/ TEST = IF CUSTNB (q1) > 10 THEN MOVE (q1, q2) ;

/EXEC/ MARKOV ;
...
```

NAME

TMAX - Definition of the maximum simulation run length.

SYNTAX

TMAX = *real* ;

DESCRIPTION

This parameter allows to specify the maximum simulation run length. The duration is expressed in time units in accordance with those used in the model. The real parameter defines the date of simulation end.

real is an expression representing a REAL type entity.

EVALUATION

At the beginning of the resolution

NOTE

- The default value is 0.
- The current date can be obtained by using the real variable TIME.

WARNINGS

- This parameter is only available for a resolution by simulation.
- This parameter is essential in simulation because the default value inhibits the simulation.
- The value must be greater than or equal to zero.

SEE ALSO

TIMER - SETTIMER - SETTMAX

EXAMPLE

```
/CONTROL/ TMAX = 100 ;

/EXEC/ SIMUL ;
...
/DECLARE/ REAL t = 123.5 ;
/CONTROL/ TMAX = t ;

/EXEC/ FOR t : = 1000, 1030, 1050 DO SIMUL ;
```

TRACE

NAME

TRACE - Definition of the trace activation period and format.

SYNTAX

TRACE = *real* [, "L80" — "L132"] ;
TRACE = *real1*, *real2* [, "L80" — "L132"] ;

DESCRIPTION

This parameter allows to specify the output format and the period according which the event trace will be produced.

The first real value defines the trace starting date (*real* and *real1*).
The second value (if someone) defines the trace ending date (*real2*).

The ouput format is defined by either the "L80" keyword if trace is displayed on 80 columns, either the "L132" keyword if trace is displayed on 132 columns.

Real values may be defined by a constant or an expression. They must be greater than zero.

real, *real1* and *real2* are expressions representing REAL type entities.

EVALUATION

At the beginning of the resolution.

NOTES

- By default of trace ending date, the maximum simulation duration is assumed (TMAX parameter).
- The output is displayed on 80 columns by default.

WARNINGS

- This parameter is not taken into account by the analytical solvers and the markovian solver.
- Starting and ending dates are greater than zero.
- Ending date is greater than starting date.
- This parameter is now replaced by the SETTRACE:ON procedure and will no longer be maintained.

SEE ALSO

TMAX - SIMUL - OPTION - FSYSTRAC - SETTRACE:keyword

EXAMPLE

```
/DECLARE/ REAL r ;
  ...
/CONTROL/ TRACE = 100, 200 ;

/EXEC/ SIMUL ;
...
/CONTROL/ TRACE = r, "L132" ;
...
```

TSTART

NAME

TSTART - Definition of the measurements starting date in simulation.

SYNTAX

TSTART = *real* ;

DESCRIPTION

This parameter allows to specify the date when the measurements on the simulated model are started. The date is expressed in time units in accordance with those used in the model. The real parameter defines the measurements starting date.

This parameter is useful in steady-state studies; it allows to reduce the bias due to the transient behavior of the simulated model.

real is an expression representing a REAL type entity.

EVALUATION

At the beginning of the resolution.

NOTE

- The default value is 0.
- The current date can be obtained by using the real variable TIME.

WARNINGS

- This parameter is only available for a resolution by simulation.
- The value may not be lower than the maximum simulation duration if statistics must be produced.
- The value must be greater than zero.

SEE ALSO

TMAX - TIMER - SETTIMER

EXAMPLE

```
/CONTROL/ TSTART = 100 ;
...
/EXEC/ SIMUL ;
...
/DECLARE/ REAL t = 123.5 ;
/CONTROL/ TSTART = t ;
...
/EXEC/ SIMUL ;
```

NAME

UNIT - Definition of input-output standard files.

SYNTAX

UNIT = *type* (*file*) ;
UNIT = *type1* (*file1*), *type2* (*file2*), ... ;

where type can be one of the following:

- OUTPUT
- PRINT
- INPUT
- GET
- LIBR
- TRACE

DESCRIPTION

Standard logical unit types define the different types of input/outut mechanisms. These types are the following:

- OUTPUT : Echo of the compilator, error messages and standard results. The default logical unit is FSYSOUTPT
- PRINT: Printing of output produced by the PRINT, WRITE and WRITELN procedures. The default logical unit is FSYSPRINT
- INPUT : Entry of the source program to be analyzed. The default logical unit is FSYSINPU.
- GET: Entry of data read using the GET and GETLN functions. The default logical unit is FSYSGET.
- LIBR: Management of SAVE, SAVERUN and RESTORE functions. The default logical unit is FSYSLIB.
- TRACE : Output of the trace. The default logical unit is FSYSTRAC.

The requested action will be effective from the next QNAP2 statement up to the next change of logical unit (else up to the end of the execution).

The file defines by argument may be a FILE type variable or a pointer to FILE type. This file must be assigned using the FILASSIGN procedure and not opened when the statement is compiled.

EVALUATION

During the compilation.

SEE ALSO

FSYSGET - FSYSOUTPT - FSYSPRINT - FSYSTRAC - FSYSLIBR - FSYSINPU - GET - GETLN - PRINT - WRITE - WRITELN - FILE - FILASSIGN

EXAMPLE

```
/DECLARE/ FILE f1, f2 ;
/EXEC/ BEGIN
         FILASSIGN (f1, "file1.txt") ;
         FILASSIGN (f2, "file2.txt") ;
         OPEN(f1,3);
         OPEN(f2,3);
       END ;

/EXEC/ BEGIN
         PRINT ("This message will be displayed on screen or in a file") ;
         PRINT ("according to the prolog specification") ;
       END ;

/CONTROL/ UNIT = PRINT (f1) ;
/EXEC/ PRINT ("This message will take place in the file1.txt file") ;

/CONTROL/ UNIT = PRINT (f2) ;
/EXEC/ PRINT ("This message will take place in the file2.txt file") ;
...
```

Chapter 4

Type of Traced Events.

"AFTCUST"	Associated to the `AFTCUST` procedure.
"BEFCUST"	Associated to the `BEFCUST` procedure.
"BLOCK"	Associated to the `BLOCK` procedure.
"ENDSERV"	End of service and transit of the customer to the next queue (or `OUT`).
"ENDSTART"	End of `TSTART` period.
"FISSION"	`FISSION` of a customer at queue departure.
"FREE"	Associated to the `FREE` procedure.
"FREEFLAG"	Freeing of a customer blocked on a flag.
"FREEP"	Freeing of a customer blocked on a semaphore or a resource.
"FUSION"	`FUSION` of a set of customers at a queue entry.
"INIT"	Initial creation of customers in a queue.
"JOINALL"	Associated to the `JOIN` or `JOINC` procedure on all sons customers.
"JOINEND"	End of customer waiting on a `JOIN` or `JOINC` procedure.
"JOINLIST"	Associated to a `JOIN` or `JOINC` procedure on a list of customers.
"JOINNB"	Associated to the `JOIN` or `JOINC` procedure on number of customers.
"MATCH"	`MATCH` of a set of customers at a queue entry.
"MOVE"	Associated to the `MOVE` procedure.
"NEWCUST"	Associated to the `NEW`(*CUSTOMER*) function.
"P"	Associated to the `P` procedure on a resource or a semaphore.
"PMULT"	Event associated to the `PMULT` procedure.
"PRIOR"	Associated to the `PRIOR` procedure.
"RESET"	Associated to the `RESET` procedure on a `FLAG` object.
"SERVTIME"	Request of a service time.
"SET"	Associated to the `SET` procedure on a `FLAG` object.
"SOURCE"	Customer creation in a source station.
"SPLIT"	`SPLIT` of a customer at queue departure.
"TMRCANCL"	Cancel of a timer by the user.
"TMRSETTM"	Launching of a timer by the user.
"TMRWAKUP"	Starting of timer execution.
"TRANSIT"	Associated to the `TRANSIT` procedure.
"UNBLOCK"	Associated to the `UNBLOCK` procedure.
"V"	Associated to the `V` procedure on resource or semaphore.
"VMULT"	Event associated to the `VMULT` procedure.
"WAIT"	Associated to the `WAIT` procedure on a `FLAG` object.
"WAITAND"	Associated to the `WAITAND` procedure on `FLAG` objects.
"WAITOR"	Associated to the `WAITOR` procedure on flag objects.

NAME

"AFTCUST" - Associated to the AFTCUST procedure.

DESCRIPTION

- GETTRACE:CPROVOKE, GETTRACE:EXCEPTPROVOKE or GETTRACE:TIMERPROVOKE return respectively (according to the result returned by GETTRACE:WHICHPRO) the customer, the exception or the timer which triggers off the operation.

- GETTRACE:CSUBJECT returns the customer which has been transited. If it has been transited to OUT, its destruction is delayed until the end of the trace action execution. In this case, its attributes are still visible (except the departure queue which can be however found by GETTRACE:QSECONDR).

- GETTRACE:CSECONDR returns the customer after which the transited customer takes place.

- GETTRACE:QPROVOKE returns the queue (if it exists) containing the customer which has triggered off the operation.

- GETTRACE:QSUBJECT returns the queue receiving the transited customer.

- GETTRACE:QSECONDR returns the departure queue of the transited customer.

- GETTRACE:CCLASS returns the previous class of the transited customer (i.e., before transit). The new class can be accessed as an attribute of this customer.

- GETTRACE:CPRIOR returns the previous priority of the transited customer (i.e., before transit). The new class can be accessed as an attribute of this customer.

- GETTRACE:EVSTATUS returns 0 if the transit has succeed and 1 if the transit has aborted because of the reject due to a finite capacity.

SEE ALSO

MOVE - BEFCUST

"BEFCUST"

NAME

"BEFCUST" - Associated to the BEFCUST procedure.

DESCRIPTION

- GETTRACE:CPROVOKE, GETTRACE:EXCEPTPROVOKE or GETTRACE:TIMERPROVOKE return respectively (according to the result returned by GETTRACE:WHICHPRO) the customer, the exception or the timer which triggers off the operation.

- GETTRACE:CSUBJECT returns the customer which has been transited. If it has been transited to OUT, its destruction is delayed until the end of the trace action execution. In this case, its attributes are still visible (except the departure queue which can be however found by GETTRACE:QSECONDR).

- GETTRACE:CSECONDR returns the customer before which the transited customer takes place.

- GETTRACE:QPROVOKE returns the queue (if it exists) containing the customer which has triggered off the operation.

- GETTRACE:QSUBJECT returns the queue receiving the transited customer.

- GETTRACE:QSECONDR returns the departure queue of the transited customer.

- GETTRACE:CCLASS returns the previous class of the transited customer (i.e., before transit). The new class can be accessed as an attribute of this customer.

- GETTRACE:CPRIOR returns the previous priority of the transited customer (i.e., before transit). The new class can be accessed as an attribute of this customer.

- GETTRACE:EVSTATUS returns 0 if the transit has succeed and 1 if the transit has aborted because of the reject due to a finite capacity.

SEE ALSO

MOVE - AFTCUST

NAME

"BLOCK" - Associated to the BLOCK procedure.

DESCRIPTION

- GETTRACE:CPROVOKE, GETTRACE:EXCEPTPROVOKE or GETTRACE:TIMERPROVOKE return respectively (according to the result returned by GETTRACE:WHICHPRO) the customer, the exception or the timer which triggers off the operation.

- GETTRACE:QPROVOKE returns the queue (if it exists) containing the customer which has triggered off the operation.

- GETTRACE:LQUNB returns the number of blocked queues.

- GETTRACE:QLISTGET (*range*) returns among blocked queues, the one which has the range defined by argument.

SEE ALSO

"UNBLOCK" - BLOCK - UNBLOCK

"ENDSERV"

NAME

"ENDSERV" - End of service and transit of the customer to the next queue (or OUT).

DESCRIPTION

- GETTRACE:CPROVOKE (or GETTRACE:CSUBJECT) returns the customer which has finished a service.

- GETTRACE:QPROVOKE returns the queue in which a service has been completed.

- GETTRACE:QSUBJECT returns the queue in which the customer is transited after the end of its service.

- GETTRACE:CCLASS returns the previous class of the transited customer (i.e., before transit). The new class can be accessed as an attribute of the customer.

- GETTRACE:CPRIOR returns the previous priority of the transited customer (i.e., before transit). The new class can be accessed as an attribute of the customer.

- GETTRACE:EVSTATUS returns 0 if the transit has succeed without any other action, 1 if the transit has aborted because of the reject on a finite capacity, 2 if the customer should participate a MATCH or FUSION operation after the transit.

NOTE

The possible destruction of a customer after a transit to OUT (or after the end of a MATCH or FUSION operation) is delayed until the completion of the trace action.

NAME

"ENDSTART" - End of TSTART period.

DESCRIPTION

In this case, there is no returned value except the nature of the event during calls to the GETTRACE:keyword functions.

"FISSION"

NAME

"FISSION" - FISSION of a customer at queue departure.

DESCRIPTION

Access to data is the same for events of "SPLIT" or "FISSION" type.

- GETTRACE:QPROVOKE returns the queue from which the FISSION operation has occured.

- GETTRACE:CPROVOKE returns the customer which is splitted.

- GETTRACE:CCLASS returns the customers class which has been used to select the FISSION in the case where specifications per class has been required by the user. In all the other cases, where the default FISSION has been chosen (i.e., all customers classes mixed), this function returns NIL.

- GETTRACE:NUMBER returns the FISSION range for the customer according to its class in the case where several FISSION are possible with a probabilistic choice. This range corresponds to the specification order defined in the /STATION/ command. If only one FISSION is possible, the returned value is 1.

- GETTRACE:LCUSTNB returns the number of customers resulting of the FISSION.

- GETTRACE:CLISTGET (*range*) returns among customers resulting of the FISSION, the one which has the range defined by argument. The queue and the class of each of this customers can be accessed by their attributes.

- GETTRACE:EVSTATUS returns 0 if all the customers resulting of the FISSION has succeeded their transit in destination queues. Else it returns the number of customers rejected on a finite capacity. This number is always greater than zero.

NOTE

The customer which is splitted by the FISSION operation stays visible during the trace action. It is destroyed after the completion of this trace action.

SEE ALSO

"FUSION" - FISSION

NAME

"FREE" - Associated to the FREE procedure.

DESCRIPTION

- GETTRACE:CPROVOKE, GETTRACE:EXCEPTPROVOKE or GETTRACE:TIMERPROVOKE return respectively (according to the result returned by GETTRACE:WHICHPRO) the customer, the exception or the timer which triggers off the operation.

- GETTRACE:QPROVOKE returns the queue (if it exists) containing the current customer (returned by GETTRACE:CPROVOKE).

- GETTRACE:CSUBJECT and GETTRACE:QSUBJECT return respectively the freed customer and the queue which contains it.

- GETTRACE:EVSTATUS is not significant in this case and returns 0.

NOTE

Calls to FREE (*customer, semaphore* | *resource*) or to FREE (*customer, flag*) are treated respectively by the "FREEP" and "FREEFLAG" events.

SEE ALSO

"FREEP" - "FREEFLAG" - FREE

NAME

"FREEFLAG" - Freeing of a customer blocked on a flag.

DESCRIPTION

- GETTRACE:CPROVOKE, GETTRACE:EXCEPTPROVOKE or GETTRACE:TIMERPROVOKE return respectively (according to the result returned by GETTRACE:WHICHPRO) the customer, the exception or the timer which triggers off the operation.

- GETTRACE:QPROVOKE returns the queue (if it exists) containing the current customer (returned by GETTRACE:CPROVOKE).

- GETTRACE:CSUBJECT and GETTRACE:QSUBJECT return respectively the freed customer and the queue which contains it.

- GETTRACE:FLAG returns the FLAG object from which the customer has been freed.

- GETTRACE:EVSTATUS is not significant in this case and returns 0.

SEE ALSO

"FREE" - FREE

NAME

"FREEP" - Freeing of a customer blocked on a semaphore or a resource.

DESCRIPTION

- GETTRACE:CPROVOKE, GETTRACE:EXCEPTPROVOKE or GETTRACE:TIMERPROVOKE return respectively (according to the result returned by GETTRACE:WHICHPRO) the customer, the exception or the timer which triggers off the operation.

- GETTRACE:QPROVOKE returns the queue (if it exists) containing the current customer (returned by GETTRACE:CPROVOKE).

- GETTRACE:CSUBJECT returns the freed customer.

- GETTRACE:QSECONDR returns the queue containing the freed customer.

- GETTRACE:CSUBJECT returns the semaphore or resource from which the customer has been freed.

- GETTRACE:EVSTATUS is not significant in this case and returns 0.

NOTE

For this event the relevant queue is the semaphore or the resource itself. Then, the queue containing the customer which is freed is secondary and considered as a partner.

SEE ALSO

"FREE" - FREE

NAME

"FUSION" - FUSION of a set of customers at a queue entry.

DESCRIPTION

Access to data is the same for events of "MATCH" or "FUSION" type.

- GETTRACE:CPROVOKE returns the last customer to group (i.e., the one which triggers off the final reassembly). This customer also appears in the set of grouped customers.

- GETTRACE:QPROVOKE or GETTRACE:QSUBJECT returns the queue from which the reassembly is done.

- GETTRACE:CSUBJECT returns the customer issued of the FUSION operation.

- GETTRACE:CCLASS returns the class of the customer issued of the FUSION operation.

- GETTRACE:LCUSTNB returns the total number of grouped customers.

- GETTRACE:CLISTGET (*range*) returns among grouped customers the one which has the range defined by argument.

NOTE

The destruction of the customers which are grouped together is delayed until completion of the trace action. The attributes of these customers are still available (except the current queue which is then meaningless).

SEE ALSO

"FISSION" - "MATCH" - FUSION

NAME

"INIT" - Initial creation of customers in a queue.

DESCRIPTION

- GETTRACE:QPROVOKE returns the queue where a customer creation has been done.

- GETTRACE:CPROVOKE returns the new customer.

NOTE

This event occurs only when a simulation is started. It is treated after the SIMSTART exception.

SEE ALSO

INIT

NAME

"JOINALL" - Associated to the JOIN or JOINC procedure on all sons customers.

DESCRIPTION

- GETTRACE:CPROVOKE, GETTRACE:EXCEPTPROVOKE or GETTRACE:TIMERPROVOKE return respectively (according to the result returned by GETTRACE:WHICHPRO) the customer, the exception or the timer which triggers off the operation.

- GETTRACE:QPROVOKE returns the queue (if it exists) containing the customer which has triggered off the operation (returned by GETTRACE:CPROVOKE).

- GETTRACE:CSUBJECT returns the customer which should wait for the completion of its sons customers.

- GETTRACE:QSUBJECT returns the queue containing the customer which should wait for the completion of its sons customers.

- GETTRACE:EVSTATUS is not significant in this case and returns 0.

SEE ALSO

JOIN - JOINC

NAME

"JOINEND" - End of customer waiting on a JOIN or JOINC procedure.

DESCRIPTION

- GETTRACE:CPROVOKE returns the customer which has been completed (last son customer transited OUT).

- GETTRACE:QPROVOKE returns the queue from which this last son customer has been completed.

- GETTRACE:CSUBJECT returns the customer (father customer) which should wait for the completion of its sons customers.

- GETTRACE:QSUBJECT returns the queue containing this last customer (father customer).

NAME

"JOINLIST" - Associated to a JOIN or JOINC procedure on a list of customers.

DESCRIPTION

- GETTRACE:CPROVOKE, GETTRACE:EXCEPTPROVOKE or GETTRACE:TIMERPROVOKE return respectively (according to the result returned by GETTRACE:WHICHPRO) the customer, the exception or the timer which triggers off the operation.

- GETTRACE:QPROVOKE returns the queue (if it exists) containing the customer which has triggered off the operation (returned by GETTRACE:CPROVOKE).

- GETTRACE:CSUBJECT returns the customer which should wait for the completion of its sons customers.

- GETTRACE:QSUBJECT returns the queue containing the customer which should wait for the completion of its sons customers.

- GETTRACE:LCUSTNB returns the number of sons customers to wait for.

- GETTRACE:CLISTGET (*range*) returns the son customer to wait for, corresponding to the range defined by argument.

- GETTRACE:EVSTATUS is not significant in this case and returns 0.

SEE ALSO

"JOINALL" - "JOINNB" - JOIN - JOINC

NAME

"JOINNB" - Associated to the JOIN or JOINC procedure on number of customers.

DESCRIPTION

- GETTRACE:CPROVOKE, GETTRACE:EXCEPTPROVOKE or GETTRACE:TIMERPROVOKE return respectively (according to the result returned by GETTRACE:WHICHPRO) the customer, the exception or the timer which triggers off the operation.

- GETTRACE:QPROVOKE returns the queue (if it exists) containing the customer which has triggered off the operation (returned by GETTRACE:CPROVOKE).

- GETTRACE:CSUBJECT returns the customer which should wait for the completion of its sons customers.

- GETTRACE:QSUBJECT returns the queue containing the customer which should wait for the completion of its sons customers.

- GETTRACE:NUMBER returns the number of sons customers to wait for.

- GETTRACE:EVSTATUS is not significant in this case and returns 0.

SEE ALSO

"JOINALL" - "JOINLIST" - JOIN - JOINC

NAME

"MATCH" - MATCH of a set of customers at a queue entry.

DESCRIPTION

Access to data is the same for events of "MATCH" or "FUSION" type.

- GETTRACE:CPROVOKE returns the last customer to group (i.e., the one which trigger off the final reassembly). This customer also appears in the set of grouped customers.

- GETTRACE:QPROVOKE or GETTRACE:QSUBJECT returns the queue from which the reassembly is done.

- GETTRACE:CSUBJECT returns the customer issued of the MATCH operation.

- GETTRACE:CCLASS returns the class of the customer issued of the MATCH operation.

- GETTRACE:LCUSTNB returns the total number of grouped customers.

- GETTRACE:CLISTGET (*range*) returns among grouped customers the one which has the range defined by argument.

NOTE

The destruction of the customers which are grouped together is delayed until the execution of the trace action. The attributes of these customers are still available (except the current queue which is then meaningless).

SEE ALSO

"FUSION"

NAME

"MOVE" - Associated to the MOVE procedure.

DESCRIPTION

- GETTRACE:CPROVOKE, GETTRACE:EXCEPTPROVOKE or GETTRACE:TIMERPROVOKE return respectively (according to the result returned by GETTRACE:WHICHPRO) the customer, the exception or the timer which triggers off the operation.

- GETTRACE:QPROVOKE returns the queue (if it exists) containing the customer which has triggered off the operation (returned by GETTRACE:CPROVOKE).

- GETTRACE:CSUBJECT returns the customer which has been transited. If it has been transited to OUT, its destruction is delayed until the end of the trace action execution. In this case, its attributes are still visible (except the departure queue which can be however found by GETTRACE:QSECONDR).

- GETTRACE:QSUBJECT returns the destination queue of the customer which has been transited.

- GETTRACE:QSECONDR returns the departure queue of the customer which has been transited.

- GETTRACE:EVSTATUS returns 0 if the transit has succeed and 1 if the transit has aborted because of the reject due to a finite capacity.

NOTE

The change of queue for a customer at the end of a service is treated by the "ENDSERV" event.

SEE ALSO

MOVE

NAME

"NEWCUST" - Associated to the NEW(*CUSTOMER*) function.

DESCRIPTION

- GETTRACE:CPROVOKE, GETTRACE:EXCEPTPROVOKE or GETTRACE:TIMERPROVOKE return respectively (according to the result returned by GETTRACE:WHICHPRO) the customer, the exception or the timer which triggers off the operation.

- GETTRACE:QPROVOKE returns the queue (if it exists) containing the customer which has called the NEW(*CUSTOMER*) function.

- GETTRACE:CSUBJECT returns the created customer.

NAME

"P" - Associated to the P procedure on a resource or a semaphore.

DESCRIPTION

- GETTRACE:CPROVOKE, GETTRACE:EXCEPTPROVOKE or GETTRACE:TIMERPROVOKE return respectively (according to the result returned by GETTRACE:WHICHPRO) the customer, the exception or the timer which triggers off the operation.

- GETTRACE:CSUBJECT returns the customer requiring a unit on the resource or the semaphore.

- GETTRACE:QPROVOKE returns the queue (if it exists) containing the current customer (returned by GETTRACE:CPROVOKE).

- GETTRACE:QSECONDR returns the queue containing the customer requiring a unit on the resource or the semaphore.

- GETTRACE:QSUBJECT returns the resource or the semaphore itself.

- GETTRACE:CCLASS returns the customer class defined by argument during the call to the P procedure.

- GETTRACE:CPRIOR returns the priority level of the request.

- GETTRACE:EVSTATUS returns 0 if the P operation is not blocking, 1 if it is blocking, 2 if the request has been rejected because of a finite capacity on the resource or semaphore.

SEE ALSO

P - V - RESSOURCE - SEMAPHORE

NAME

"PMULT" - Event associated to the PMULT procedure.

DESCRIPTION

- GETTRACE:CPROVOKE, GETTRACE:EXCEPTPROVOKE or GETTRACE:TIMERPROVOKE return respectively (according to the result returned by GETTRACE:WHICHPRO) the customer, the exception or the timer which triggers off the operation.

- GETTRACE:CSUBJECT returns the customer for which units of semaphores and/or resources are requested.

- GETTRACE:QPROVOKE returns the queue (or NIL if it does not exist) containing the current customer (returned by GETTRACE:CPROVOKE).

- GETTRACE:QSUBJECT returns the queue which contains the customer for which the requests are performed.

- GETTRACE:LQUNB returns the number of different semaphores and/or resources requested by the operation.

- GETTRACE:QLISTGET (n) returns the n^{th} semaphore and/or resource.

- GETTRACE:LNUMNB returns the number of the different sets of requests on the different semaphores and/or resources.

- GETTRACE:NUMLISTGET (n) returns the number of requests on the n^{th} semaphore and/or resource.

- GETTRACE:LCLASSNB returns the number of different request's classes on all semaphores and/or resources requested by the operation.

- GETTRACE:CLLISTGET (n) returns the class of requests on the n^{th} semaphore and/or resource.

- GETTRACE:LPRIONB returns the number of different request's priorities on all semaphores and/or resources requested by the operation.

- GETTRACE:PRILISTGET (n) returns the priority of requests on the n^{th} semaphore and/or resource.

- GETTRACE:EVSTATUS returns 0 if the operation causes no wait, 1 if the customer subject is waiting on a semaphore or a resource as a result of the operation, 2 if some request has been rejected because of a limited capacity on one of the semaphores or resources.

SEE ALSO

VMULT

"PRIOR"

NAME

"PRIOR" - Associated to the PRIOR procedure.

DESCRIPTION

- GETTRACE:CPROVOKE, GETTRACE:EXCEPTPROVOKE or GETTRACE:TIMERPROVOKE return respectively (according to the result returned by GETTRACE:WHICHPRO) the customer, the exception or the timer which triggers off the operation.

- GETTRACE:QPROVOKE returns the queue (if it exists) containing the current customer (returned by GETTRACE:CPROVOKE).

- GETTRACE:CSUBJECT returns the customer of which the priority has been modified.

- GETTRACE:QSUBJECT returns the queue containing the customer of which the priority has been modified.

- GETTRACE:CPRIOR returns the priority before the modification. The new priority can be accessed as an attribute of the modified customer.

- GETTRACE:EVSTATUS is not significant in this case and returns 0.

SEE ALSO

PRIOR

NAME

"RESET" - Associated to the RESET procedure on a FLAG object.

DESCRIPTION

- GETTRACE:CPROVOKE, GETTRACE:EXCEPTPROVOKE or GETTRACE:TIMERPROVOKE return respectively (according to the result returned by GETTRACE:WHICHPRO) the customer, the exception or the timer which triggers off the operation.

- GETTRACE:QPROVOKE returns the queue (if it exists) containing the current customer (returned by GETTRACE:CPROVOKE).

- GETTRACE:FLAG returns the modified FLAG object.

- GETTRACE:EVSTATUS is not significant in this case and returns 0.

SEE ALSO

WAIT - WAITAND - WAITOR - FLAG - SET

"SERVTIME"

NAME

"SERVTIME" - Request of a service time.

DESCRIPTION

- GETTRACE:CPROVOKE or GETTRACE:CSUBJECT returns the current customer which has required a service time.

- GETTRACE:QPROVOKE or GETTRACE:QSUBJECT returns the queue containing the current customer which has required a service time.

- GETTRACE:DELAY returns the real value corresponding to the service time required.

- GETTRACE:DISTRI returns the distribution law used to the random drawing (if it exists) of the service time.

- GETTRACE:PARDISTR (*parameter_range*) allows to obtain one by one as real values, the parameters of the distribution service time used. These parameters are defined by arguments of the time delay request procedure used. Their meaning depends on each procedure and is given with its description.

- GETTRACE:EVSTATUS is not significant in this case and returns 0.

SEE ALSO

GETTRACE:DISTRI - GETTRACE:PARDISTR - GETTRACE:DELAY

NAME

"SET" - Associated to the SET procedure on a FLAG object.

DESCRIPTION

- GETTRACE:CPROVOKE, GETTRACE:EXCEPTPROVOKE or GETTRACE:TIMERPROVOKE return respectively (according to the result returned by GETTRACE:WHICHPRO) the customer, the exception or the timer which triggers off the operation.

- GETTRACE:QPROVOKE returns the queue (if it exists) containing the current customer (returned by GETTRACE:CPROVOKE).

- GETTRACE:FLAG returns the modified FLAG object.

- GETTRACE:EVSTATUS is not significant in this case and returns 0.

SEE ALSO

FLAG - WAIT - WAITAND - WAITOR - RESET

"SOURCE"

NAME

"SOURCE" - Customer creation in a source station.

DESCRIPTION

- GETTRACE:QPROVOKE returns the source station which has created the customer.

- GETTRACE:CSUBJECT returns the created customer.

SEE ALSO

SOURCE

NAME

"SPLIT" - SPLIT of a customer at queue departure.

DESCRIPTION

Access to data is the same for events of "SPLIT" or "FISSION" type.

- GETTRACE:QPROVOKE returns the queue from which the SPLIT operation has occured.

- GETTRACE:CPROVOKE returns the customer which is splitted.

- GETTRACE:CCLASS returns the customers class which has been used to select the SPLIT in the case where specifications per class has been required by the user. In all the other cases, where the default SPLIT has been chosen (i.e., all customers classes mixed), this function returns NIL.

- GETTRACE:NUMBER returns the SPLIT range for the customer according to its class in the case where several SPLIT are possible with a probabilistic choice. This range corresponds to the specification order defined in the STATION command. If only one SPLIT is possible, the returned value is 1.

- GETTRACE:LCUSTNB returns the number of customers resulting of the SPLIT.

- GETTRACE:CLISTGET (*range*) returns among customers resulting of the SPLIT, the one which has the range defined by argument. The queue and the class of each of this customers can be accessed by their attributes.

- GETTRACE:EVSTATUS returns 0 if all the customers resulting of the SPLIT has succeed their transite to destination queues. Otherwise, it returns the number of customers rejected on a finite capacity. This number is always greater than zero.

NOTE

The customer which is splitted by the SPLIT operation stays visible during the trace action. It is destroyed after the completion of this trace action.

SEE ALSO

"FISSION" - "MATCH" - SPLIT

NAME

"TMRCANCL" - Cancel of a timer by the user.

DESCRIPTION

- GETTRACE:CPROVOKE, GETTRACE:EXCEPTPROVOKE or GETTRACE:TIMERPROVOKE return respectively (according to the result returned by GETTRACE:WHICHPRO) the customer, the exception or the timer which triggers off the operation.

A current timer can be cancelled by itself and then considered as having triggered off the operation.

- GETTRACE:TIMERSUBJECT returns the timer cancelled by the operation (call to SETTIMER:CANCEL).

NOTE

Only a call to the SETTIMER:CANCEL procedure generates this kind of event.

SEE ALSO

SETTIMER:CANCEL

NAME

"TMRSETTM" - Launching of a timer by the user.

DESCRIPTION

- GETTRACE:CPROVOKE, GETTRACE:EXCEPTPROVOKE or GETTRACE:TIMERPROVOKE return respectively (according to the result returned by GETTRACE:WHICHPRO) the customer, the exception or the timer which triggers off the operation.

A current timer can be launched by itself and then considered as having triggered off the operation.

- GETTRACE:TIMERSUBJECT returns the timer launched again for a future date by the operation (call to SETTIMER:ABSOLUTE, SETTIMER:RELATIVE, SETTIMER:CYCLIC, SETTIMER:TRACKTIME).

- GETTRACE:DELAY returns the absolute activation date for the timer in the future (this is an absolute date in opposition to service requests which are relative).

- GETTRACE:NUMBER returns the timer activation code. Its value can also be accessed using the STATE timer attribute.

NOTE

Only an activation by a user generates this kind of event. The automatic activation or reactivation by QNAP2 does not give rise to an event which can be traced.

SEE ALSO

STATE

"TMRWAKUP"

NAME

"TMRWAKUP" - Starting of timer execution.

DESCRIPTION

GETTRACE:TIMERPROVOKE returns the reference to the timer of which the code execution (implicite or user defined) begins immediately after the trace action. No other information is pertinent in this case.

NOTE

The trace action is performed just before the timer code execution itself.

SEE ALSO

"TMRSETTM" - "TMRCANCL"

NAME

"TRANSIT" - Associated to the TRANSIT procedure.

DESCRIPTION

- GETTRACE:CPROVOKE, GETTRACE:EXCEPTPROVOKE or GETTRACE:TIMERPROVOKE return respectively (according to the result returned by GETTRACE:WHICHPRO) the customer, the exception or the timer which triggers off the operation.

- GETTRACE:CSUBJECT returns the customer which has been transited. If it has been transited to OUT, its destruction is delayed until the end of the trace action execution. In this case, its attributes are still visible (except the departure queue which can be however found by GETTRACE:QSECONDR).

- GETTRACE:QPROVOKE returns the queue (if it exists) containing the current customer (returned by GETTRACE:CPROVOKE).

- GETTRACE:QSUBJECT returns the destination queue of the customer which has been transited.

- GETTRACE:QSECONDR returns the departure queue of the customer which has been transited.

- GETTRACE:CCLASS returns the previous class of the transited customer (i.e., before transit). The new class can be accessed as an attribute of this customer.

- GETTRACE:CPRIOR returns the previous priority of the transited customer (i.e., before transit). The new class can be accessed as an attribute of this customer.

- GETTRACE:EVSTATUS returns 0 if the transit has succeed and 1 if the transit has aborted because of the reject due to a finite capacity.

NOTE

The change of queue for a customer at the end of a service is treated by the "ENDSERV" event.

SEE ALSO

STATUS - STATION - TRANSIT

NAME

"UNBLOCK" - Associated to the UNBLOCK procedure.

DESCRIPTION

- GETTRACE:CPROVOKE, GETTRACE:EXCEPTPROVOKE or GETTRACE:TIMERPROVOKE return respectively (according to the result returned by GETTRACE:WHICHPRO) the customer, the exception or the timer which triggers off the operation.

- GETTRACE:QPROVOKE returns the queue (if it exists) containing the customer which has triggered off the operation (returned by GETTRACE:CPROVOKE).

- GETTRACE:LQUNB returns the number of queues which are released.

- GETTRACE:QLISTGET (*range*) returns among released queue, the one of which range is defined by argument.

SEE ALSO

"BLOCK" - BLOCK - UNBLOCK

NAME

"V" - Associated to the V procedure on resource or semaphore.

DESCRIPTION

- GETTRACE:CPROVOKE, GETTRACE:EXCEPTPROVOKE or GETTRACE:TIMERPROVOKE return respectively (according to the result returned by GETTRACE:WHICHPRO) the customer, the exception or the timer which triggers off the operation.

- GETTRACE:CSUBJECT returns the customer which has performed the P operation. The result is the same as GETTRACE:CPROVOKE in the case of a semaphore.

- GETTRACE:QPROVOKE returns (if it exists) the queue containing the current customer (returned by GETTRACE:CPROVOKE).

- GETTRACE:QSECONDR returns the queue containing the customer which frees a unit of the resource or the semaphore.

- GETTRACE:QSUBJECT returns the resource or the semaphore itself.

- GETTRACE:EVSTATUS is not significant in this case and returns 0.

SEE ALSO

P - V - RESSOURCE - SEMAPHORE

NAME

"VMULT" - Event associated to the VMULT procedure.

DESCRIPTION

- GETTRACE:CPROVOKE, GETTRACE:EXCEPTPROVOKE or GETTRACE:TIMERPROVOKE return respectively (according to the result returned by GETTRACE:WHICHPRO) the customer, the exception or the timer which triggers off the operation.

- GETTRACE:CSUBJECT returns the customer which releases units of semaphores and/or resources during the operation.

- GETTRACE:QPROVOKE returns the queue (or NIL if it does not exist) containing the current customer (returned by GETTRACE:CPROVOKE).

- GETTRACE:QSUBJECT returns the queue which contains the customer which is releasing units of semaphores and/or resources.

- GETTRACE:LQUNB returns the number of different semaphores and/or resources on which the operation is performed.

- GETTRACE:QLISTGET (*n*) returns the n^{th} semaphore and/or resource.

- GETTRACE:LNUMNB returns the number of the different sets of releasings on the different semaphores and/or resources.

- GETTRACE:NUMLISTGET (*n*) returns the number of units released on the n^{th} semaphore and/or resource.

SEE ALSO

PMULT

NAME

"WAIT" - Associated to the WAIT procedure on a FLAG object.

DESCRIPTION

- GETTRACE:CPROVOKE, GETTRACE:EXCEPTPROVOKE or GETTRACE:TIMERPROVOKE return respectively (according to the result returned by GETTRACE:WHICHPRO) the customer, the exception or the timer which triggers off the operation.

- GETTRACE:CSUBJECT returns the customer blocked on the current FLAG object.

- GETTRACE:QPROVOKE returns the queue (if it exists) containing the current customer (returned by GETTRACE:CPROVOKE).

- GETTRACE:QSUBJECT returns the queue containing the blocked customer.

- GETTRACE:FLAG returns the FLAG object involved.

- GETTRACE:EVSTATUS returns 0 if flag state is "SET", and 1 if it is "UNSET".

SEE ALSO

"WAITAND" - "WAITOR" - WAIT - FLAG - SET - RESET

"WAITAND"

NAME

"WAITAND" - Associated to the WAITAND procedure on FLAG objects.

DESCRIPTION

- GETTRACE:CPROVOKE, GETTRACE:EXCEPTPROVOKE or GETTRACE:TIMERPROVOKE return respectively (according to the result returned by GETTRACE:WHICHPRO) the customer, the exception or the timer which triggers off the operation.

- GETTRACE:QPROVOKE returns the queue (if it exists) containing the current customer (returned by GETTRACE:CPROVOKE).

- GETTRACE:CSUBJECT returns the customer blocked on a set of FLAG objects.

- GETTRACE:QSUBJECT returns the queue containing the blocked customer.

- GETTRACE:LFLAGNB returns the number of FLAG objects involved.

- GETTRACE:FLISTGET (*range*) returns among FLAG objects involved, the one which has the range defined by argument.

- GETTRACE:EVSTATUS returns 0 if state of all flags is "SET", and 1 if at least one flag state is "UNSET".

SEE ALSO

"WAIT" - WAIT - WAITAND - FLAG

NAME

"WAITOR" - Associated to the WAITOR procedure on flag objects.

DESCRIPTION

- GETTRACE:CPROVOKE, GETTRACE:EXCEPTPROVOKE or GETTRACE:TIMERPROVOKE return respectively (according to the result returned by GETTRACE:WHICHPRO) the customer, the exception or the timer which triggers off the operation.

- GETTRACE:QPROVOKE returns the queue (if it exists) containing the current customer (returned by GETTRACE:CPROVOKE).

- GETTRACE:CSUBJECT returns the customer blocked on a set of FLAG objects.

- GETTRACE:QSUBJECT returns the queue containing the blocked customer.

- GETTRACE:LFLAGNB returns the number of FLAG objects involved.

- GETTRACE:FLISTGET (*range*) returns among FLAG objects involved, the one which has the range defined by argument.

- GETTRACE:EVSTATUS returns 0 if at least one flag state is "SET", and 1 if state of all flags is "UNSET".

SEE ALSO

"WAIT" - WAIT - WAITOR - FLAG

Chapter 5

FORTRAN Interface

QLOADB	Boolean loading.
QLOADI	Integer loading.
QLOADR	Real loading.
QLOADS	String loading.
QSTORB	Boolean storage.
QSTORI	Integer storage.
QSTORR	Real storage.
QSTORS	String storage.
UTILIT	QNAP2 to FORTRAN interface subroutine.

NAME

QLOADB - Boolean loading.

SYNTAX

QLOADB (IRTAB 1 | IRTAB 2, *integer*, *integer*, *array - booleans*, *integer*, *integer*, *integer*)

DESCRIPTION

This subroutine permits one to recover a QNAP2 boolean array in a FORTRAN boolean array (LOGICAL).

The first argument corresponds to the second or the third argument of UTILIT (first or second QNAP2 boolean array). This position is defined by the order the boolean array in the UTILITY call.

The second argument is the index of the first element from the QNAP2 boolean array to recover.

The third argument is the index of the last element from the QNAP2 boolean array to recover.

The forth argument is the name of the FORTRAN boolean array where the recovered values will be stored.

The fifth argument is the index of the first element recovered in the FORTRAN boolean array

The sixth argument is the index of the last element recovered in the FORTRAN boolean array

The seventh argument is the error code returned by QLOADB. Possible error codes are:

- 0 if no error occured,
- 2 if the operation failed.

NOTE

This subroutine must be used in the UTILIT subroutine, or in another subroutine called by UTILIT.

SEE ALSO

QSTORB - UTILITY - UTILIT

EXAMPLE

```
 QNAP2 CODE
 ----------

/ DECLARE / BOOLEAN itab (3) = (TRUE, TRUE, FALSE);
/ EXEC / UTILITY (1, itab);


FORTRAN CODE
------------
 SUBROUTINE UTILIT (ICODE, IRTAB 1, IRTAB 2)
 INTEGER ICODE, IRTAB 1, IRTAB 2
 INTEGER N
 PARAMETER (N = 3)
 LOGICAL TABLOG (N)
 CALL  QLOADB (IRTAB 1, 1, N, TABLOG, 1, N, IERR)
 ...
```

At the end of QLOADB, TABLOG contains .TRUE., .TRUE. and .FALSE.

NAME

QLOADI - Integer loading.

SYNTAX

QLOADI (IRTAB 1 | IRTAB 2, *integer*, *integer*, *array* - *integers*, *integer*, *integer*, *integer*)

DESCRIPTION

This subroutine permits one to recover a QNAP2 integer array in a FORTRAN integer array.

The first argument corresponds to the second or the third argument of UTILIT (first or second QNAP2 integer array). This position is defined by the order of the integer array in the UTILITY call.

The second argument is the index of the first element from the QNAP2 integer array to recover.

The third argument is the index of the last element from the QNAP2 integer array to recover.

The forth argument is the name of the FORTRAN integer array where the recovered values will be stored.

The fifth argument is the index of the first element recovered in the FORTRAN integer array.

The sixth argument is the index of the last element recovered in the FORTRAN integer array.

The seventh argument is the error code returned by QLOADI. The possible error codes are:

- 0 if no error occured,
- 2 if the operation failed.

NOTE

This subroutine must be used in the UTILIT subroutine, or in another subroutine called by UTILIT.

SEE ALSO

QSTORI - UTILITY - UTILIT

EXAMPLE

```
 QNAP2 CODE
 ----------

/DECLARE/ INTEGER itab (3) = (2, 5, 4) ;
/ EXEC / UTILITY (1, itab);


FORTRAN CODE
------------
 SUBROUTINE UTILIT (ICODE, IRTAB 1, IRTAB 2)
 INTEGER ICODE, IRTAB 1, IRTAB 2
 INTEGER N
 PARAMETER (N = 3)
 INTEGER TABINT (N)
 CALL  QLOADI (IRTAB 1, 1, N, TABINT, 1, N, IERR)
 ...
```

At the end of `QLOADI`, `TABINT` contains 2, 5 and 4

NAME

QLOADR - Real loading.

SYNTAX

QLOADR (IRTAB 1 | IRTAB 2, *integer*, *integer*, *array* - *integers*, *integer*, *integer*, *integer*)

DESCRIPTION

This subroutine permits one to recover a QNAP2 integer array in a FORTRAN real array.

The first argument corresponds to the second or the third argument of UTILIT (first or second QNAP2 integer array). This position is defined by the order of the real array in the UTILITY call.

The second argument is the index of the first element from the QNAP2 real array to recover.

The third argument is the index of the last element from the QNAP2 real array to recover.

The forth argument is the name of the FORTRAN real array where the recovered values will be stored.

The fifth argument is the index of the first element recovered in the FORTRAN real array.

The sixth argument is the index of the last element recovered in the FORTRAN real array.

The seventh argument is the error code returned by QLOADR. The possible error codes are:

- 0 if no error occured,
- 2 if the operation failed.

NOTE

This subroutine must be used in the UTILIT subroutine, or in another subroutine called by UTILIT.

SEE ALSO

QSTORR - UTILITY - UTILIT

EXAMPLE

```
 QNAP2 CODE
 ----------

/DECLARE/ REAL itab (3) = (2.2, 5.4, 4.5);
/EXEC/ UTILITY (1, itab);


FORTRAN CODE
------------
 SUBROUTINE UTILIT (ICODE, IRTAB 1, IRTAB 2)
 INTEGER ICODE, IRTAB 1, IRTAB 2
 INTEGER N
 PARAMETER (N = 3)
 REAL TABREA (N)
 CALL  QLOADR (IRTAB 1, 1, N, TABREA, 1, N, IERR)
 ...
```

At the end of `QLOADR`, `TABREA` contains 2.2, 5.4 and 4.5.

NAME

QLOADS - String loading.

SYNTAX

QLOADS (IRTAB 1 | IRTAB 2,*integer*, *integer*, *array - string*, *array - integer*, *integer*, *integer*,*intger*)

DESCRIPTION

This subroutine permits one to recover a QNAP2 string array in a FORTRAN string array (CHARACTER).

The first argument corresponds to the second or the third argument of UTILIT (first or second QNAP2 integer array). This position is defined by the order of the string array in the UTILITY call.

The second argument is the index of the first element from the QNAP2 string array to recover.

The third argument is the index of the last element from the QNAP2 string array to recover.

The forth argument is the name of the FORTRAN string array where the recovered values will be stored.

The fifth argument is the index of the first element recovered in the FORTRAN string array.

The sixth argument is the index of the last element recovered in the FORTRAN string array.

The seventh argument is the error code returned by QLOADR. The possible error codes are:

- 0 if no error occured,
- 1 if a string is truncated,
- 2 if the operation failed.

NOTE

This subroutine must be used in the UTILIT subroutine, or in another subroutine called by UTILIT.

SEE ALSO

QSTORS - UTILITY - UTILIT

EXAMPLE

```
 QNAP2 CODE
 ----------

/DECLARE/ STRING itab (3) = ("one ", "two", "three");
/EXEC/ UTILITY (1, itab);


FORTRAN CODE
------------
 SUBROUTINE UTILIT (ICODE, IRTAB 1, IRTAB 2)
 INTEGER ICODE, IRTAB 1, IRTAB 2
 INTEGER N
 PARAMETER (N = 3)
 CHARACTER *236 TABCHA (N)
 INTEGER TABLON (N)
 CALL  QLOADS (IRTAB 1, 1, N, TABCHA, TABLON, 1, N, IERR)
...
```

At the end of QLOADB, TABLOG contains 'one', 'two' and 'three' and TABLON contains 3, 4 et 5.

NAME

QSTORB - Boolean storage.

SYNTAX

QSTORB (*array - boolean*, *integer*, *integer*, IRTAB 1 | IRTAB 2,
integer, *integer*, *integer*

DESCRIPTION

This subroutine permits to store a FORTRAN boolean array in a QNAP2 boolean array.

The first parameter is the name of the FORTRAN boolean array in the QNAP2 boolean array to store.

The second argument is the index of the first element from the FORTRAN boolean array to store.

The third argument is the index of the last element to store from the FORTRAN boolean array.

The forth argument is the name of the variable definied in second or third position in the UTILIT list of arguments (first or second QNAP2 boolean array).

The fifth argument is the index of the first element stored in the QNAP2 boolean array.

The sixth argument is the index of the last element stored in the QNAP2 boolean array.

The seventh argument is the error code returned by QSTORB. The possible error codes are:

- 0 if no error occured
- 2 if the operation failed

NOTE

This subroutine must be used in the UTILIT subroutine, or in another subroutine called by UTILIT.

SEE ALSO

QLOADB - UTILITY - UTILIT

EXAMPLE

```
FORTRAN CODE
------------

 SUBROUTINE UTILIT (ICODE, IRTAB 1, IRTAB 2)
 INTEGER ICODE, IRTAB 1, IRTAB 2
 INTEGER N
 PARAMETER (N = 3)
 LOGICAL TABLOG (N)
 TABLOG (1) = .TRUE.
 TABLOG (2) = .TRUE.
 TABLOG (3) = .FALSE.
 CALL  QSTORB (TABLOG, 1, N, IRTAB 1, 1, N, IERR)
 ...
```

```
QNAP2 CODE
----------
/DECLARE/ BOOLEAN itab (3);
/EXEC/ UTILITY (1, itab);
...
```

At the end of `UTILITY`, `itab` contains `.TRUE.`, `.TRUE.` and `.FALSE.`.

NAME

QSTORI - Integer storage.

SYNTAX

QSTORI (*array* - *integer*, *integer* ,*integer*, IRTAB 1 | IRTAB 2, *integer*, *integer*, *integer*)

DESCRIPTION

This subroutine permits to store a FORTRAN integer array in a QNAP2 integer array.

The first parameter is the name of the FORTRAN integer array in the QNAP2 integer array to store.

The second argument is the index of the first element from the FORTRAN integer array to store.

The third argument is the index of the last element to store from the FORTRAN integer array.

The forth argument is the name of the variable definied in second or third position in the UTILIT list of arguments (first or second QNAP2 integer array).

The fifth argument is the index of the first element stored in the QNAP2 integer array.

The sixth argument is the index of the last element stored in the QNAP2 integer array.

The seventh argument is the error code returned by QSTORI. The possible error codes are:

- 0 if no error occured,
- 2 if the operation failed.

NOTE

This subroutine must be used in the UTILIT subroutine, or in another subroutine called by UTILIT.

SEE ALSO

QLOADI - UTILITY - UTILIT

EXAMPLE

```
 FORTRAN CODE
 ------------

  SUBROUTINE UTILIT (ICODE, IRTAB 1, IRTAB 2)
  INTEGER ICODE, IRTAB 1, IRTAB 2
  INTEGER N
  PARAMETER (N = 3)
  TABINT (1) = 2
  TABINT (2) = 5
  TABINT (3) = 4
  CALL  QSTORI (TABINT, 1, N, IRTAB 1, 1, N, IERR
  ...
```

```
QNAP2 CODE
----------
/DECLARE/ INTEGER itab (3);
/EXEC/ UTILITY (1, itab);
...
```

At the end of `UTILITY`, `itab` contains 2, 5 and 4.

NAME

QSTORR - Real storage.

SYNTAX

QSTORR (*array - real, integer, integer,* IRTAB 1 | IRTAB 2, *integer, integer, integer*)

DESCRIPTION

This subroutine permits to store a FORTRAN real array in a QNAP2 real array.

The first parameter is the name of the FORTRAN real array in the QNAP2 real array to store.

The second argument is the index of the first element from the FORTRAN real array to store.

The third argument is the index of the last element from the FORTRAN integer array to store.

The forth argument is the name of the variable definied in second or third position in the UTILIT liste of arguments, first or second QNAP2 real array.

The fifth argument is the index of the first element stored in the QNAP2 real array.

The sixth argument is the index of the last element stored in the QNAP2 real array.

The seventh argument is the error code returned by QSTORR. The possible error codes are:

- 0 if no error occured,
- 2 if the operation failed.

NOTE

This subroutine must be used in the UTILIT sub-program, or in another subroutine called by UTILIT.

SEE ALSO

QLOADR - UTILITY - UTILIT

EXAMPLE

```
FORTRAN CODE
------------

 SUBROUTINE UTILIT (ICODE, IRTAB 1, IRTAB 2)
 INTEGER ICODE, IRTAB 1, IRTAB 2
 INTEGER N
 PARAMETER (N = 3)
 TABREA (1) = 2.2
 TABREA (2) = 5.4
 TABREA (3) = 4.5
 CALL  QSTORR (TABREA, 1, N, IRTAB 1, 1, N, IERR)
 ...  ...
```

```
QNAP2 CODE
----------
/DECLARE/ INTEGER itab (3);
/EXEC/ UTILITY (1, itab);
...
```

At the end of `UTILITY`, `itab` contains 2.4, 5.4 and 4.5.

NAME

QSTORS - String storage.

SYNTAX

QSTORS (*array - string*, *array -integer*, *integer*, *integer*, IRTAB 1 | IRTAB 2, *integer*, *integer*, *integer*)

DESCRIPTION

This subroutine permits to store a FORTRAN string array (CHARACTER) in a QNAP2 string array.

The first parameter is the name of the FORTRAN string array in the QNAP2 real array to store.

The second argument is the index of the first element from the FORTRAN string array to store.

The third argument is the index of the last element from the FORTRAN string array to store.

The forth argument is the name of the variable definied in second or third position in the UTILIT list of arguments, first or second QNAP2 string array.

The fifth argument is the index of the first element stored in the QNAP2 string array.

The sixth argument is the index of the last element stored in the QNAP2 string array.

The seventh argument is the error code returned by QSTORS, the error codes are:

- 0 if no error occured,
- 1 if a string is truncated,
- 2 if the operation failed.

NOTE

This subroutine must be used in the UTILIT subroutine, or in another subroutine called by UTILIT.

SEE ALSO

QLOADS - UTILITY - UTILIT

EXAMPLE

```
 FORTRAN CODE
 ------------

  SUBROUTINE UTILIT (ICODE, IRTAB 1, IRTAB 2)
  INTEGER ICODE, IRTAB 1, IRTAB 2
  INTEGER N
  PARAMETER (N = 3)
  CHARACTER TABCHA (N)
  INTEGER.TABLON (N)
  TABCHA (1) = 'un'
  TABLON (1) = 3
  TABCHA (2) = 'deux'
  TABLON (2) = 4
  TABCHA (3) = 'trois'
  TABLON (3) = 5
  CALL  QSTORS (TABCHA, TABLON, 1, N, IRTAB 1, 1, N, IERR)
  ...

QNAP2 CODE
----------
/DECLARE/ STRING itab (3);
/EXEC/ UTILITY (1, itab);
...
```

At the end of `UTILITY`, `itab` contains "un ", "deux" and "trois".

NAME

UTILIT - QNAP2 to FORTRAN interface subroutine.

SYNTAX

```
SUBROUTINE UTILIT(ICODE, IRTAB1, IRTAB2)
INTEGER ICODE, IRTAB1, IRTAB2
RETURN
END
```

DESCRIPTION

FORTRAN subroutines provide a way for QNAP2 to communicate with external modules.
The user FORTRAN subroutine must always have the name UTILIT and the minimum skeleton is given in the syntax paragraph.

NOTES

Transmisson of information from QNAP2 to FORTRAN or from FORTRAN to QNAP2 are managed by subroutines such as: QLOADI, QLOADR, QLOADB, QLOADS, QSTORI, QSTORR, QSTORB and QSTORS.

WARNING

- The user may declare COMMON variable only if the COMMON statements are named commons. The name of user commons must have the initial ''u'' to avoid collisions with the named commons of QNAP2.
- The variables ICODE, IRTAB1 and IRTAB2 must not be modified in the subroutine UTILIT.

SEE ALSO

QLOADI - QLOADR - QLOADB - QLOADS - QSTORI - QSTORR - QSTORB- QSTORS- UTILITY

EXAMPLE

```
      SUBROUTINE  UTILIT (ICODE, IRTAB 1, IRTAB 2)
      INTEGER ICODE, IRTAB 1, IRTAB 2
      INTEGER N
      PARAMETER (N = 3)
      INTEGER TABINT (N)
      CALL QLOADI (IRTAB 1, 1, N, TABINT, 1, N, IERR)
C call to a user subroutine
      CALL USER SP
      ...
```

Chapter 6

Macro-commands

$	Call to QNAP2 language macro-statement.
&	Definition of a comments area.
$MACRO	Definition of a QNAP2 language macro-statement.
$END	Marker for end of a QNAP2 language macro-statement definition.

NAME

$ - Call to QNAP2 language macro-statement.

SYNTAX

$*identifier* [(*parameter1, parameter2,...*)]

DESCRIPTION

This mechanism allows to invoke a sequence of statements recorded in a macro-statement.

identifier represents the name of the macro-statement to execute.

parameter1, *parameter2*, ... represent the effective arguments of the macro-statement. Each argument will replace in the sequence of statements beforehand recorded each of the occurences of the corresponding formal argument.

NOTES

A call to a macro-statement can be started only after its definition.

EVALUATION

- During the compilation.
- The content of the macro-statement is evaluated when the macro-statement is called.

WARNING

- A call to a macro-statement cannot contain argument of type element of array.
- The debugger cannot be used during the execution of the algorithmic code defined in a macro-statement.

SEE ALSO

`$MACRO - $END`

EXAMPLE

```
/DECLARE/ QUEUE cpu1,cpu2;

$MACRO DISK(name, time, destin)

   /DECLARE/ QUEUE name;

   /STATION/ NAME=name;
             SERVICE=EXP(time);
             TRANSIT=destin;

$END
...

$DISK( d1, 10.0, cpu1 )
$DISK( d2, 5.7, cpu2 )
```

QNAP2 statements generated by call to the macro-statements :

```
/DECLARE/ QUEUE d1;

/STATION/ NAME=d1
          SERVICE=EXP(10.0);
          TRANSIT=cpu1;

/DECLARE/ QUEUE d2;

/STATION/NAME=d2
       SERVICE=EXP(5.7);
       TRANSIT=cpu2;
```

NAME

& - Definition of a comments area.

SYNTAX

[*statements*] & *comment*

DESCRIPTION

This specific character allows to define a comments area.

It can be placed at the beginning of a line or after a statement of the QNAP2 language.

The comments area is defined between the & character and the end of the line.

EVALUATION

During the compilation.

EXAMPLE

```
& This is a comment
& This is another comment
/DECLARE/ OBJECT gamme ;      & description of a gamme
            INTEGER nbphas ;   & number of phases
            END ;
```

$MACRO

NAME

`$MACRO` - Definition of a QNAP2 language macro-statement.

SYNTAX

`$MACRO` *identifier* [(*parameter1, parameter2,...*)]
statement1;
statement2;
...
`$END`

DESCRIPTION

This mechanism allows the definition of a sequence of any statements which can be duplicated by a single call to the macro-statement defined.

identifier represents the name of the macro-statement.

parameter1, *parameter2*, ... represent the dummy arguments of the macro-statement. They have no type. Each occurence of a dummy argument will be replaced in the sequence of statements up to the `$END` by the effective argument defined during the call of the macro-statement.

statement1, *statement2*,... represent a sequence of any QNAP2 language statements. It can contains declarations, stations definitions, algorithmic language statements, ...

EVALUATION

- During the compilation.
- The content of the macro-statement will be evaluated when the macro-statement will be called.

WARNING

- A macro-statement should not contain calls to other macro-statements.
- The debugger cannot be used during the execution of the algorithmic code defined in a macro-statement.

SEE ALSO

`$ - $END`

EXAMPLE

```
/DECLARE/ QUEUE cpu1,cpu2;

$MACRO DISK(name, time, destin)

   /DECLARE/ QUEUE name;

   /STATION/ NAME=name;
             SERVICE=EXP(time);
             TRANSIT=destin;

$END
...

$DISK( d1, 10.0, cpu1 )
$DISK( d2, 5.7, cpu2 )
```

QNAP2 statements generated by call to the macro-statements :

```
/DECLARE/ QUEUE d1;

/STATION/ NAME=d1
          SERVICE=EXP(10.0);
          TRANSIT=cpu1;

/DECLARE/ QUEUE d2;

/STATION/NAME=d2
       SERVICE=EXP(5.7);
       TRANSIT=cpu2;
```

$END

NAME

`$END` - Marker for end of a QNAP2 language macro-statement definition.

SYNTAX

```
$END
```

DESCRIPTION

This keyword marks the end of a macro-statement definition.

EVALUATION

During the compilation.

SEE ALSO

`$` - `$MACRO`

EXAMPLE

```
/DECLARE/ QUEUE cpu1, cpu2;

$MACRO DISK(name, temps, destin)

   /DECLARE/ QUEUE name;

   /STATION/NAME=name;
            SERVICE=EXP(time);
            TRANSIT=destin;

$END
...

$DISK( d1, 10.0, cpu1 )
$DISK( d2, 5.7, cpu2 )
```

QNAP2 statements generated by call to the macro-statements:

```
/DECLARE/ QUEUE d1;

/STATION/NAME=d1
         SERVICE=EXP(10.0);
         TRANSIT=cpu1;

/DECLARE/ QUEUE d2;

/STATION/NAME=d2
         SERVICE=EXP(5.7);
         TRANSIT=cpu2;
```

Chapter 7

Algorithmic Language

7.1 Attributs

7.1.1 Attributes of CUSTOMER objects

ACTIVETIME	Returns the length of the active service performed
BLOCKED	Returns the state: blocked or not, of a customer.
BLOCKTIME	Returns the global blocked length for a customer since the beginning of the service execution.
CCLASS	References the customer class.
CHANGEDATE	Returns the date of the last customer state modification: active, waiting or blocked.
CLREJECT	References the class of customer responsible of the reject, of a limited capacity queue.
CONCSETN	Returns a customer concurrency-set number.
CPREEMPT	References the customer responsible of a customer reject.
CPRIOR	Returns a customer priority level.
CQUEUE	References the station containing a customer.
ENTERDATE	Returns the arrival date of a customer in the current station.
FATHER	References the father of a customer.
INSERVICE	Returns information about the affectation of a server to a customer.
NEXT	References the next customer in a queue.
PREVIOUS	References the previous customer in a queue.
QREJECT	Returns the queue which has rejected a customer because of a limited capacity.
RESPTIME	Returns the time spent by a customer in a station.
SON	References the last customer created by a customer.
STARTED	Returns the beginning date of a customer service.

NAME

ACTIVETIME - Returns the length of the active service performed by a customer.

SYNTAX

[*customer.*]ACTIVETIME

DESCRIPTION

Returns, a real, corresponding to the length of the active service performed by a customer.
The service of a customer is active when a server is allocated to the customer.
The returned value is the total of all the active durations. It takes into account the specified sevice rate (RATE option or SCHED = PS of /STATION/ command), blocking time are deduced.

customer is an expression representing a type CUSTOMER entity. This specification can be omitted, in a service specification or in a WITH structure.

EVALUATION

During the execution of the algorithmic language with a simulation resolution.

SEE ALSO

ENTERDATE - CHANGEDATE - BLOCKTIME

EXAMPLE

```
/STATION/ NAME = ...
          SERVICE = BEGIN
                      CST (10.0);
                      IF (CUSTOMER.ACTIVETIME <> 10.0) THEN
                         PRINT ("ERROR");
                    END;
```

BLOCKED

NAME

BLOCKED - Returns the state: blocked or not, of a customer.

SYNTAX

[*customer.*] BLOCKED

DESCRIPTION

The attribute returns a boolean which is TRUE if the customer is blocked (on a synchronization, for example), otherwise the value is FALSE.

customer is an expression representing a CUSTOMER type entity. This specification can be omitted, in a service specification or in a WITH structure.

EVALUATION

During the execution of the algorithmic language with a simulation resolution.

NOTES

- The blocking operations are P, WAIT, WAITOR, WAITAND, JOIN, JOINC et BLOCK.
- In a service procedure, the value associated to the current customer is FALSE.

WARNING

- This attribute can be used only in a simulation execution.
- This attribute cannot be modified by the user.

SEE ALSO

CUSTOMER - SERVICE - P - WAIT - WAITOR - WAITAND - JOIN - JOINC - BLOCK - WITH

EXAMPLE

```
/DECLARE/QUEUE q1, q2;

/STATION/NAME = q1;
         SERVICE = BEGIN
                     ...
                     PRINT (BLOCKED);   & the value is FALSE in every
                                        & cases because the customer
                                        & executes a service

                      ...
                      IF rc. BLOCKED THEN
                        ...
                    END;

/CONTROL/ TEST = BEGIN
                   ...
                   WITH rc DO
                   IF  BLOCKED THEN
                     ...
                 END;
  ...

/EXEC/ SIMUL;
```

BLOCKTIME

NAME

BLOCKTIME - Returns the global blocked length for a customer since the beginning of the service execution.

SYNTAX

[*customer.*]BLOCKTIME

DESCRIPTION

Returns a real, corresponding to the total blocked length for the specified customer since the beginning of the current service execution.

customer is an expression representing a CUSTOMER type entity. This specification can be omitted, in a service specification or in a WITH structure.

EVALUATION

During the execution of the algorithmic language with a simulation resolution.

WARNING

- This attribute can be used only in a simulation execution.
- This attribute cannot be modified by the user.

SEE ALSO

ENTERDATE - CHANGEDATE - ACTIVETIME - RESPTIME

EXAMPLE

```
/DECLARE/ QUEUE A,B;

/STATION/ NAME= A;
          TYPE = RESSOURCE;

/STATION/ NAME= B;
          TRANSIT= OUT;
          SERVICE= BEGIN
                     EXP (0.5);
                     P(A);
                     PRINT (CUSTOMER.BLOCKTIME);
                   END;

/CONTROL/ TMAX= 10.0;

/EXEC/ SIMUL;
```

NAME

CCLASS - References the customer class.

SYNTAX

[*customer* .] CCLASS

DESCRIPTION

This attribute references the customer class.

customer is an expression representing a CUSTOMER type entity. This specification can be omitted, in a service specification or in a WITH structure.

EVALUATION

During the execution of the algorithmic language with a simulation resolution.

NOTES

A customer class is specified statically by TRANSIT (/STATION/ parameter) and dynamically by TRANSIT, AFTCUST and BEFCUST (procedures).

WARNING

- This attribute can be used only in a simulation execution.
- This attribute cannot be modified by the user.

SEE ALSO

CUSTOMER - SERVICE - TRANSIT (parametre) - TRANSIT (procedure) - AFTCUST - BEFCUST - WITH

EXAMPLE

```
/DECLARE/ QUEUE q1, q2;
/STATION/ NAME = q1;
          SERVICE =
 BEGIN
   ...
   PRINT ( CCLASS) ;
   ...
   TRANSIT (q2, rc.CCLASS);
   ...
 END;

/CONTROL/ TEST = BEGIN
                  ...
                  WITH rc DO
                  PRINT ( CCLASS);
                  ...
                 END;
  ...

/EXEC/ SIMUL;
```

NAME

CHANGEDATE - Returns the date of the last customer state modification: active, waiting or blocked.

SYNTAX

[*customer.*]CHANGEDATE

DESCRIPTION

Returns a real corresponding to the date of the last customer state modification: active, waiting or blocked.

customer is an expression representing a type CUSTOMER entity. This specification can be omitted, in a service specification or in a WITH structure.

EVALUATION

During the execution of the algorithmic language with a simulation resolution.

WARNING

- This attribute can be used only in a simulation execution.
- This attribute cannot be modified by the user.

SEE ALSO

ENTERDATE - ACTIVETIME - BLOCKTIME

EXAMPLE

```
/DECLARE/ QUEUE A,B,C;

/STATION/ NAME= A;
          TYPE= RESSOURCE;

/STATION/ NAME= B;
          TRANSIT= OUT;
          SERVICE= BEGIN
                     IF (C.FIRST.BLOCKED) THEN
                     PRINT ("Waiting length",TIME- C.FIRST.CHANGEDATE);
                     EXP (0.5);
                   END;
/STATION/ NAME= C;
          SERVICE= BEGIN
                     P(A);
                     EXP (0.5);
                   END


/CONTROL/ TMAX= 10.0;

/EXEC/ SIMUL;
```

NAME

CLREJECT - References the class of customer responsible of the reject, of a limited capacity queue.

SYNTAX

[*customer.*]CLREJECT

DESCRIPTION

When a customer has been rejected because of a limited capacity, the procedure associated to the reject treatment is executed. The CLREJECT attribute, can be used only in reject sequence, it returns the class of customer responsible of the reject. If the reject is due to a global limited capacity (no class is specified), the attribute returns NIL.

This attribute can be used only in a reject sequence.

customer is an expression representing a type CUSTOMER entity. This specification can be omitted, in a service specification or in a WITH structure.

EVALUATION

During the execution of the algorithmic language with a simulation resolution.

WARNING

- An error is generated if the attribute is used out of a reject sequence.
- This attribute can be used only in a simulation execution.
- This attribute cannot be modified by the user.

SEE ALSO

QREJECT - CPREEMPT - CAPACITY - REJECT

EXAMPLE

```
/DECLARE/ QUEUE QLIM;
          CLASS X;

/STATION/ NAME = QLIM;
          CAPACITY (X) = 3;
          REJECT (X) = BEGIN
                         ...
                         PRINT ("Reject of class ",CUSTOMER.CLREJECT);
                         & print class X
                         ...
                       END ;
```

CONCSETN

NAME

CONCSETN - Returns a customer concurrency-set number.

SYNTAX

[*customer.*]CONCSETN

DESCRIPTION

If the access to a server is managed by a concurrency mechanism, a number, specifying a concurrency-set, is affected to any customer entering in the station. This number is a random number or is depending on the user specification (EXCLUDE option).

The attribute CONCSETN returns a customer concurrency-set number or 0 if no concurrency-set is affected to the customer.

customer is an expression representing a type CUSTOMER entity. This specification can be omitted, in a service specification or in a WITH structure.

EVALUATION

During the execution of the algorithmic language with a simulation resolution.

NOTES

The concurrency-set number is defined when the customer transits to a station. This number is deleted when the customer enters in another station.

This attribute can be used even if no concurrency mechanism is specified for the station.

WARNING

- This attribute can be used only in a simulation execution.
- This attribute cannot be modified by the user.

SEE ALSO

EXCLUDE

EXAMPLE

```
/DECLARE/ QUEUE A;
          CLASS X1, X2, X3;


/STATION/ NAME = A;
          TYPE = MULTIPLE (3);
          SCHED = FIFO, FEFS,  EXCLUDE (X1, 0.1, X2, 0.3, X3, 0.05), (X1, 0.2,
                  X2, 0.1, X3, 0.3), (X1, 0.4, X2, 0.2, X3, 0.4);
          SERVICE = BEGIN
                      IF (CONCSETN<>0) THEN
                         PRINT ("concurrency-set:",CONCSETN);
                       ....
                    END;
```

Specifies three different concurrency-sets defined by customer classes and probabilities.

CPREEMPT

NAME

CPREEMPT - References the customer responsible of a customer reject.

SYNTAX

[*customer.*]CPREEMPT

DESCRIPTION

This attribute returns a reference on the customer responsible of the specified customer reject from a limited capacity station associated to a preemption scheduling (SCHED=PREEMPT).

The CPREEMPT attribute, can be used only in reject sequence, it returns the class of customer responsible of the reject. If the reject is due to a global limited capacity (no class is specified), the attribute returns NIL.

customer is an expression representing a type CUSTOMER entity. This specification can be omitted, in a service specification or in a WITH structure.

EVALUATION

During the execution of the algorithmic language with a simulation resolution.

WARNING

- An error is generated if the attribute is used out of a reject sequence.
- This attribute can be used only in a simulation execution.
- This attribute cannot be modified by the user.

SEE ALSO

QREJECT - CLREJECT - CAPACITY - REJECT - SKIP

EXAMPLE

```
 /DECLARE/ QUEUE QLIM;
           CLASS X;

 /STATION/ NAME = QLIM;
           SCHED = FIFO, PREEMPT;
           CAPACITY (X) = 3;
           REJECT (X) =
BEGIN
  ...
  PRINT ("Reject because of ", CPREEMPT," customer");
  ...
END ;
```

NAME

CPRIOR - Returns a customer priority level.

SYNTAX

[*customer.*]CPRIOR

DESCRIPTION

This attribute returns an integer representing a customer priority level.

customer is an expression representing a type CUSTOMER entity. This specification can be omitted, in a service specification or in a WITH structure.

EVALUATION

During the execution of the algorithmic language with a simulation resolution.

NOTES

A customer priority is specified statically by PRIOR (/STATION/ parameter) and dynamically by PRIOR (procedure), TRANSIT (procedure), AFTCUST and BEFCUST.

WARNING

- This attribute can be used only in a simulation execution.
- This attribute cannot be modified by the user.

SEE ALSO

CUSTOMER - SERVICE - PRIOR (parameter) - PRIOR (procedure) - TRANSIT (procedure) - AFTCUST - BEFCUST - WITH

EXAMPLE

```
 /DECLARE/ QUEUE q1, q2 ;

/STATION/ NAME = q1;
          SERVICE = BEGIN
                      ...
                      PRINT ( CPRIOR);
                      ...
                      PRIOR (rc.CPRIOR);
                      ...
                    END;

 /CONTROL/ TEST = BEGIN
                     ...
                     WITH rc DO
                     i :=  CPRIOR + 4;
                     ...
                   END;

/EXEC/ SIMUL;
```

NAME

CQUEUE - References the station containing a customer.

SYNTAX

[*customer.*]CQUEUE

DESCRIPTION

This attribute returns a reference on the station which contains the customer.

customer is an expression representing a type CUSTOMER entity. This specification can be omitted, in a service specification or in a WITH structure.

EVALUATION

During the execution of the algorithmic language with a simulation resolution.

NOTES

NIL is returned if the customer has been created by the NEW function and not sent yet in a station.

WARNING

- This attribute can be used only in a simulation execution.
- This attribute cannot be modified by the user.

SEE ALSO

NEW - CUSTOMER - SERVICE - WITH

EXAMPLE

```
/DECLARE/ QUEUE q1, q2;

/STATION/ NAME = q1;
          SERVICE = BEGIN
                      ...
                      TRANSIT (NEW (CUSTOMER), rc.CQUEUE);
                      ...
                      TRANSIT ( CQUEUE);  & equivalent to : TRANSIT (q1)
                      ...
                    END;

/CONTROL/ TEST = BEGIN
                   ...
                   WITH rc DO
                   PRINT ( CQUEUE);
                   ...
                 END;
  ...

/EXEC/ SIMUL;
```

NAME

ENTERDATE - Returns the arrival date of a customer in the current station.

SYNTAX

[*customer.*]ENTERDATE

DESCRIPTION

Returns a real value corresponding to the arrival date of a customer in a station.

If the customer is in a source station, the attribute value is the date of the beginning of the source service.

If the customer is created when the simulation is initialized, (INIT option of /STATION/ command) or by a NEW function but not sent yet to another station, the returned value is 0.

customer is an expression representing a type CUSTOMER entity. This specification can be omitted, in a service specification or in a WITH structure.

EVALUATION

During the execution of the algorithmic language with a simulation resolution.

NOTES

A transition to the departure station by a TRANSIT procedure, is considered as an arrival in a new station.

WARNING

- This attribute can be used only in a simulation execution.
- This attribute cannot be modified by the user.

SEE ALSO

CHANGEDATE - ACTIVETIME - BLOCKTIME

EXAMPLE

```
/DECLARE/ REAL WAITING;

/STATION/ NAME = ...
          SERVICE = BEGIN
                      WAITING : = TIME - ENTERDATE;
                      PRINT ("waiting time:", WAITING);
                      ...
                    END ;
```

NAME

FATHER - References the father of a customer.

SYNTAX

[*customer.*] FATHER

DESCRIPTION

This attribute returns a reference on the father of a customer.

customer is an expression representing a type CUSTOMER entity. This specification can be omitted, in a service specification or in a WITH structure.

EVALUATION

During the execution of the algorithmic language with a simulation resolution.

NOTES

- NIL is returned if the customer has no father, and DELETED if the father has been deleted.
- This attribute is managed by QNAP2 when creation operations are performed: NEW (CUSTOMER) or when the customer leaves the network.

WARNING

- This attribute can be used only in a simulation execution.
- This attribute cannot be modified by the user.

SEE ALSO

CUSTOMER - SERVICE - SON - NEW - TRANSIT - WITH

EXAMPLE

```
/DECLARE/ QUEUE q1, q2;
          REF CUSTOMER rc;

/STATION/NAME = q1;
         SERVICE =
 BEGIN
   ...
   IF FATHER <> NIL THEN         & Has the customer a father ?
      ...
   rc := rc.FATHER;
   ...
 END;

/CONTROL/ TEST = BEGIN
                   ...
                   WITH q2.FIRST DO
                     PRINT ( FATHER);
                   ...
                 END;
 ...

 /EXEC/SIMUL;
```

NAME

INSERVICE - Returns information about the affectation of a server to a customer.

SYNTAX

[*customer.*] INSERVICE

DESCRIPTION

This attribute returns a boolean which value is TRUE if the customer executes a service (blocking or not) otherwise the returned value is FALSE.

customer is an expression representing a type CUSTOMER entity. This specification can be omitted, in a service specification or in a WITH structure.

EVALUATION

During the execution of the algorithmic language with a simulation resolution.

NOTES

In a service sequence, the value associated to the current customer is TRUE.

WARNING

- This attribute can be used only in a simulation execution.
- This attribute cannot be modified by the user.

SEE ALSO

CUSTOMER - SERVICE - BLOCKED - WITH

EXAMPLE

```
/DECLARE/ QUEUE q1, q2;
          REF CUSTOMER rc;

/STATION/ NAME = q1;
          SERVICE =  BEGIN
                       ...
                       PRINT ( INSERVICE);   & the value is TRUE in
                                             & every cases
                       ...
                       IF rc.INSERVICE THEN
                         ...
                     END;

/CONTROL/ TEST = BEGIN
                   ...
                   WITH rc DO
                     IF  INSERVICE THEN
                       ...
                 END;
 ...

/EXEC/ SIMUL;
```

NAME

`NEXT` - References the next customer in a queue.

SYNTAX

[*customer.*]`NEXT`

DESCRIPTION

This attribute returns a reference on the next customer in the specified customer queue.

customer is an expression representing a type `CUSTOMER` entity. This specification can be omitted, in a service specification or in a `WITH` structure.

EVALUATION

During the execution of the algorithmic language with a simulation resolution.

NOTES

`NIL` is returned if the specified customer in the last one in the queue.

WARNING

- This attribute can be used only in a simulation execution.
- This attribute cannot be modified by the user.

SEE ALSO

`PREVIOUS - CUSTOMER - SERVICE - WITH`

EXAMPLE

```
/DECLARE/ QUEUE q1, q2;
          REF CUSTOMER rc;

/STATION/ NAME = q1;
          SERVICE =
BEGIN
  ...
  IF  NEXT <> NIL THEN   & the customer in service is
     rc := rc.NEXT;      & not the last in the queue.
  ...
END;

/CONTROL/ TEST = BEGIN
                   ...
                   WITH q2. FIRST DO
                     PRINT ( NEXT);
                   ...
                 END;
 ...

/EXEC/ SIMUL;
```

NAME

PREVIOUS - References the previous customer in a queue.

SYNTAX

[*customer.*] PREVIOUS

DESCRIPTION

This attribute references on the previous customer in the specified customer queue.

customer is an expression representing a type CUSTOMER entity. This specification can be omitted, in a service specification or in a WITH structure.

EVALUATION

During the execution of the algorithmic language with a simulation resolution.

NOTES

NIL is returned if the specified customer in the last one in the queue.

WARNING

- This attribute can be used only in a simulation execution.
- This attribute cannot be modified by the user.

SEE ALSO

NEXT - CUSTOMER - SERVICE - WITH

EXAMPLE

```
/DECLARE/ QUEUE q1, q2;
          REF CUSTOMER rc;

/STATION/ NAME = q1;
          SERVICE = BEGIN
                       ...
                       IF PREVIOUS <> NIL THEN  & the customer in service
                                                & is not the first in q1
                         rc := rc.PREVIOUS;
                       ...
                     END;

/CONTROL/ TEST = BEGIN
                   ...
                   WITH q2. LAST DO
                   PRINT ( PREVIOUS);
                   ...
                 END;
 ...

/EXEC/ SIMUL;
```

NAME

QREJECT - Returns the queue which has rejected a customer because of a limited capacity.

SYNTAX

[*customer.*]QREJECT

DESCRIPTION

When a customer has been rejected because of a limited capacity, the procedure associated to the reject treatement is executed. The QREJECT attribute, can be used only in a reject sequence, it returns the queue which has rejected a customer because of a limited capacity.

This attribute can be used only in a reject sequence.

customer is an expression representing a type CUSTOMER entity. This specification can be omitted, in a service specification or in a WITH structure.

EVALUATION

During the execution of the algorithmic language with a simulation resolution.

WARNING

- An error is generated if the attribute is used out of a reject sequence.
- This attribute can be used only in a simulation execution.
- This attribute cannot be modified by the user.

SEE ALSO

CLREJECT - CPREEMPT - CAPACITY - REJECT

EXAMPLE

```
/DECLARE/ QUEUE QLIM;

/STATION/ NAME = QLIM;
          CAPACITY = 3;
          REJECT =
BEGIN
  PRINT ("Reject because of limited capacity of file", QREJECT);
  & QLIM is returned
  ...
END;
```

RESPTIME

NAME

RESPTIME - Returns the time spent by a customer in a station.

SYNTAX

[*customer.*]RESPTIME

DESCRIPTION

Returns in real, for the considered customer, the time spent in the station since the arrival of the customer in the station.

customer is an expression representing a type CUSTOMER entity. This specification can be omitted, in a service specification or in a WITH structure.

EVALUATION

During the execution of the algorithmic language with a simulation resolution.

SEE ALSO

ENTERDATE - CHANGEDATE - ACTIVETIME - BLOCKTIME

EXAMPLE

```
/DECLARE/ QUEUE B;

/STATION/ NAME= B;
          TRANSIT= OUT;
          SERVICE= BEGIN
                     EXP (0.5);
                     PRINT ("TRESP1=",TIME-CUSTOMER.ENTERDAT);
                     PRINT ("TRESP2=",CUSTOMER.RESPTIME);
                   END;

/CONTROL/ TMAX= 10.0;

/EXEC/ SIMUL;
```

NAME

SON - References the last customer created by a customer.

SYNTAX

[*customer.*] SON

DESCRIPTION

This attribute references the last son customer created by a customer.

customer is an expression representing a type CUSTOMER entity. This specification can be omitted, in a service specification or in a WITH structure.

EVALUATION

During the execution of the algorithmic language with a simulation resolution.

NOTES

- NIL is returned if the customer has currently no son, and DELETED if the last son has been deleted.
- The modifications of this attribute are managed by QNAP2 during customer creation operations: NEW (CUSTOMER) or the output of the network.

WARNING

- This attribute can be used only in a simulation execution.
- This attribute cannot be modified by the user.

SEE ALSO

CUSTOMER - SERVICE - FATHER - NEW - TRANSIT - WITH

EXAMPLE

```
/DECLARE/ QUEUE q1, q2;
          REF CUSTOMER rc;

/STATION/ NAME = q1;
          SERVICE =
BEGIN
  ...
  IF  SON <> NIL THEN  & Has the customer created a son ?
     ...
  rc := rc.SON;
  ...
END;

/CONTROL/ TEST = BEGIN
                   ...
                   WITH q2.FIRST DO
                   PRINT ( SON);
                   ...
                 END;
 ...

/EXEC/ SIMUL;
```

NAME

STARTED - Returns the beginning date of a customer service.

SYNTAX

[*customer.*] STARTED

DESCRIPTION

This attribute returns a boolean which value is TRUE if the customer executes a service (even if it is suspended) otherwise the returned value is FALSE.

customer is an expression representing a type CUSTOMER entity. This specification can be omitted, in a service specification or in a WITH structure.

EVALUATION

During the execution of the algorithmic language with a simulation resolution.

NOTES

In a service sequence, the value associated to the current customer is TRUE.

WARNING

- This attribute can be used only in a simulation execution.
- This attribute cannot be modified by the user.

SEE ALSO

CUSTOMER - SERVICE - WITH

EXAMPLE

```
 /DECLARE/ QUEUE q1, q2;
           REF CUSTOMER rc;

 /STATION/ NAME = q1;
           SERVICE =
  BEGIN
    ...
    PRINT ( STARTED);  & The value is always TRUE
    ...
    IF rc.STARTED THEN
      ...
  END;

/CONTROL/ TEST = BEGIN
                   ...
                   WITH rc DO
                   IF  STARTED THEN
                   ...
                 END;
  ...

/EXEC/ SIMUL;
```

7.1.2 Attributes of FILE objects

BUFPOSPT	Returns the position of the last character treated in a buffer.
ERRHANDLE	Returns the level of error treatment.
ERRRETRY	Returns the number of possible accesses.
ERRSTATUS	Returns the error level of the last input-output operation.
FILASSGN	Returns the name of the assigned file.
FILPOS	Returns the number of the current record.
HARDIOS	Returns the error level affected by the operating system.
OPENMODE	Returns the open mode of a file.
RECLENGTH	Returns the maximum length of a record.

BUFPOSPT

NAME

BUFPOSPT - Returns the position of the last character treated in a buffer.

SYNTAX

[*file.*]BUFPOSPT

DESCRIPTION

This attribute returns the position of the last character treated in a buffer.
The returned value is an integer, which can be equal to:

- 0 : before and after the buffer treatment
- value greater than RECLENGTH : empty buffer
- -1 : for a file creation.

file is an expression representing a FILE type entity. This specification can be omitted, in a service specification or in a WITH structure.

EVALUATION

During the execution.

NOTES

This attribute is modified by QNAP2 for any input/output operation.

WARNING

This attribute cannot be modified by the user.

SEE ALSO

FILE - GETLN - GET - WRITE - WRITELN

EXAMPLE

```
/DECLARE/ FILE f;

/EXEC/ BEGIN
         FILASSIGN (f, "file.dat");
         OPEN (f, 1);
         WITH f DO
           PRINT ( BUFPOSPT);
         PRINT (GET (f, INTEGER));
         PRINT (f.BUFPOSPT);
         PRINT (GET (f, INTEGER));
         PRINT (f.BUFPOSPT);
         PRINT (GETLN (f, INTEGER));
         PRINT (f.BUFPOSPT);
       END;
```

File.dat content:

```
12 13 14 15
```

Results :

```
-1
12
2
13
5
14
121
```

ERRHANDLE

NAME

ERRHANDLE - Returns the level of error treatment.

SYNTAX

[*file.*] ERRHANDLE

DESCRIPTION

This attribute returns the level of error treatment for a file.
The returned value is an integer which possible value are:

- <=0 : no error treatment
- = 1 : treatment of errors due to the end of the file
- = 2 : treatment of errors due to conversion or detected by QNAP2
- = 3 : treatment of errors detected by the operating system.

file is an expression representing a FILE type entity. This specification can be omitted, in a service specification or in a WITH structure.

EVALUATION

During the execution.

NOTES

This attribute is modified by the FILSETERR procedure.

WARNING

This attribute cannot be modified by the user.

SEE ALSO

FILE - FILSETERR - ERRSTATUS

EXAMPLE

```
/DECLARE/ FILE f;
          INTEGER i;

/EXEC/ BEGIN
         FILASSIGN (f, "file.dat");
         FILSETERR (f, 1);
         PRINT (f.  ERRHANDLE);       & result: 1 because EOF
         OPEN (f, 1);
         WITH f DO
           PRINT ( ERRHANDLE);        & result: 1 because EOF
         i := GETLN (f, INTEGER);
         WHILE f. ERRSTATUS <> 1 DO   & until the end of the file
                                      & be achieved
           BEGIN
             i : = GETLN (f, INTEGER);
             ...
           END;
       END;
```

ERRRETRY

NAME

ERRRETRY - Returns the number of possible accesses.

SYNTAX

[*file.*]ERRRETRY

DESCRIPTION

This attribute returns the number of possible reading operations on a variable written in a file (GET and GETLN operations) when specified type is not respected.
The returned value is an integer.

file is an expression representing a FILE type entity. This specification can be omitted, in a service specification or in a WITH structure.

EVALUATION

During the execution.

NOTES

- This attribute is modified by the SETRETRY procedure.
- The default value is 5.

WARNING

- This attribute cannot be modified by the user.
- If ERRHANDLE is greater than 2 new reading are not performed.

SEE ALSO

FILE - GETLN - GET - ERRHANDLE - SETRETRY

EXAMPLE

```
/DECLARE/ FILE f;
          INTEGER i;

/EXEC/ BEGIN
         FILASSIGN (f, "file.dat");
         OPEN (f, 1);
         SETRETRY (f, 2);
         WITH f DO
           PRINT (ERRRETRY); & result : 2
         i := GETLN (f, INTEGER);
         ...
       END;
```

If 3.5 is the first result, QNAP2 will suggest a second reading.

ERRSTATUS

NAME

ERRSTATUS - Returns the error level of the last input-output operation.

SYNTAX

[*file.*]ERRSTATUS

DESCRIPTION

This attribute returns the error level of the last input/output operation.
The returned value is an integer which possible values are:

- <= 0 : no error
- = 1 : end of file
- = 2 : conversion error or error detected by QNAP2
- = 3 : error detected by host system.

file is an expression representing a FILE type entity. This specification can be omitted, in a service specification or in a WITH structure.

EVALUATION

During the execution.

NOTES

This attribute is modified after every input/output operation.

WARNING

This attribute cannot be modified by the user.

SEE ALSO

FILE - FILSETERR - GETLN - GET - PRINT - WRITE - WRITELN

EXAMPLE

```
 /DECLARE/ FILE f ;
           INTEGER i ;

/EXEC/ BEGIN
         FILASSIGN (f, "file.dat") ;
         FILSETERR (f, 1) ;
         OPEN (f, 1) ;
         WITH f DO
            PRINT ( ERRSTATUS) ;    & result : 0 because no error
         i := GETLN (f, INTEGER) ;
         WHILE f.ERRSTATUS <> 1 DO  & until the end of the file be
                                    & achieved
           BEGIN
             i : = GETLN (f, INTEGER) ;
              ...
           END ;
       END ;
```

FILASSGN

NAME

FILASSGN - Returns the name of the assigned file.

SYNTAX

[*file.*]FILASSGN

DESCRIPTION

This attribute returns the name of the file assigned to *file*.

The returned value is a string.

file is an expression representing a type FILE entity. This specification can be omitted, in a service specification or in a WITH structure.

EVALUATION

During the execution.

NOTES

- This attribute is modified by the FILASSGN procedure.
- Before any assigned treatment, the value of this attribute is "".

WARNING

This attribute cannot be modified by the user.

SEE ALSO

FILE - FILASSIGN

EXAMPLE

```
/DECLARE/ FILE f;
          INTEGER i;

/EXEC/ BEGIN
         WITH f DO
           PRINT (FILASSGN) ;           & result :  "" because no
                                        & assigned file
         FILASSIGN (f, "file.dat");
         PRINT (f.FILASSGN) ;           & result : file.dat
         ...
       END ;
```

NAME

FILPOS - Returns the number of the current record.

SYNTAX

[*file*.] FILPOS

DESCRIPTION

This attribute returns the number of the current record.

The returned value is an integer.

file is an expression representing a FILE type entity. This specification can be omitted, in a service specification or in a WITH structure.

EVALUATION

During the execution.

NOTES

- This attribute is modified after any input/output operation.
- Before any input/output operation the attribute value is 0.

WARNING

This attribute cannot be modified by the user.

SEE ALSO

FILE - GETLN - PRINT - WRITELN

EXAMPLE

```
/DECLARE/ FILE f;
          INTEGER i;

/EXEC/ BEGIN
         FILASSIGN (f, "file.dat");
         OPEN (f, 1);
         WITH f DO
           PRINT( FILPOS);            & result : 0 because no I/O
         i : = GETLN (f, INTEGER);
         PRINT (f.  FILPOS);          & result : 1
         ...
       END;
```

HARDIOS

NAME

HARDIOST - Returns the error level affected by the operating system.

SYNTAX

[*file*.] HARDIOST

DESCRIPTION

This attribute returns the error level affected by the operating system during an input/output operation.

The returned value is an integer depending on the operating system and the FORTRAN library.

file is an expression representing a FILE type entity. This specification can be omitted, in a service specification or in a WITH structure.

EVALUATION

During the execution.

NOTES

- This attribute is modified after any input/output operation.
- The returned value corresponds to IOSTAT returned by FORTRAN.

WARNING

This attribute cannot be modified by the user.

SEE ALSO

FILE - GETLN - GET - ERRHANDLE - ERRSTATUS

EXAMPLE

```
/DECLARE/ FILE f;
          INTEGER i;
/EXEC/ BEGIN
         FILASSIGN (f, "file.dat");
         OPEN (f, 1);
         WITH f DO
           PRINT (HARDIOST);   & result : 0
         i := GETLN (f, INTEGER);
         ...
       END;
```

NAME

OPENMODE - Returns the open mode of a file.

SYNTAX

[*file.*] OPENMODE

DESCRIPTION

This attribute returns the open mode of a file.

The returned value is an integer which possible values are:

- <= 0 : file closed
- = 1 : file opened in reading mode
- = 2 : file opened in writting mode
- = 3 : file opened in reading/writting mode

file is an expression representing a FILE type entity. This specification can be omitted, in a service specification or in a WITH structure.

EVALUATION

During the execution.

NOTES

This attribute is modified by the OPEN and CLOSE procedures.

WARNING

This attribute cannot be modified by the user.

SEE ALSO

FILE - OPEN - CLOSE

EXAMPLE

```
/DECLARE/ FILE f;
          INTEGER i;

/EXEC/ BEGIN
         FILASSIGN (f, "file.dat");
         PRINT (f.OPENMODE);          & result : 0 because the file
                                      & is closed
         OPEN (f, 1);
         WITH f DO
           PRINT ( OPENMODE);         & result : 1 because the file is
                                      & opened in reading mode
       END;
```

NAME

RECLENGTH - Returns the maximum length of a record.

SYNTAX

[*file*.] RECLENGTH

DESCRIPTION

This attribute returns the maximum length of a record.

The returned value is an intger.

file is an expression representing a FILE type entity. This specification can be omitted, in a service specification or in a WITH structure.

EVALUATION

During the execution.

NOTES

- This attribute is modified by the SETBUF procedure.
- The default value is 120.

WARNING

This attribute cannot be modified by the user.

SEE ALSO

FILE - SETBUF - BUFPOSPT

EXAMPLE

```
/DECLARE/ FILE f;
          INTEGER i ;

/EXEC/ BEGIN
         FILASSIGN (f, "file.dat");
         SETBUF (f, 80);
         OPEN (f, 1);
         WITH f DO
           PRINT (RECLENGTH);   & result : 80
         ...
       END;
```

7.1.3 Attributes of QUEUE objects

CAPACITY	Returns the capacity of a queue.
FIRST	References the first customer in a queue.
LAST	References the last customer in a queue.
MULT	Returns the number of servers of a station.
NB	Returns the number of customers in a station.
NBIN	Returns the number of customers entered in a station.
NBINSERV	Returns the number of customers in service in a station.
NBOUT	Returns the number of customers which have left the station.
VALUE	Returns the number of free servers.

NAME

CAPACITY - Returns the capacity of a queue.

SYNTAX

[*queue.*] CAPACITY [(*class*)]

DESCRIPTION

Returns the capacity of a queue, global or for a specific class.

If the capacity of the queue is not limited the returned value is -1.

queue is an expression representing a QUEUE type entity.

EVALUATION

During the execution.

NOTES

This attribute indicates if a limited capacity has been specified for the station.

WARNING

The value of the attribute is affected after the beginning of the resolution.

SEE ALSO

NBIN - NBOUT - NB - NBINSERV - MULT

EXAMPLE

```
/DECLARE/ CLASS X, Y, Z ;

/STATION/
   NAME = ... ;
   CAPACITY = 5 ;
   CAPACITY (X) = 4 ;
   SERVICE = BEGIN
               IF QUEUE.CAPACITY <> 5 THEN
                 PRINT ("ERROR ON GLOBAL CAPACITY") ;

               IF QUEUE.CAPACITY(X) <> 4 THEN
                 PRINT ("ERROR ON CAPACITY ASSOCIATED TO CLASS X") ;
             END ;
```

FIRST

NAME

FIRST - References the first customer in a queue.

SYNTAX

[*queue.*]FIRST

DESCRIPTION

This attribute returns a reference on the first customer in a queue.

queue is an expression representing a QUEUE type entity. This specification can be omitted, in a service specification or in a WITH structure.

EVALUATION

During the execution.

NOTES

At the beginning of a simulation resolution, NIL is returned if no customer have been created by INIT parameter of /STATION/ command.

WARNING

- This attribute can be used only during a simulation.
- The first customer is not always an active customer, especially if the station scheduling is SCHED = LIFO | PRIOR | LIFO,PRIOR.

SEE ALSO

QUEUE - WITH - SERVICE - TYPE - LAST - INIT - SCHED

EXAMPLE

```
 /DECLARE / QUEUE q1, q2, q3;
            REF QUEUE rq;

 /STATION/ NAME = q1;
           TYPE = MULTIPLE (7);
           SERVICE =
BEGIN
  ...
  IF  FIRST <> CUSTOMER THEN
       TRANSIT (q2);
  ...
  IF rq.  FIRST <> NIL THEN
       TRANSIT (q3);
  ...
END;

/CONTROL/ TEST = BEGIN
                   ...
                   WITH q3 DO PRINT ( FIRST);
                   ...
                 END;

...

/EXEC/ SIMUL;
```

LAST

NAME

LAST - References the last customer in a queue.

SYNTAX

[*queue.*]LAST

DESCRIPTION

This attribute returns a reference on the last customer in a queue.

queue is an expression representing a QUEUE type entity. This specification can be omitted, in a service specification or in a WITH structure.

EVALUATION

During the execution.

NOTES

At the beginning of a simulation resolution, NIL is returned if no customer have been created by the INIT parameter of the /STATION/ command.

WARNING

This attribute can be used only during a simulation.

SEE ALSO

QUEUE - WITH - SERVICE - TYPE - FIRST - INIT - SCHED

EXAMPLE

```
 /DECLARE/ QUEUE q1, q2, q3;
           REF QUEUE rq;

 /STATION/ NAME = q1;
           TYPE = MULTIPLE (7);
           SERVICE =
BEGIN
  ...
  IF  LAST <> CUSTOMER THEN TRANSIT (q2);
  ...
  IF rq.  LAST <> NIL THEN TRANSIT (q3);
  ...
END;

/CONTROL/ TEST = BEGIN
                   ...
                   WITH q3 DO PRINT ( LAST);
                   ...
                 END;
...

/EXEC/ SIMUL;
```

MULT

NAME

MULT - Returns the number of servers of a station.

SYNTAX

[*queue*.]MULT

DESCRIPTION

Returns an integer corresponding to the number of servers of a station. The value corresponding to an infinite server is -1.

queue is an expression representing a type QUEUE entity.

EVALUATION

During the execution.

WARNING

The value of the attribute is affected after the beginning of the resolution.

SEE ALSO

TYPE - QUEUE - EXCLUDE

EXAMPLE

```
 /STATION/ NAME = ... ;
           TYPE = MULTIPLE (4);
           SERVICE = BEGIN
                       IF (QUEUE. MULT <> 4) THEN
                         PRINT ("ERROR") ;
                     END ;
```

NAME

NB - Returns the number of customers in a station.

SYNTAX

[*queue.*]NB (*list_of_classes)*

DESCRIPTION

This attribute returns the number of customers in a station.

queue is an expression representing a QUEUE type entity. This specification can be omitted, in a service specification or in a WITH structure.

EVALUATION

During the execution.

NOTES

- CUSTNB function returns the same value when no class is specified.
- At the beginning of the simulation, the value is provided by the INIT parameter of the /STATION/ command.

WARNING

This attribute can be used only in simulation.

SEE ALSO

QUEUE - WITH - SERVICE - CUSTNB - INIT - NBIN - NBOUT - NBINSERV

EXAMPLE

```
 /DECLARE/ QUEUE q1, q2, q3;
           CLASS c1,c2,c3;
           REF QUEUE rq;

 /STATION/ NAME = q1;
           TRANSIT(c1)=q2,c2;
           TRANSIT(c2)=q3,c3;
           SERVICE =
  BEGIN
    ...
    IF  NB > 4 THEN
      TRANSIT (q2);
    ...
    IF rq.  NB > 4 THEN
      TRANSIT (q3);
    ...
  END;

/CONTROL/ TEST = BEGIN
                   ...
                   WITH q3 DO
                     PRINT ( NB(c3));
                    ...
                 END;
 ...

/EXEC/ SIMUL;
```

NAME

NBIN - Returns the number of customers entered in a station.

SYNTAX

[*queue.*]NBIN (*class*);

DESCRIPTION

This attribute returns an integer representing the number of customers entered in a station, for a specific class or for all classes.

For a SEMAPHORE or a RESOURCE station, the returned value is the number of customers which have performed a P operation on the station.

queue is an expression representing a QUEUE type entity. This specification can be omitted, in a service specification or in a WITH structure.

EVALUATION

During the execution.

NOTES

- The CUSTNB function returns the same value when no class is specified.
- At the beginning of the simulation, the value is null.
- The relation : NBIN >= NBOUT is always respected.

WARNING

- This attribute returns a result for a class of customers only if results have been requested for classes (SETSTAT:CLASS procedure).
- This attribute can be used only in simulation.

SEE ALSO

NBOUT - NB - NBINSERV - QUEUE - WITH - SERVICE - TYPE - NBOUT

EXAMPLE

```
/DECLARE/ QUEUE q1, q2, q3;
          CLASSE c1,c2,c3;
          REF QUEUE rq;

/STATION/  NAME = q1;
           TYPE = MULTIPLE (7);
           SERVICE =
  BEGIN
    ...
    IF NBIN > 45 THEN
      TRANSIT (q2);
    ...
    IF rq. NBIN > 104 THEN
      TRANSIT (q3);
    ...
  END;

/CONTROL/ TEST = BEGIN
                   ...
                   WITH q3 DO
                     PRINT (NBIN(c1),NBIN(c2),NBIN(c3));
                   ...
                 END;

 ...

/EXEC/ BEGIN
         SETSTAT:CLASS (q3);
         SIMUL;
       END;
```

NAME

NBINSERV - Returns the number of customers in service in a station.

SYNTAX

[*queue.*] NBINSERV [(*class*)]

DESCRIPTION

This attribute returns an integer corresponding to the number of customers in service in a station, for all classes or only for the specified class.

queue is an expression representing a QUEUE type entity. This specification can be omitted, in a service specification or in a WITH structure.

EVALUATION

During the execution.

NOTES

- This attribute returns a result for a class of customers only if results have been requested for classes (SETSTAT:CLASS procedure).
- This attribute can be used only in simulation.

SEE ALSO

NBIN - NBOUT - NB - MULT - CAPACITY

EXAMPLE

```
/STATION/ NAME = ...;
          TYPE = MULTIPLE (...);
          SERVICE =
BEGIN
  IF (QUEUE.NBINSERV >1) THEN
     BEGIN
       PRINT ((QUEUE.NBINSERV-1)" CUSTOMERS IN SERVICE");
     END
  ELSE
     IF (QUEUE. NBINSERV < 1) THEN
      BEGIN
        PRINT ("ERROR");
      END;
END;
```

NBOUT

NAME

NBOUT - Returns the number of customers which have left the station.

SYNTAX

[*queue.*]NBOUT (*class*);

DESCRIPTION

This attribute returns an integer corresponding to the number of customers which have left the station, for a specific class or for all classes.

For a SEMAPHORE or a RESOURCE station, the returned value is the number of customer which have performed a P operation on the station.

queue is an expression representing a type QUEUE entity. This specification can be omitted, in a service specification or in a WITH structure.

EVALUATION

During the execution.

NOTES

- The CUSTNB function returns the same value when no class is specified.
- At the beginning of the simulation, the value null.
- The relation : NBIN >= NBOUT is always respected.

WARNING

- This attribute returns a result for a class of customers only if results have been requested for classes (SETSTAT:CLASS procedure).
- This attribute can be used only in simulation.

SEE ALSO

QUEUE - WITH - SERVICE - TYPE - NBIN - SERVNB

EXAMPLE

```
/DECLARE/ QUEUE q1, q2, q3 ;
          CLASSE c1,c2;

/EXEC/ BEGIN
         SIMUL;
         PRINT ("number of customer which have left q3:");
         PRINT ("total: ", q3.NBOUT);
         PRINT ("class c1 customers: ", q3.NBOUT(c1));
         PRINT ("class c2 customers: ", q3.NBOUT(c2));
       END;
```

VALUE

NAME

VALUE - Returns the number of free servers.

SYNTAX

[*queue.*]VALUE

DESCRIPTION

This attribute returns an integer representing the number of free servers, for a station.

For a RESOURCE type station, the returned value is the number of free resource units.
For a SEMAPHORE type station, the returned value is the counter value.

queue is an expression representing a type QUEUE entity. This specification can be omitted, in a service specification or in a WITH structure.

EVALUATION

During the execution.

NOTES

At the beginning of the simulation, the returned value is the number of servers specified for the station using the TYPE parameter of /STATION/.

WARNING

This attribute can be used only in simulation.

SEE ALSO

QUEUE - WITH - SERVICE - CUSTNB - INIT

EXAMPLE

```
 /DECLARE/ QUEUE q1, q2, q3;
           REF QUEUE rq;

 /STATION/ NAME = q1;
           TYPE = MULTIPLE (7);
           SERVICE =
  BEGIN
    ...
    IF  VALUE > 4 THEN TRANSIT (q2);
    ...
    IF rq. VALUE > 4 THEN TRANSIT (q3);
    ...
  END;

/CONTROL/ TEST = BEGIN
                   ...
                   WITH q3 DO PRINT ( VALUE);
                   ...
                 END;

 ...

/EXEC/ SIMUL;
```

7.1.4 Attributes of TIMER objects

ACTIARG	Returns the last argument specified for a timer activation.
HANDLER	References the treatment procedure associated to a timer.
STATE	Returns the timer activation type.
TIMPRIOR	Returns the timer priority.

NAME

ACTIARG - Returns the last argument specified for a timer activation.

SYNTAX

[*timer*.]ACTIARG

DESCRIPTION

This attribute returns a real corresponding to the last argument specified in SETTIMER:ABSOLUTE, SETTIMER:RELATIVE, or SETTIMER:CYCLIC procedures, no signification otherwise.

timer is an expression representing a TIMER type entity.

EVALUATION

During the execution.

NOTES

The value of this attribute is used by QNAP2 to set automatically a timer activated by a SETTIMER:CYCLIC procedure.

WARNING

The date of a timer activation is returned by the function GETSIMUL:WAKETIME.

SEE ALSO

SETTIMER:ABSOLUTE - SETTIMER:RELATIVE - SETTIMER:CYCLIC - STATE - GETSIMUL:WAKETIME - HANDLER - TIMPRIOR

EXAMPLE

```
/DECLARE/ TIMER MYTIMER;
  ...

/STATION/ NAME = ...
          SERVICE = BEGIN
                      SETTIMER:CYCLIC (MYTIMER, 10.0);
                      IF (MYTIMER.ACTIARG <> 10.0) THEN
                        PRINT ("ERROR");
                    END;
```

HANDLER

NAME

HANDLER - References the treatment procedure associated to a timer.

SYNTAX

[*timer*.]HANDLER

DESCRIPTION

This attribute returns a reference on the treatment procedure associated to a timer.

timer is an expression representing a type TIMER entity.

EVALUATION

During the execution.

VOIR AUSS

SETTIMER:SETPROC - ACTIARG - TIMPRIOR - STATE

EXAMPLE

```
/DECLARE/ TIMER MYTIMER;
          PROCEDURE TIMHANDLE;
          BEGIN
            ...
          END;

/EXEC/ BEGIN
         SETTIMER:SETPROC (MYTIMER , TIMHANDLE);
         IF (MYTIMER.HANDLER <> TIMHANDLE) THEN
           PRINT ("ERROR");
       END;
```

NAME

STATE - Returns the timer activation type.

SYNTAX

[*timer*.] STATE

DESCRIPTION

This attribute returns an integer corresponding to the timer activation type:

- 0 : timer no activated
- 1 : timer activated by SETTIMER:ABSOLUTE
- 2 : timer activated by SETTIMER:RELATIVE
- 3 : timer activated by SETTIMER:CYCLIC
- 4 : timer activated by SETTIMER:TRACKTIME
- negative value : timer executing its treatment procedure, the value depends on the way it has been activated

timer is an expression representing a type TIMER entity.

EVALUATION

During the execution.

SEE ALSO

SETTIMER:CANCEL - SETTIMER:ABSOLUTE - SETTIMER:RELATIVE - SETTIMER:CYCLIC - SETTIMER:TRACKTIME - ACTIARG - TIMPRIOR

EXAMPLE

```
/DECLARE/ TIMER MYTIMER;
           ...

/STATION/ NAME = ...
          SERVICE = BEGIN
                      SETTIMER:ABSOLUTE (MYTIMER, 10);
                      IF (MYTIMER. STATE <> 1) THEN
                        PRINT ("ERROR");
                    END;
```

TIMPRIOR

NAME

TIMPRIOR - Returns the timer priority.

SYNTAX

[*timer.*] TIMPRIOR

DESCRIPTION

This attribute returns an integer corresponding to the timer priority. This priority can be compared with a customer priority, and is modified by PRIOR procedure.

timer is an expression representing a type TIMER entity.

EVALUATION

During the execution.

NOTES

The default priority is 0.

SEE ALSO

TIMER - STATE - ACTIARG - HANDLER

EXAMPLE

```
/DECLARE/ TIMER MYTIMER;
           ...

/STATION/ NAME = ...
          SERVICE = BEGIN
                      PRIOR (MYTIMER, 10);
                      IF (MYTIMER.TIMPRIOR <> 10) THEN
                        PRINT ("ERROR");
                    END;
```

7.1.5 Other Attributes

DEFHANDLER	References the procedure associated to the exception treatment.
ICLASS	Returns the range of creation of a class.
STATE	Returns a flag state.

DEFHANDLER

NAME

DEFHANDLER - References the procedure associated to the exception treatment.

SYNTAX

[*exception.*]DEFHANDLER

DESCRIPTION

This attribute returns a reference on the procedure associated to the exception treatment. The procedure must be defined without argument.

exception is an expression representing a EXCEPTION type entity.

EVALUATION

During the execution of the algorithmic language of a simulation resolution.

SEE ALSO

SETEXCEPT:CONNECT - SETEXCEPT:HANDLER - EXCEPTION

EXAMPLE

```
/DECLARE/ EXCEPTION EXCP;
          PROCEDURE HANDLER;
          BEGIN
            ...
          END;

/EXEC/ BEGIN
         SETEXCEPT:HANDLER (EXCP, HANDLER);
         IF EXCP. DEFHANDLER <> HANDLER THEN
            PRINT ("ERROR")
       END;
```

NAME

ICLASS - Returns the range of creation of a class.

SYNTAX

[*class.*]ICLASS

DESCRIPTION

This attribute returns an integer corresponding to the range of creation of a class.

class is an expression representing a CLASS type entity. This specification can be omitted, in a service specification or in a WITH structure.

EVALUATION

During the execution.

NOTES

The creation order depends on static and dynamic (NEW) treatments.

WARNING

- This attribute can be used only in a simulation execution.
- This attribute cannot be modified by the user.

SEE ALSO

CLASS - SERVICE - NEW - WITH

EXAMPLE

```
/DECLARE/ QUEUE q1, q2;

/STATION/ NAME = q1;
          SERVICE =
BEGIN
  ...
  PRINT ( ICLASS);
  ...
  TRANSIT (q2, c (rc.  ICLASS));
  ...
END;

/CONTROL/ TEST = BEGIN
                   ...
                   WITH rc DO
                     i :=  ICLASS + 4;
                   ...
                 END;
 ...

/EXEC/ SIMUL;
```

NAME

STATE - Returns a flag state.

SYNTAX

[*flag.*]STATE

DESCRIPTION

This attribute returns a boolean which is FALSE if the flag is RESET (blocking state) and TRUE otherwise.

flag is an expression representing a FLAG type entity. This specification can be omitted, in a service specification or in a WITH structure.

EVALUATION

During the execution.

NOTES

- SET and RESET operations can modify this attribute value.
- At the beginning of the simulation the value is FALSE.

WARNING

- This attribute can be used only in a simulation execution.
- This attribute cannot be modified by the user.

SEE ALSO

FLAG - SERVICE - SET - RESET - WAIT - WAITAND - WAITOR - FREE - WITH

EXAMPLE

```
/DECLARE/ QUEUE q1, q2;
/STATION / NAME = q1;
           SERVICE =
 BEGIN
   ...
   IF F1. STATE THEN
      RESET (F1);
   ...
 END;

/CONTROL/ TEST = BEGIN
                   ...
                   WITH rf DO
                     IF  STATE THEN
                        ...
                 END;
 ...

/EXEC/ SIMUL;
```

7.2 Operators

:=	Assignment operator.
Comparison	Equal, not equal, lower or equal, lower, greater or equal, greater operators.
Logic	Logical `and`, `or` and `not` operators.
Mathematic	Addition, substraction, multiplication, division and exponentiation operators.
::	Used to specify that a reference should point at an object of a sub-type.
FOR	Repetitive loop statement.
IF	Conditional statement.
IN	Subtyping test operator.
IS	Type recognition test operator.
WHILE	Conditional loop statement.
WITH	Implicit access to the attributes of an object statement.
WITH	Item selection within a list operator.

NAME

`Assignment :=` - Assignment operator.

SYNTAX

expr1`:=` *expr2*

DESCRIPTION

This operator is the assignment operator that allows value to be assigned to variables.

expr1 and *expr2* are expressions of the same type (scalar or not).

If *expr1* is a reference to an object, *expr2* must evaluate to an object or a reference to an object of the same type or one of its subtypes.

EVALUATION

During execution.

WARNING

This operator can only be used within algorithmic code.

EXAMPLE

```
/DECLARE/ BOOLEAN b1, b2;
          INTEGER i, j;
          REF QUEUE rq1;
          QUEUE q2;
          QUEUE OBJECT SUBQUEUE;
            ...
          END;
          SUBQUEUE q3;

/EXEC/ BEGIN
         b1 := b2;
         ...
         i  := j;
         rq1 := q2;
         ...
         rq1 := q3;
       END;
```

NAME

`Comparison-operators` = <> <= < >= > - Equal, not equal, lower or equal, lower, greater or equal, greater operators.

SYNTAX

expr1 = *expr2*

expr1 <> *expr2*

expr1 <= *expr2*

expr1 < *expr2*

expr1 >= *expr2*

expr1 > *expr2*

DESCRIPTION

Those operators make comparison tests on two expressions.

expr1 and *expr2* must evaluate to the same type (scalar or not).

The result of the comparison is a `BOOLEAN` value. If the comparison expression is true, the result is `TRUE`, otherwise `FALSE`.

EVALUATION

During execution.

WARNING

This operator can only be used within algorithmic code.

NOTES

`FALSE` is lower than `TRUE`.

EXAMPLE

```
/DECLARE/ BOOLEAN b1, b2;
          INTEGER i, j;
          REF QUEUE rq1, rq2;
          REAL r1,r2;
   ...

/EXEC/ BEGIN
         IF (b1 = b2) THEN PRINT ("ok");
```

```
          IF (i = j) THEN
            PRINT (rq1 = rq2);
        END;

/EXEC/ BEGIN
          IF (b1 <> b2) THEN PRINT ("ok");
          IF (i <> j) THEN
            PRINT (rq1 <> rq2);
        END;

/EXEC/ BEGIN
          IF (r1 <= r 2) THEN PRINT ("ok");
          IF (i <= j) THEN
          PRINT (b1 <= b2);
        END;

/EXEC/ BEGIN
          IF (r1 < r2) THEN PRINT ("ok");
          IF (i < j) THEN
          PRINT (b1 < b2);
        END;

/EXEC/ BEGIN
          IF (r1 >= r2) THEN PRINT ("ok");
          IF (i >= j) THEN
          PRINT (b1 >= b2);
        END;

/EXEC/ BEGIN
          IF (r1 > r2) THEN PRINT ("ok");
          IF (i > j) THEN
          PRINT (b1 > b2);
        END;
```

NAME

Logical-Operators AND OR NOT - Logical and, or and not operators.

SYNTAX

expr1 AND *expr2*

expr1 OR *expr2*

NOT (*expr1*)

DESCRIPTION

Those operators compute logical operation on expressions *expr1* and *expr2*.

expr1 and *expr2* are expressions evaluating to BOOLEAN.

The result is a BOOLEAN value.

NOTES

The QNAP2 compiler systematically evaluates both expressions.

EVALUATION

During execution.

WARNING

This operator can only be used within algorithmic code.

EXAMPLE

```
/DECLARE/ BOOLEAN b1, b2;
          ...

/EXEC/ BEGIN
         IF (b1 AND b2) THEN PRINT("ok");
         b2 := b1 AND TRUE;
         PRINT (TRUE AND TRUE);    & result TRUE
         PRINT (TRUE AND FALSE);   & result FALSE
         PRINT (FALSE AND TRUE);   & result FALSE
         PRINT (FALSE AND FALSE);  & result FALSE
       END;

/EXEC/ BEGIN
         IF (b1 OR b2) THEN PRINT ("ok");
         b2 := b1 OR FALSE;
         PRINT (TRUE OR TRUE);     & result TRUE
```

```
          PRINT (TRUE OR FALSE);    & result TRUE
          PRINT (FALSE OR TRUE);    & result TRUE
          PRINT (FALSE OR FALSE);   & result FALSE
        END;

/EXEC/  BEGIN
          IF  NOT (b1) THEN PRINT ("ok");
          b2 :=  NOT (b1);
          PRINT ( NOT (TRUE));      & result FALSE
          PRINT ( NOT (FALSE));     & result TRUE
        END;
```

NAME

`Mathematical-Operators` + − ∗ / ∗∗ - Addition, substraction, multiplication, division and exponentiation operators.

SYNTAX

expr1 + *expr2*

expr1 − *expr2*

expr1 ∗ *expr2*

expr1 / *expr2*

expr1 ∗∗ *expr2*

DESCRIPTION

Those operators calculate respectively the addition, substraction, multiplication, division and exponentiation of the two mathematical expressions *expr1* and *expr2*.

During evaluation, the priority level of the addition and substraction is lower than the priority level of the multiplication and division, which is lower than the exponentiation one.

expr1 and *expr2* are expressions evaluating to either `INTEGER` or `REAL`.

EVALUATION

During execution.

WARNING

If both expressions evaluate to an `INTEGER` value, the result will be of type `INTEGER`. The result is otherwise of type `REAL`.

EXAMPLE

```
/DECLARE/ INTEGER i, j;
          ...

/EXEC/ BEGIN
         PRINT(i + 2);     & INTEGER result
         PRINT(i + 2.4);   & REAL result
       END;

/EXEC/ BEGIN
         j : = i  - 2;
         PRINT (i  - 2);   & INTEGER result
```

```
          PRINT (5.1  - 1); & REAL result: 4.1
        END;

/EXEC/  BEGIN
          j : = i  * 2;
          PRINT (i  * 2);   & INTEGER result
          PRINT(3.1  * 2);  & REAL result: 6.2
        END;

/EXEC/  BEGIN
          j := i ** 2;      & INTEGER result
          PRINT (i ** 2);   & INTEGER result
          PRINT (3.1 ** 2); & REAL result
        END;
```

NAME

`Type-specifier ::` - Used to specify that a reference should point at an object of a sub-type.

SYNTAXE

ptr1 `::` *type1*

DESCRIPTION

This operateur is used to specify that a reference to an object should point at an object of a sub-type. The result of the expression is a reference to the object of the sub-type.

ptr1 is a reference, `REF` type entity.

type1 is a sub-type of standard types: `CUSTOMER, QUEUE,...`, defined by the user.

EVALUATION

During execution.

WARNING

This operator can only be used within algorithmic code.

SEE ALSO

`IS - IN`

EXEMPLE

```
/DECLARE/ QUEUE A,B,...;
          CUSTOMER OBJECT person;
            REAL d_entree;
          END;
          REF person @person;

/STATION/NAME=B;
         INIT=1;
         TRANSIT=OUT;
         SERVICE=BEGIN
                  @person:=NEW(person);
                  @person.d_entree:=TIME;
                  TRANSIT(@person,A);
                 END;

/STATION/NAME=A;
         TRANSIT=OUT;
         SERVICE=BEGIN
                   CST(2);
```

```
                    WITH CUSTOMER::person DO
                      PRINT('date entree: ',d_entree);
                    END;

/EXEC/ BEGIN
         ....
         SIMUL;
         ....
       END;
```

NAME

FOR - Repetitive loop statement.

SYNTAX

FOR *variable* := *list* DO *statement-list*
FOR *variable* := *value1* STEP *pas* UNTIL *value2* DO *statement-list*

DESCRIPTION

This statement allows to execute a statement or statement list for a list of values of the index variable.

list is a list expression whose result is a list of value of the same type.

EVALUATION

During execution.

WARNING

This statement can only be used within algorithmic code.

SEE ALSO

IF - WHILE

EXAMPLE

```
/DECLARE/ INTEGER i, n, sum;
          REAL t (10);
          QUEUE q1, q2;
          REF QUEUE rq;
          ...

/EXEC/ BEGIN
         ...
         FOR  i : = 1  STEP 1  UNTIL n  DO
            PRINT (i);
         ...
         FOR i : = (1, 3, 4, 7)  DO
            BEGIN & complex statement-list: beginning with BEGIN
              t (i) : = i-1;
              PRINT ( t (i) );
            END;  & and ending with END
        END;

         ...

/STATION/ NAME = q1;
          SERVICE = BEGIN
                      ...
                      sum := 0;
                      FOR rq := ALL QUEUE DO & list expression returning all queues
                         sum := sum + CUSTNB ( rq ); & sum := sum + CUSTNB(q1);
                                                     & sum := sum + CUSTNB(q2);
                      ...
                      FOR rq : = ALL QUEUE WITH CUSTNB (QUEUE) <> 0  DO
                         TRANSIT ( rq.FIRST, q1 );
                    END;
```

NAME

IF - Conditional statement.

SYNTAX

IF *condition* THEN *statement-list*
IF *condition* THEN *statement-list1* ELSE *statement-list2*

DESCRIPTION

This statement allows to execute a statement or statement list (*statement-list* or *statement-list1*) if some condition is satisfied (*condition* evaluates to TRUE), and possibly another statement or statement list (*statement-list2*) if the condition is not satisfied (*condition* evaluates to FALSE).

condition is an expression evaluating to BOOLEAN.

EVALUATION

During execution.

WARNING

- This statement can only be used within algorithmic code.
- The last statement right before the ELSE keyword shouldn't end with a ";".

SEE ALSO

FOR - WHILE

EXAMPLE

```
/DECLARE/ INTEGER i;
          QUEUE q1, q2;
          REF QUEUE rq;
          ...

/EXEC/ BEGIN
         ...
         IF (i = 3) OR (i = 7)  THEN
           PRINT (i);
         ...
       END;
          ...

/STATION/ NAME = q1;
          SERVICE = BEGIN
                      ...
                      IF rq <> NIL  THEN
                        TRANSIT ( rq )  & no ";" before ELSE
                      ELSE
                        BEGIN           & complex statement-list: beginning with BEGIN
                          CST( 10 );
                          TRANSIT( q2 );
                        END;            & and ending with END
                      ...
                    END;
```

NAME

IN - Subtyping test operator.

SYNTAX

object-reference IN *type*

DESCRIPTION

This operator tests if the object referenced by *object-reference* is an object of type *type* or of one of its subtype.

The result is a BOOLEAN value.

EVALUATION

During execution.

SEE ALSO

IS

EXAMPLE

```
/DECLARE/ CUSTOMER OBJECT message;
            INTEGER long;
          END;

          message OBJECT request;
            REF QUEUE destin;
          END;

          REF CUSTOMER rc;
          QUEUE q;

/STATION/ NAME = q;
          SERVICE = BEGIN
                      ...
                      IF rc IN message THEN
                        rc :: message.long : = 64;
                      ...
                    END;
```

NAME

IS - Type recognition test operator.

SYNTAX

object-reference IS *type*

DESCRIPTION

This operator tests if the object referenced by *object-reference* is an object of type *type*.

The result is a BOOLEAN value.

EVALUATION

During execution.

SEE ALSO

IN

EXAMPLE

```
/DECLARE/ CUSTOMER OBJECT message;
            INTEGER long;
          END;

          message OBJECT request;
            REF QUEUE destin;
          END;

          REF CUSTOMER rc;
          QUEUE q;

/STATION/ NAME = q;
          SERVICE = BEGIN
                      ...
                      IF rc IS request THEN
                        TRANSIT (rc :: request.destin);
                      ...
                    END;
```

NAME

WHILE - Conditional loop statement.

SYNTAX

WHILE *condition* DO *statement-list*

DESCRIPTION

This statement allows to execute a statement or statement list while a certain condition is statisfied (*condition* evaluates to TRUE).

condition is an expression evaluating to BOOLEAN.

EVALUATION

During execution.

WARNING

This statement can only be used within algorithmic code.

SEE ALSO

IF - FOR

EXAMPLE

```
/DECLARE/ INTEGER i, n, cum ;
          REAL t (10) ;
          QUEUE q1, q2 ;
          BOOLEAN b ;
          ...

/EXEC/ BEGIN
         ...
         WHILE i > n DO
         BEGIN
           PRINT (i) ;
           i := i - 1 ;
         END ;
       END ;

/STATION/ NAME = q1 ;
          SERVICE = BEGIN
                      ...
                      WHILE b DO CST (1) ;
                    END ;
```

WITH

NAME

`WITH` - Implicit access to the attributes of an object statement.

SYNTAX

`WITH` *object-id* `DO` *statement-list*

DESCRIPTION

This statement allows to execute a statement list with an implicit access to the attributes of an object specified by *object-id*. The main purpose of such a statement is to ease programming.

object-id is an identifier of a variable evaluating to an object type.

EVALUATION

During execution.

NOTES

This statement can only be used within algorithmic code.

EXAMPLE

```
/DECLARE/ OBJECT message;
            INTEGER length;
            STRING typmess;
          END;
          message mess1;

/EXEC/ BEGIN
         WITH mess1 DO
           BEGIN
             length  := GET(INTEGER) ;
             typmess := GET(STRING) ;
           END;
       END;
```

This code is equivalent to:

```
/DECLARE/ OBJECT message;
            INTEGER length;
            STRING typmess;
          END;
          message mess1;

/EXEC/ BEGIN
 mess1.length  := GET(INTEGER) ;
         mess1.typmess := GET(STRING) ;
       END;
```

WITH

NAME

WITH - Item selection within a list operator.

SYNTAX

object-list WITH *condition*

DESCRIPTION

This operator allows the selection of objects within a list. *condition* specifies the way to select the objects in the list *object-list*, which can be obtained for instance using the ALL operator.

condition is an expression evaluating to BOOLEAN.

The WITH operator returns the list of relevant objects.

EVALUATION

During execution.

NOTES

- An implicit reference to the attributes of the type specified is available inside *condition*.
- The WITH operator does not apply to lists of scalar values (BOOLEAN, INTEGER, REAL and STRING).
- This statement can only be used within algorithmic code.

EXAMPLE

```
/DECLARE/  QUEUE INTEGER code ;
           REF QUEUE rq (10) ;
           FLAG f(10) ;
           REF FLAG rf ;
           ...

/CONTROL/  CLASS = ALL QUEUE WITH code = 1 ;
           ...

/EXEC/ BEGIN
         ...
         rq := (ALL QUEUE WITH MTHRUPUT (QUEUE) < 0.5) ;
         ...
         FOR rf := f(1 STEP 1 UNTIL 10) WITH STATE DO
            PRINT (rf) ;
       END ;
```

7.3 Functions and Procedures

7.3.1 Simulation Control

GETPROFILE	Simulation execution profile. Access to the number of calls and CPU time taken in user procedures or functions.
GETSIMUL	Returns information concerning the simulation : scheduler, state of the processes, customers or `TIMER` objects, progress of the replications (if they exist).
GETSTAT	Gives statistical results requested for a variable.
GETTRACE	Returns in a trace treatment procedure in simulation the nature and the parameters of the traced operation.
SETEXCEPT	Exception processing description.
SETPROFILE	Measurement of the CPU times used by QNAP2 procedures.
SETSTAT	Description of the searched statistical results on a variable.
SETTIMER	`TIMER` objects handling.
SETTRACE	Handling of the trace events in simulation and-or the associated generalized notions.

GETPROFILE

NAME

GETPROFILE - Access to the number of calls and CPU time taken in procedures or user functions after they have been measured by means of the SETPROFILE:*keyword* procedure.

SYNTAX

Is only used with a keyword. Refer to each keyword description.

DESCRIPTION

The available calls with keyword are:
GETPROFILE:PROCNB - GETPROFILE:ISMETERED - GETPROFILE:RESULTS:*keyword* -
GETPROFILE:CALLERS:*keyword* - GETPROFILE:CALLED:*keyword*

Refer to each corresponding description and to the descriptions of:
GETPROFILE:RESULTS - GETPROFILE:CALLERS - GETPROFILE:CALLED
for the list of corresponding keywords.

EVALUATION

When the algorithmic code is running and after metering has stopped by SETPROFILE:STOPMETER.

NOTES

The number of procedures and/or user functions measured is increased by a few fictional QNAP2 procedures intended to measure how much QNAP2 itself uses to handle a simulation operation for example, or as a result of the user calling up pre-declared functions that carry out certain simulation operations. These fictional QNAP2 procedures have the character $ before and after their names to avoid any conflict of name with user procedures. In accessing the metering results functions, they can be specified by their name in the form of a QNAP2 string. These procedures are as follows in the current version:

- $ANACOM$ represents the analysis of the QNAP2 command language between the beginning and end of metering (to carry out such metering, create a small /EXEC/ block to start metering, and end the /EXEC/ block without stopping the measurement, which will only be done in a later /EXEC/ block). In this way, you can measure the time taken to compile a model, for example, but this level of metering is also the most external possible and therefore gives the total CPU time measured.
- $DEFEXC$ represents the running of the code between entering and leaving an /EXEC/ block (unless limited to a shorter time following the starting then stopping of measurement).
- $ANYPRO$ represents any code sequence that is not precisely defined, such as a service described by a BEGIN ... END sequence without a special procedure call, or the running of a procedure that is not otherwise measured.
- $METHOD$ represents the solution of the model, mathematically or by simulation. Its local use of CPU time in simulation represents the time QNAP2 itself takes to handle the

simulator.

- $RUNTIM$ represents all the predeclared procedures which act as simulation primitives, such as CST, EXP, WAIT, P, V, etc.

WARNING

If these functions are called while measurements are still in progress, an error will generally result.

SEE ALSO

GETPROFILE:PROCNB - GETPROFILE:ISMETERED - GETPROFILE:RESULTS - GETPROFILE:CALLERS - GETPROFILE:CALLED - SETPROFILE

EXAMPLE

The global example shown here is an EDITPROF procedure that can be reused in any model which uses the metering system for numbers of calls and CPU time taken per procedure. This procedure has already been used for real measurements and it is possible to copy it as it is to handle a concrete case. We will reproduce parts as an example in the description of each GETPROFILE keyword so as to give a better illustration of the whole.

```
/DECLARE/
& EDIT THE CPU TIME TAKEN FOR EACH PROCEDURE CALLED...
PROCEDURE EDITPROF;
INTEGER  I, J, I1, I2, J1, J2;
REAL     R1;
STRING   S1, S2;
REF FILE @FILE;
BEGIN
  @FILE := NEW(FILE);
  FILASSIGN(@FILE, "profile.log");
  OPEN(@FILE, 3);

  I := GETPROFILE:PROCNB;

  WRITELN(@FILE, " ", STRREPEAT ("-",40));
  WRITELN(@FILE, " Simulation total time : ", GETCPUT," s");
  WRITELN(@FILE, I:6," procedures or functions have been tested. These are: ");

  IF (I >= 1) THEN
  BEGIN
    FOR J:= 1 STEP 1 UNTIL I DO
    BEGIN
      S1:= GETPROFILE:RESULTS:PROCNAME (J);
      I1:= GETPROFILE:CALLERS:PROCNB (S1);
```

```
      I2:= GETPROFILE:CALLED:PROCNB (S1);

      WRITELN(@FILE, "### ", S1:10,
                     " called ",      GETPROFILE:RESULTS:CALLNB(S1), " FOIS ",
                     " total CPU ",    GETPROFILE:RESULTS:TOTALCPU(S1),
                     " local CPU ",    GETPROFILE:RESULTS:LOCALCPU(S1));
      WRITELN(@FILE, " NB callers ", I1,
                     " NB called ",   I2);

      IF (I1 >= 1) THEN
      BEGIN
        WRITELN(@FILE, " Callers: ");
        FOR J1:= 1 STEP 1 UNTIL I1 DO
        BEGIN
          WRITELN(@FILE,"   ",GETPROFILE:CALLERS:PROCNAME (S1,J1)," ");
        END;
      END;

      IF (I2 >= 1) THEN
      BEGIN
        WRITELN(@FILE, " Results on called :");
        FOR J2:= 1 STEP 1 UNTIL I2 DO
        BEGIN
          S2:= GETPROFILE:CALLED:PROCNAME (S1,J2);
          WRITELN(@FILE, "        ", GETPROFILE:CALLED:PROCNAME (S1,J2),
                         " called ", GETPROFILE:RESULTS:XREFNB (S1,S2),
                         " times. Total CPU ", GETPROFILE:RESULTS:XREFCPU (S1,S2));
        END;
      END;
      WRITELN(@FILE, " ", STRREPEAT ("-",40));
    END;
  END;
  CLOSE(@FILE);
END;
```

An actual application for the metering of a simulation will then be done by:

```
SETPROFILE:METERALL;       & Ask for all the procedures of the
                           & model to be measured
SETPROFILE:STARTMETER;     & Start metering
SIMUL;                     & Start the simulation you
                           & intend to measure
SETPROFILE:STOPMETER;      & Stop metering before editing
```

```
EDITPROF;                   & Edition of measurements using the
                            & user procedure EDITPROF (see
                            & a GETPROFILE:keyword for an example
                            & of this kind of procedure).
```

Typical extracts of results edited by this procedure that we have got for a real case are:

```
 ---------------------------------------
 Simulation total time : 12.20 s
    11 procedures or functions have been tested. These are:
### $ANACOM$   called 1 times  total CPU 11.86 local CPU 0.1000E-01
 NB callers 1 NB called 1
 Callers:
   NIL
 Results on called :
        $DEFEXC$ called 3 times. Total CPU 11.85
 ---------------------------------------
### $DEFEXC$   called 3 times  total CPU 11.85 local CPU 0.
 NB callers 1 NB called 1
 Callers:
   $ANACOM$
 Results on called :
        $ANYPRO$ called 3 times. Total CPU 11.85
 ---------------------------------------
### $METHOD$   called 1 times  total CPU 11.85 local CPU 8.610
 NB callers 1 NB called 1
 Callers:
   $ANYPRO$
 Results on called :
        $ANYPRO$ called 20758 times. Total CPU 3.240
 ---------------------------------------
### $ANYPRO$   called 20761 times  total CPU 11.85 local CPU 1.410
 NB callers 2 NB called 4
 Callers:
   $DEFEXC$
   $METHOD$
 Results on called :
        $RUNTIM$ called 20758 times. Total CPU 1.680
        $METHOD$ called 1 times. Total CPU 11.85
        ONE called 146 times. Total CPU 0.1300
        TRUEFALS called 146 times. Total CPU 0.2000E-01
 ---------------------------------------
### $RUNTIM$   called 20959 times  total CPU 1.690 local CPU 1.690
```

```
 NB callers 5 NB called 0
 Callers:
   $ANYPRO$
   FIVE
   TWO
   THREE
   FOUR
 ----------------------------------------
### ONE        called 146 times  total CPU 0.1300 local CPU 0.5000E-01
 NB callers 1 NB called 5
 Callers:
   $ANYPRO$
 Results on called :
        TRUEFALS called 292 times. Total CPU 0.2000E-01
        TWO called 25 times. Total CPU 0.1000E-01
        FIVE called 53 times. Total CPU 0.
        THREE called 50 times. Total CPU 0.4000E-01
        FOUR called 18 times. Total CPU 0.1000E-01
 ----------------------------------------
### TRUEFALS   called 590 times  total CPU 0.6000E-01 local CPU 0.6000E-01
 NB callers 5 NB called 0
 Callers:
   ONE
   TWO
   $ANYPRO$
   THREE
   FOUR
 ----------------------------------------
### TWO        called 38 times  total CPU 0.1000E-01 local CPU 0.1000E-01
 NB callers 2 NB called 3
 Callers:
   ONE
   FOUR
 Results on called :
        TRUEFALS called 38 times. Total CPU 0.
        FIVE called 20 times. Total CPU 0.
        $RUNTIM$ called 38 times. Total CPU 0.
 ----------------------------------------
### FIVE       called 73 times  total CPU 0. local CPU 0.
 NB callers 2 NB called 1
 Callers:
   TWO
   ONE
 Results on called :
```

```
       $RUNTIM$ called 73 times. Total CPU 0.
 ---------------------------------------
### THREE      called 50 times  total CPU 0.4000E-01 local CPU 0.1000E-01
 NB callers 1 NB called 3
 Callers:
   ONE
 Results on called :
       TRUEFALS called 50 times. Total CPU 0.
       $RUNTIM$ called 50 times. Total CPU 0.1000E-01
       FOUR called 22 times. Total CPU 0.2000E-01
 ---------------------------------------
### FOUR       called 40 times  total CPU 0.3000E-01 local CPU 0.1000E-01
 NB callers 2 NB called 3
 Callers:
   THREE
   ONE
 Results on called :
       TRUEFALS called 64 times. Total CPU 0.2000E-01
       $RUNTIM$ called 40 times. Total CPU 0.
       TWO called 13 times. Total CPU 0.
 ---------------------------------------
      ...
```

GETPROFILE:CALLED

NAME

GETPROFILE:CALLED - Gives all the procedures or functions called by a given procedure or function.

SYNTAX

Can only be used with a keyword. See the description of each keyword.

DESCRIPTION

The available keyword calls are:
GETPROFILE:CALLED:PROCNB and GETPROFILE:CALLED:PROCNAME

They give the number (by GETPROFILE:CALLED:PROCNB) and all the names one by one (by GETPROFILE:CALLED:PROCNAME) of the procedures or functions that have been called by a given procedure.

EVALUATION

When the algorithmic code is running and after metering has stopped.

NOTES

Functions calls in the form:
GETPROFILE:CALLERS:*keyword* and GETPROFILE:CALLED:*keyword*
play absolutely identical roles, the first (GETPROFILE:CALLERS:*keyword*) to find all callers of a given procedure, the second (GETPROFILE:CALLED:*keyword*) to find all those called. The keywords are identical in both these cases.

SEE ALSO

GETPROFILE:CALLED:PROCNB - GETPROFILE:CALLED:PROCNAME

GETPROFILE:CALLED:PROCNAME

NAME

GETPROFILE:CALLED:PROCNAME - Returns one by one the names of the procedures or functions called by a given procedure or function.

SYNTAX

```
string:= GETPROFILE:CALLED:PROCNAME (procedure,integer);
```

where the procedure used as an argument can be specified by a procedure or function pointer or a character string which represents the name (especially for fictional procedures intended for the measuring of QNAP2 self-consumption).

DESCRIPTION

This function supplies one by one (depending on the values of the integer used in a second argument) the names of all the procedures or functions (as long as they were involved in the measurements) that were called by the given procedure or function and used in the first argument.

The second argument is an integer which must lie between 1 and the total number of called in the given procedure or function.
This number is usually supplied by calling GETPROFILE:CALLED:PROCNB. A sequence of calls thus enables all the called in the given procedure or function to be handled one by one.

EVALUATION

When the algorithmic code is running and after metering has stopped.

NOTE

This function helps (jointly with the GETPROFILE:CALLED:PROCNB function) with the automatic building of tables of cross-references in the direction of the movement from callers to called.

WARNING

Specifying an invalid procedure as an argument leads to an error, as does specifying a second argument which is negative or zero or greater than the number of called in the procedure or function used in the first argument.

SEE ALSO

GETPROFILE:CALLED:PROCNB - GETPROFILE:CALLERS:PROCNB -
GETPROFILE:CALLERS:PROCNAME - GETPROFILE - GETPROFILE:RESULTS

EXAMPLE

Refer to the example of the EDITPROF procedure described with regard to GETPROFILE.

GETPROFILE:CALLED:PROCNB

NAME

GETPROFILE:CALLED:PROCNB - Returns the number of procedures or functions called by a given procedure or function.

SYNTAX

```
integer:= GETPROFILE:CALLED:PROCNB (procedure);
```

where the procedure used in an argument can be specified by a procedure or function pointer or a character string which represents the name (especially for the fictional procedures intended for the metering of QNAP2 self-consumption).

DESCRIPTION

The result of the function is an integer equal to the number of different user procedures or functions (or fictional procedures intended for the metering of QNAP2 self-consumption) which were called by the procedure or function used as an argument.

EVALUATION

When the algorithmic code is running and after metering has stopped.

NOTE

This function helps (jointly with the GETPROFILE:CALLED:PROCNAME function) with the automatic building of tables of cross-references in the direction of the movement from callers to called.

WARNING

Specifying an invalid procedure as an argument leads to an error.

SEE ALSO

GETPROFILE:CALLED:PROCNAME - GETPROFILE:CALLERS:PROCNB - GETPROFILE:CALLERS:PROCNAME - GETPROFILE - GETPROFILE:RESULTS

EXAMPLE

Refer to the example of the EDITPROF procedure described with regard to GETPROFILE.

NAME

GETPROFILE:CALLERS - Gives all the callers of a given procedure or function.

SYNTAX

Can only be used with a keyword. See the description of each keyword.

DESCRIPTION

The available keyword calls are:
GETPROFILE:CALLERS:PROCNB and GETPROFILE:CALLERS:PROCNAME

They are used to discover the number (by GETPROFILE:CALLERS:PROCNB) and all the names one by one (by GETPROFILE:CALLERS:PROCNAME) of the procedures or functions which have called a given procedure.

EVALUATION

When the algorithmic code is running and after metering has stopped.

NOTE

Calls for functions in the form:
GETPROFILE:CALLERS:*keyword* and GETPROFILE:CALLED:*keyword*
play absolutely identical roles, the first (GETPROFILE:CALLERS:*keyword*) to find all the callers of a given procedure, the second (GETPROFILE:CALLED:*keyword*) to find all the called. The keywords are identical in both these cases.

SEE ALSO

GETPROFILE:CALLERS:PROCNB - GETPROFILE:CALLERS:PROCNAME

GETPROFILE:CALLERS:PROCNAME

NAME

GETPROFILE:CALLERS:PROCNAME - Returns one by one the names of callers of a given procedure or function.

SYNTAX

```
string:= GETPROFILE:CALLERS:PROCNAME (procedure,integer);
```

where the procedure used as an argument can be specified by a procedure or function pointer or a character string which represents the name (especially for the fictional procedures intended for the metering of QNAP2 self-consumption).

DESCRIPTION

This function supplies one by one (depending on the values of the integer used in a second argument) the names of all the procedures or functions (as long as they were involved in the measurements) that were called by the given procedure or function and used in the first argument.

The second argument is an integer which must lie between 1 and the total number of callers of the given procedure or function.
This number is usually supplied by calling GETPROFILE:CALLERS:PROCNB. A sequence of calls thus enables all the called in the given procedure or function to be handled one by one.

EVALUATION

When the algorithmic code is running and after metering has stopped.

NOTE

This function helps (jointly with the GETPROFILE:CALLERS:PROCNAME function) with the automatic building of tables of cross-references in the direction of the movement from called to callers.

WARNING

Specifying an invalid procedure as an argument leads to an error, as does specifying a second argument which is negative or zero or greater than the number of callers to the procedure or function used in the first argument.

SEE ALSO

GETPROFILE:CALLERS:PROCNB - GETPROFILE:CALLED:PROCNB -
GETPROFILE:CALLED:PROCNAME - GETPROFILE - GETPROFILE:RESULTS

EXAMPLE

Refer to the example of the EDITPROF procedure described with regard to GETPROFILE.

NAME

GETPROFILE:CALLERS:PROCNB - Returns the number of procedures or functions which have called a given procedure or function.

SYNTAX

```
integer:= GETPROFILE:CALLERS:PROCNB (procedure);
```

where the procedure used as an argument can be specified by a procedure or function pointer or a character string which represents the name (especially for the fictional procedures intended for the metering of QNAP2 self-consumption).

DESCRIPTION

The result of the function is an integer equal to the number of different user procedures or functions (or fictional procedures intended for the metering of QNAP2 self-consumption) which have called the procedure or function used as an argument.

EVALUATION

When the algorithmic code is running and after metering has stopped.

NOTE

This function helps (jointly with the GETPROFILE:CALLERS:PROCNAME function) with the automatic building of tables of cross-references in the direction of the movement from called to callers.

WARNING

Specifying an invalid procedure as an argument leads to an error.

SEE ALSO

GETPROFILE:CALLERS:PROCNAME - GETPROFILE:CALLED:PROCNB - GETPROFILE:CALLED:PROCNAME - GETPROFILE - GETPROFILE:RESULTS

EXAMPLE

Refer to the example of the EDITPROF procedure described with regard to GETPROFILE.

GETPROFILE:ISMETERED

NAME

GETPROFILE:ISMETERED - Tests whether a user procedure or function is being metered or not.

SYNTAX

```
boolean := GETPROFILE:ISMETERED (procedure);
```

or else :

```
IF (GETPROFILE:ISMETERED (procedure)) THEN BEGIN ...  END;
```

The procedure given as an argument may also be specified by a pointer or its name given in the form of a string.

DESCRIPTION

Is used to test whether a user procedure or function is being measured, and thus prevents errors being made as a result of an unavailable result being requested.

EVALUATION

When the algorithmic code is running.

WARNING

A running error occurs if the argument passed to this function represents neither a procedure nor a user function, nor a fictional QNAP2 procedure intended to enable QNAP2 to access its own consumption. These errors will occur in particular if the argument is a pointer to a procedure with a NIL value, or else a character string whose content does not represent a procedure or function name.

SEE ALSO

GETPROFILE:PROCNB - GETPROFILE:RESULTS - GETPROFILE:CALLERS - GETPROFILE:CALLED - GETPROFILE

EXAMPLE

```
/DECLARE/ PROCEDURE FOO;
   ...
/EXEC / BEGIN
SETPROFILE:CLEAR;   & Removes all prior metering requests
   ...
IF (NOT GETPROFILE:ISMETERED (FOO)) THEN
  SETPROFILE:METERPROC (FOO);
   ...
END;
```

GETPROFILE:PROCNB

NAME

GETPROFILE:PROCNB - Access to the total number of procedures for which the numbers of calls and CPU time taken have been measured.

SYNTAX

```
integer := GETPROFILE:PROCNB;
```

DESCRIPTION

Returns in the form of an integer the number of procedures and/or user functions on which measurements of numbers of calls and measurements of CPU time have been made. The result of this function can be used among other things to find out the number of times a results editing loop for all procedures and/or functions must be done.

EVALUATION

When the algorithmic code is running.

NOTE

The number of procedures and/or user functions measured is increased by the special fictional QNAP2 procedures (see GETPROFILE).

SEE ALSO

GETPROFILE - GETPROFILE:ISMETERED - GETPROFILE:RESULTS - GETPROFILE:CALLERS - GETPROFILE:CALLED

EXAMPLE

This example is taken from the EDITPROF procedure described for GETPROFILE :

```
/DECLARE/
& EDITION CPU TIME TAKEN BY EACH PROCEDURE CALLED...
PROCEDURE EDITPROF;
       ...
BEGIN
  I:= GETPROFILE:PROCNB;

  WRITELN(@FILE, " ", STRREPEAT ("-",40));
  WRITELN(@FILE, " Simulation total time : ", GETCPUT," s");
  WRITELN(@FILE, I:6," procedures or functions have been tested. These are: ");
       ...
&    acquisition and edition of the results for each procedure
       ...
  END;
END;
```

NAME

GETPROFILE:RESULTS - Actual access to the measurements made on the procedures and/or user functions.

SYNTAX

Can only be used as a sub-keyword. The sub-keyword calls available are:
GETPROFILE:RESULTS:PROCNAME - GETPROFILE:RESULTS:CALLNB -
GETPROFILE:RESULTS:TOTALCPU - GETPROFILE:RESULTS:LOCALCPU -
GETPROFILE:RESULTS:XREFNB - GETPROFILE:RESULTS:XREFCPU

DESCRIPTION

Access to the number of calls of a procedure or user function, or their use of CPU time, local and global and within each procedure called; number of calls made by the procedure to another procedure.

EVALUATION

When the algorithmic code is running and after metering has stopped by:
SETPROFILE:STOPMETER;

WARNING

A running error occurs when one of these functions is called while metering is in progress.

SEE ALSO

GETPROFILE:PROCNB - GETPROFILE:ISMETERED - GETPROFILE:RESULTS:PROCNAME -
GETPROFILE:RESULTS:CALLNB - GETPROFILE:RESULTS:TOTALCPU -
GETPROFILE:RESULTS:LOCALCPU - GETPROFILE:RESULTS:XREFNB -
GETPROFILE:RESULTS:XREFCPU - GETPROFILE:CALLERS -
GETPROFILE:CALLED - GETPROFILE

GETPROFILE:RESULTS:CALLNB

NAME

GETPROFILE:RESULTS:CALLNB - Returns the total number of calls to a given procedure or function.

SYNTAX

```
integer:= GETPROFILE:RESULTS:CALLNB (procedure);
```

where the procedure used as an argument can be specified by a procedure or function pointer or a character string which represents the name (especially for the fictional procedures intended for the measurement of QNAP2 self-consumption).

DESCRIPTION

This function returns in integer form the total number of times that a procedure or function has been called between the start and end of measurement. The counter goes up by 1 at each entry.

EVALUATION

When the algorithmic code is running and after metering has stopped.

WARNING

Counting only happens during the period of time when metering is going on.

Specifying an invalid procedure as an argument results in an error.

SEE ALSO

GETPROFILE - GETPROFILE:RESULTS:PROCNAME - GETPROFILE:RESULTS:TOTALCPU - GETPROFILE:RESULTS:LOCALCPU - GETPROFILE:RESULTS:XREFNB - GETPROFILE:RESULTS:XREFCPU - GETPROFILE:CALLERS - GETPROFILE:CALLED

EXAMPLE

```
PROCEDURE EDITPROF;
INTEGER I,J,I1;
STRING S1;
BEGIN
  I:= GETPROFILE:PROCNB;
...
      WRITELN(@FILE, "### ", S1:10,
                     " called ",       GETPROFILE:RESULTS:CALLNB(S1), " times ",
                     " total CPU ",    GETPROFILE:RESULTS:TOTALCPU(S1),
                     " local CPU ",    GETPROFILE:RESULTS:LOCALCPU(S1),
...
  END;
```

```
END;
```

GETPROFILE:RESULTS:LOCALCPU

NAME

GETPROFILE:RESULTS:LOCALCPU - Returns the CPU time taken within the body of a procedure or function.

SYNTAX

```
real:= GETPROFILE:RESULTS:LOCALCPU (procedure);
```

where the procedure used as an argument can be specified by a procedure or function pointer or a character string which represents the name (especially for the fictional procedures intended for the measurement of QNAP2 self-consumption).

DESCRIPTION

Returns in real number form and expressed in seconds the CPU time taken within the body of a procedure or a function, excluding the CPU time taken by any procedures or user functions that may be called, as well as that of the QNAP2 self-consumption counted by means of fictional procedures. However, this function can be used to access QNAP2 self-consumption if it is called using this type of fictional procedure as an argument.

EVALUATION

When the algorithmic code is running and after metering has stopped.

NOTE

QNAP2 self-consumption for handling the schedule and the different basic base operations is obtained by calling:

```
GETPROFILE:RESULTS:LOCALCPU("$METHOD$")
```

QNAP2 self-consumption for service time or simulation synchronisation procedures is obtained by:

```
GETPROFILE:RESULTS:LOCALCPU("$RUNTIM$")
```

WARNING

The CPU time taken by a procedure or function is only counted during the period of time when metering is going on.

Specifying an invalid procedure as an argument results in an error.

SEE ALSO

GETPROFILE - GETPROFILE:RESULTS:PROCNAME - GETPROFILE:RESULTS:CALLNB - GETPROFILE:RESULTS:TOTALCPU - GETPROFILE:RESULTS:XREFNB - GETPROFILE:RESULTS:XREFCPU - GETPROFILE:CALLERS - GETPROFILE:CALLED

EXAMPLE

```
PROCEDURE EDITPROF;
INTEGER I,J;
REAL X;
STRING S1;
BEGIN
  I:= GETPROFILE:PROCNB;
...
      WRITELN(@FILE, "### ", S1:10,
                     " called ",       GETPROFILE:RESULTS:CALLNB(S1), " times ",
                     " total CPU ",    GETPROFILE:RESULTS:TOTALCPU(S1),
                     " local CPU ",    GETPROFILE:RESULTS:LOCALCPU(S1),
...
  END;
END;
```

GETPROFILE:RESULTS:PROCNAME

NAME

GETPROFILE:RESULTS:PROCNAME - Automatic acquisition of the names of the procedures and/or user functions measured.

SYNTAX

```
string:= GETPROFILE:RESULTS:PROCNAME(integer);
```

DESCRIPTION

The integer argument for calling this function represents the position number of the procedure or user function (or fictional QNAP2) measured. This position number may vary from 1 to the total number of procedures measured, a total number obtained by calling the GETPROFILE:PROCNB function (see this function). The result returned by calling the GETPROFILE:RESULTS:PROCNAME (I) function is a QNAP2 character string which represents the name of the Ith procedure or user function (or fictional QNAP2 in which case the string returned starts and finishes with dollars) measured. Repeated calls can be used to get all the procedures measured. The names obtained as character strings in this way can be used as arguments for the other access to get automatically the whole set of metered results on a model (see the example of the EDITPROF procedure described for GETPROFILE:*keyword*).

EVALUATION

When the algorithmic code is running and after metering has stopped.

WARNING

An error occurs if the integer call argument is negative or zero or greater than the total number of procedures and/or functions metered.

SEE ALSO

GETPROFILE - GETPROFILE:PROCNB - GETPROFILE:ISMETERED - GETPROFILE:CALLNB - GETPROFILE:RESULTS:TOTALCPU - GETPROFILE:RESULTS:LOCALCPU - GETPROFILE:RESULTS:XREFNB - GETPROFILE:RESULTS:XREFCPU - GETPROFILE:CALLERS - GETPROFILE:CALLED

EXAMPLE

```
/DECLARE/ PROCEDURE EDITPROF;
...
  WRITELN(@FILE, " ", STRREPEAT ("-",40));
  WRITELN(@FILE, " Simulation total time: ", GETCPUT," s");
  WRITELN(@FILE, I:6, " procedures ou fonctions have been tested. These are: ");

  IF (I >= 1) THEN
  BEGIN
    FOR J:= 1 STEP 1 UNTIL I DO
    BEGIN
      S1:= GETPROFILE:RESULTS:PROCNAME (J);
      I1:= GETPROFILE:CALLERS:PROCNB (S1);
      I2:= GETPROFILE:CALLED:PROCNB (S1);
...
END;
```

GETPROFILE:RESULTS:TOTALCPU

NAME

GETPROFILE:RESULTS:TOTALCPU - Returns the total CPU time taken for a given procedure or function.

SYNTAX

```
real:= GETPROFILE:RESULTS:TOTALCPU (procedure);
```

where the procedure used as an argument can be specified by a procedure or function pointer or a character string which represents the name (especially for the fictional procedures intended for the measurement of QNAP2 self-consumption).

DESCRIPTION

Returns in real number form and expressed in seconds the total CPU time taken by a procedure or function, including the time spent within the body of the procedure or function itself, and the time spent within any procedures or functions that may be called. However, if the procedure or function is running during a simulation by a customer who hands over control due to a period of service or a synchronisation, the CPU time during which the customer handed over control is not counted for the procedure or function in progress, unless of course this is run in parallel by other customers who are sharing.

EVALUATION

When the algorithmic code is run and after metering has stopped.

WARNING

Specifying an invalid procedure as an argument produces an error.

SEE ALSO

GETPROFILE - GETPROFILE:RESULTS:PROCNAME - GETPROFILE:RESULTS:CALLNB - GETPROFILE:RESULTS:LOCALCPU - GETPROFILE:RESULTS:XREFNB - GETPROFILE:RESULTS:XREFCPU - GETPROFILE:CALLERS - GETPROFILE:CALLED

EXAMPLE

```
PROCEDURE EDITPROF;
...
      WRITELN(@FILE, "### ", S1:10,
                     " called ",      GETPROFILE:RESULTS:CALLNB(S1), " times ",
                     " total CPU ",    GETPROFILE:RESULTS:TOTALCPU(S1),
                     " local CPU ",    GETPROFILE:RESULTS:LOCALCPU(S1),
...
  END;
END;
```

GETPROFILE:RESULTS:XREFCPU

NAME

GETPROFILE:RESULTS:XREFCPU - Returns the CPU time used by a procedure or function as called by another procedure or function.

SYNTAX

```
real:= GETPROFILE:RESULTS:XREFCPU (procedure1,procedure2);
```

where the procedures used as arguments can be specified by procedure or function pointers or QNAP2 character strings representing names (especially for fictional procedures intended for the measurement of QNAP2 self-consumption).

DESCRIPTION

Returns as a real number the CPU time used by the first procedure or function used as an argument within the second procedure or function used as an argument, assuming that this second procedure or function is called. It is thus possible to get the breakdown of CPU time used for a procedure or function between its own part and its called parts (for which the CPU time used may nevertheless include a part of the time used locally by the first procedure used as an argument, for example for a crossed recursive call).

EVALUATION

When the algorithmic code is running and after metering has stopped.

NOTE

Use this function together with:
GETPROFILE:CALLERS:PROCNB GETPROFILE:CALLERS:PROCNAME GETPROFILE:CALLED:PROCNB
to edit entirely automatically all the CPU time used by paired procedures (caller, called).

WARNING

Specifying an invalid procedure as an argument produces an error.

SEE ALSO

GETPROFILE - GETPROFILE:RESULTS:PROCNAME - GETPROFILE:RESULTS:CALLNB - GETPROFILE:RESULTS:TOTALCPU - GETPROFILE:RESULTS:LOCALCPU - GETPROFILE:RESULTS:XREFNB - GETPROFILE:CALLERS - GETPROFILE:CALLED

EXAMPLE

Refer to the example of the EDITPROF procedure described for GETPROFILE.

NAME

GETPROFILE:RESULTS:XREFNB - Returns the number of times where a given procedure or function has called another given procedure or function.

SYNTAX

```
integer:= GETPROFILE:RESULTS:XREFNB (procedure1, procedure2);
```

where the procedure used as an argument can be specified by a procedure or function pointer or a character string which represents the name (especially for the fictional procedures intended for the measurement of QNAP2 self-consumption).

DESCRIPTION

Returns as an integer the number of times that the first procedure or function used as an argument has called the second procedure or function used as an argument, i.e. returns the number of cross references observed during the metering period.

EVALUATION

When the algorithmic code is running and after metering has stopped.

NOTE

Use this function together with:
GETPROFILE:CALLERS:PROCNB GETPROFILE:CALLERS:PROCNAME GETPROFILE:CALLED:PROCNB
and GETPROFILE:CALLED:PROCNAME
to edit entirely automatically all the cross references made by a model.

WARNING

Specifying an invalid procedure as an argument produces an error.

SEE ALSO

GETPROFILE - GETPROFILE:RESULTS:PROCNAME - GETPROFILE:RESULTS:CALLNB - GETPROFILE:RESULTS:TOTALCPU - GETPROFILE:RESULTS:LOCALCPU - GETPROFILE:RESULTS:XREFCPU - GETPROFILE:CALLERS - GETPROFILE:CALLED

EXAMPLE

Refer to the example of the EDITPROF procedure described for GETPROFILE.

GETSIMUL

NAME

`GETSIMUL` - Returns information concerning the simulation : scheduler, state of the processes, customers or `TIMER` objects, progress of the replications (if they exist).

SYNTAX

`GESTSIMUL:key-word`

DESCRIPTION

The available key-words are :

GETSIMUL:CURREPLI	Returns the number of the current replication.
GETSIMUL:FIRSTPROC	Returns the first process (customer or timer) at the top of the scheduler.
GETSIMUL:NEXTPROC	Returns the process (customer or timer) following a given process.
GETSIMUL:PRSTATUS	Returns the state of a process (customer or timer).
GETSIMUL:REPLINB	Returns the total number of replications forecasted for the current simulation.
GETSIMUL:WAKETIME	Returns the waking time of a process (customer or timer).

EVALUATION

During the execution.

NAME

GETSIMUL:CURREPLI - Returns the number of the current replication in simulation.

SYNTAX

```
GETSIMUL:CURREPLI ;
```

DESCRIPTION

During a simulation, returns an integer which represents the number of the current replication in case of REPLICATION method use.

Returns 1 (default value) in the other cases.

EVALUATION

During the execution.

NOTES

No error is produced if the REPLICATION method is not used, but the returned value is 1.

If this function is called after the simulation, the returned value is the total number of replications.

If the function is called during the treatment of a SIMSTART exception, the number of the replication returned is the number of the beginning replication. In case of handling the SIMSTOP exception, the number returned is relative to the last replication.

SEE ALSO

GETSIMUL:REPLINB - ESTIMATION

EXAMPLE

```
 /STATION/ NAME = ...
           SERVICE =
 BEGIN
   PRINT ("THE NUMBER OF THE CURRENT REPLICATION IS",  GETSIMUL:CURREPLI) ;
 END;

/CONTROL/ ESTIM = REPLICATION (4);
```

GETSIMUL:FIRSTPROC

NAME

GETSIMUL:FIRSTPROC - Returns the first process (customer or timer) at the top of the scheduler.

SYNTAX

GETSIMUL:FIRSTPROC ;

DESCRIPTION

Returns (using a REF ANY) the current process, customer or timer, at the top of the scheduler, handled by the simulator (or waiting to take it immediately if the current process handles an exception).

EVALUATION

During the execution, in a simulation (or immediately after a simulation).

NOTES

The use of this function and the GETSIMUL:NEXTPROC and GETSIMUL:WAKETIME functions, allows the complete reading of the scheduler (for instance, for debugging).

SEE ALSO

GETSIMUL:NEXTPROC - GETSIMUL:WAKETIME

EXAMPLE

```
 /STATION/ NAME =q;
           SERVICE =
BEGIN
  IF ( GETSIMUL.FIRSTPROC::CUSTOMER<>CUSTOMER) THEN
& the process at the top of the scheduler must be the current customer
     PRINT ("BUG") ;
END;
```

NAME

GETSIMUL:PRSTATUS - Returns the state of a process (customer or timer).

SYNTAX

GETSIMUL:PRSTATUS (*rany*);

DESCRIPTION

Returns (using an integer) the state of the process (customer or timer) given in argument. The possible returned values are :

- 0 : Sleeping process (customer without server or sleeping timer)
- 1 : Waiting process (waiting until the waking time)
- 2 : Active current process
- 3 : Customer blocked on a synchronization operation

rany is an expression of REF ANY type.

EVALUATION

During the execution in a simulation (or immediately after a simulation).

WARNING

An error occurs if the REF ANY argument does not point a customer or a timer (or a subtype of customer or timer).

SEE ALSO

GETSIMUL:FIRSTPROC - GETSIMUL:NEXTPROC - GETSIMUL:WAKETIME - TIMER

EXAMPLE

```
 /STATION/ NAME = q;
           SERVICE =
BEGIN
  IF (GETSIMUL:PRSTATUS (CUSTOMER) <> 2)
&  In a service, the current customer is the active current process
    THEN  PRINT ("BUG");
END ;
```

GETSIMUL:REPLINB

NAME

GETSIMUL:REPLINB - Returns the total number of replications requested for a simulation.

SYNTAX

```
GETSIMUL:REPLINB ;
```

DESCRIPTION

Returns (using an integer) the total number of replications requested using the ESTIMATION = REPLICATION option of the CONTROL command.

Returns 1 in the other cases.

EVALUATION

During the execution.

NOTES

The function does not perform an error if the REPLICATION method is not used, but returns 1.

SEE ALSO

GETSIMUL:CURREPLI - ESTIMATION

EXAMPLE

```
 /STATION/ NAME = q;
           SERVICE =
BEGIN
   PRINT ("THE SIMULATION WILL NOT RUN AFTER", GETSIMUL:REPLINB,
"REPLICATIONS");
END ;

/CONTROL/ ESTIM=REPLICATION (4);
```

NAME

GETSIMUL:WAKETIME - Returns the waking time of a process (customer or timer).

SYNTAX

GETSIMUL:WAKETIME (*rany*);

DESCRIPTION

Returns the waking time of the process (customer or timer) given in argument (according to the service time for a customer and the future waking time for a timer).

This time is returned in the form of a real.

If the process is not in the scheduler (blocked customer, customer without server, sleeping timer), this function returns -1.0.

rany is an expression of REF ANY type.

EVALUATION

During the execution in a simulation (or immediately after a simulation).

NOTES

With this function you may test if a process is waiting (returned value greater or equal than 0 or not). However, the GETSIMUL:PRSTATUS function is better for this use.

WARNING

An error occurs if the argument given using a REF ANY does not point a customer or a timer (or a subtype of customer or timer).

SEE ALSO

GETSIMUL:FIRSTPROC - GETSIMUL:NEXTPROC - GETSIMUL:PRSTATUS - TIMER

EXAMPLE

```
/DECLARE/ REF ANY @proc;
......
/EXEC/ BEGIN
         @proc:= GETSIMUL:FIRSTPROC;
         WHILE (@proc <> TSYSWDOG ) DO
           BEGIN
             PRINT ("process: ",@proc," Waking time",
             GETSIMUL:WAKETIME(@proc));
             @proc:=GETSIMUL:NEXTPROC(@proc);
           END;
       END;
```

NAME

`GETSTAT` - Gives statistical results requested for a variable.

SYNTAX

`GETSTAT:key-word`

DESCRIPTION

The available key-words are :

GETSTAT:ACCURACY	Returns the accuracy on the mean of a statistical variable.
GETSTAT:BLOCKED:MEAN	Returns the mean blocked time of a queue.
GETSTAT:BUSYPCT:MEAN	Returns the mean busy rate of a station.
GETSTAT:CORRELATION	Returns the auto-correlation coefficients of a statistical variable.
GETSTAT:CUSTNB:MEAN	Returns the mean number of customers in the queue.
GETSTAT:MARGINAL	Returns the probability for a variable to be in an interval.
GETSTAT:MAXIMUM	Returns the maximum value of a statistical variable.
GETSTAT:MEAN	Returns the mean of a statistical variable.
GETSTAT:MINIMUM	Returns the minimum value of a statistical variable.
GETSTAT:RESPONSE:MEAN	Returns the mean response time of the queue.
GETSTAT:SAMPSIZE	Returns the number of measures of a discrete sample.
GETSTAT:SAMPTIME	Returns the total sampling duration of a continuous sample.
GETSTAT:SERVICE:MEAN	Returns the mean service time of a queue.
GETSTAT:THRUPUT:MEAN	Returns the mean throughput of a queue.
GETSTAT:VARIANCE	Returns the variance of a statistical variable.

NOTES

Statistical results are only available when the SETSTAT procedure has been used on the variable before the beginning of the simulation.

GETSTAT:ACCURACY

NAME

GETSTAT:ACCURACY - Computes and returns the accuracy on the mean of a statistical variable.

SYNTAX

GETSTAT:ACCURACY (*variable*) ;

DESCRIPTION

Computes the accuracy on the mean of the values taken by a statistical variable.

The variable must be WATCHED declared and its nature, discrete or continuous, specified.

The accuracy on the mean of the standard results of a queue or a (queue, class) couple are given by the following functions:

- GETSTAT:SERVICE:ACCURACY for the service time.
- GETSTAT:RESPONSE:ACCURACY for the response time.
- GETSTAT:BLOCKED:ACCURACY for the blocking time.
- GETSTAT:BUSYPCT:ACCURACY for the busy rate.
- GETSTAT:CUSTNB:ACCURACY for the customer number.
- GETSTAT:THRUPUT:ACCURACY for the throughput.

EVALUATION

During the execution.

WARNING

- In simulation, the request of statistical results must have be done explicitly.
- The method used to compute confidence intervals must be specified.

SEE ALSO

SETSTAT:DISCRETE - SETSTAT:CONTINUE - SETSTAT:QUEUE - SETSTAT:CLASS - SETSTAT:ACCURACY

EXAMPLE

```
 /DECLARE/ WATCHED INTEGER I;
           REAL;
           PROCEDURE init (j);
             VAR WATCHED INTEGER j;
             BEGIN
               SETSTAT:DISCRETE (j);
               SETSTAT:PARTIAL (j);
               SETSTAT:ACCURACY (j);
             END;

 /CONTROL/ TMAX = 10000;
           PERIOD = 2000;
           ESTIMATION = REGENERATION;
           TEST = BEGIN
                    SETSTAT:SAMPLE (I);
                    result := GETSTAT:MEAN (I);
                  END;

/EXEC/ BEGIN
         init (I);
         .....
         SIMUL;
         PRINT ( GETSTAT:ACCURACY (I));
         ....
      END;
```

GETSTAT:BLOCKED:MEAN

NAME

GETSTAT:BLOCKED:MEAN - Returns the mean blocking time in a queue.

SYNTAX

GETSTAT:BLOCKED:MEAN (*list-of-queues, list-of-classes*);

DESCRIPTION

Returns the mean blocking time in a queue.

The request of statistical calculation will have been specified by the SETSTAT:BLOCKED:MEAN procedure for the blocking time, SETSTAT:QUEUE to compute all the standard results on a queue and SETSTAT:CLASS to compute all the standard results for customers classes.

EVALUATION

During the execution.

SEE ALSO

SETSTAT:QUEUE - SETSTAT:CLASS - SETSTAT:PARTIAL - SETSTAT:ACCURACY - SETSTAT:MARGINAL - SETSTAT:CORRELAT - SETSTAT:PRECISION

EXAMPLE

```
/DECLARE/ QUEUE q;
          CLASS c1, c2;
          .....

/EXEC/ BEGIN
         SETSTAT:BLOCKED:MEAN (q);
         ....
         SIMUL;
         PRINT ( GETSTAT:BLOCKED:MEAN (q));
         ....
       END;
```

NAME

GETSTAT:BUSYPCT:MEAN - Returns the mean occupation rate of the queue servers.

SYNTAX

GETSTAT:BUSYPCT:MEAN (*list-of-queues, list-of-classes*);

DESCRIPTION

Returns the occupation rate of the queue servers.

The request of statistical calculation will have been specified by the SETSTAT:BUSYPCT:MEAN procedure for the occupation rate, SETSTAT:QUEUE to compute all the standard results on a queue and SETSTAT:CLASS to compute all the standard results for customers classes.

EVALUATION

During the execution.

SEE ALSO

SETSTAT:QUEUE - SETSTAT:CLASS - SETSTAT:PARTIAL - SETSTAT:ACCURACY - SETSTAT:MARGINAL - SETSTAT:CORRELAT - SETSTAT:PRECISION

EXAMPLE

```
/DECLARE/ QUEUE q;
          CLASS c1, c2;
          .....

/EXEC/ BEGIN
         SETSTAT:BUSYPCT:MEAN (q);
         ....
         SIMUL;
         PRINT ( GETSTAT:BUSYPCT:MEAN (q));
         ....
       END;
```

GETSTAT:CORRELATION

NAME

GETSTAT:CORRELATION - Computes and returns the auto-correlation coefficients of a statistical variable.

SYNTAX

GETSTAT:CORRELATION (*variable, order*) ;

DESCRIPTION

Computes and returns the auto-correlation coefficient with the specified order concerning a variable.

The variable must be WATCHED declared and its nature discrete or continuous specified.

The auto-correlation coefficients of the standard results of a queue or a (queue, class) couple are given by the following functions :

- GETSTAT:SERVICE:CORRELATION for the service time.
- GETSTAT:RESPONSE:CORRELATION for the response time.
- GETSTAT:BLOCKED:CORRELATION for the blocking time.
- GETSTAT:BUSYPCT:CORRELATION for the busy rate.
- GETSTAT:CUSTNB:CORRELATION for the customer number.

EVALUATION

During the execution.

WARNING

In simulation, the request of statistical results must have be done explicitly.

SEE ALSO

SETSTAT:DISCRETE - SETSTAT:CONTINUE - SETSTAT:QUEUE - SETSTAT:CLASS - SETSTAT:ACCURACY - SETSTAT:CORRELATION

EXAMPLE

```
 /DECLARE/ WATCHED INTEGER I;
           PROCEDURE init (j);
             VAR WATCHED INTEGER j;
             BEGIN
               SETSTAT:DISCRETE (j);
               SETSTAT:ACCURACY (j);
               SETSTAT:CORRELATION (j, 3);
             END;

/CONTROL/ TMAX = 10000;
          PERIOD = 2000;
          ESTIMATION = REGENERATION;
          TEST = BEGIN
                   SETSTAT:SAMPLE (I);
                 END;

/EXEC/ BEGIN
         init (I);
         .....
         SIMUL;
         PRINT ( GETSTAT:CORRELATION (I,3));
         ....
       END;
```

GETSTAT:CUSTNB:MEAN

NAME

GETSTAT:CUSTNB:MEAN - Returns the mean number of customers in a queue.

SYNTAX

GETSTAT:CUSTNB:MEAN (*list-of-queues, list-of-classes*);

DESCRIPTION

Returns the mean customer number in a queue.

The request of statistical calculation will have been specified by the SETSTAT:CUSTNB:MEAN procedure for the customer number, SETSTAT:QUEUE to compute all the standard results on a queue and SETSTAT:CLASS to compute all the standard results for customers classes.

EVALUATION

During the execution.

SEE ALSO

SETSTAT:QUEUE - SETSTAT:CLASS - SETSTAT:PARTIAL - SETSTAT:ACCURACY - SETSTAT:MARGINAL - SETSTAT:CORRELAT - SETSTAT:PRECISION

EXAMPLE

```
/DECLARE/ QUEUE q;
          CLASS c1, c2;
          .....

/EXEC/ BEGIN
         SETSTAT:CUSTNB:MEAN (q);
         ....
         SIMUL;
         PRINT ( GETSTAT:CUSTNB:MEAN (q));
         ....
       END;
```

NAME

GETSTAT:MARGINAL - Computes and returns the probability for a variable to be in an interval.

SYNTAX

GETSTAT:MARGINAL (*variable, num-int*) ;

DESCRIPTION

Computes and returns the probability for a variable to be in a given interval.

num-int : is an INTEGER which refers to the interval number.

The variable must be WATCHED declared which discrete or continuous nature would already have been specified.

Marginal probabilities on standard results of a queue or a (queue, class) couple are given by the following functions :

- GETSTAT:SERVICE:MARGINAL for the service time.
- GETSTAT:RESPONSE:MARGINAL for the response time.
- GETSTAT:BLOCKED:MARGINAL for the blocking time.
- GETSTAT:BUSYPCT:MARGINAL for the busy rate.
- GETSTAT:CUSTNB:MARGINAL for the customer number.

EVALUATION

During the execution.

WARNING

In simulation, the request of statistical results must have be done explicitly.

SEE ALSO

SETSTAT:DISCRETE - SETSTAT:CONTINUE - SETSTAT:QUEUE - SETSTAT:CLASS

EXAMPLE

```
/DECLARE/ WATCHED INTEGER I;
          INTEGER IND;
          PROCEDURE init (j);
            VAR WATCHED INTEGER j;
            BEGIN
              SETSTAT:DISCRETE (j);
              SETSTAT:MARGINAL (j, 5, 0., 1.);
            END;

/EXEC/ BEGIN
         init (I);
         ....
         PRINT (GETSTAT:MEAN (I));
         FOR IND : = 1 STEP 1 UNTIL 5 DO
           PRINT ( GETSTAT:MARGINAL (I, IND));
       END;
```

NAME

GETSTAT:MAXIMUM - Computes and returns the maximum value of a statistical variable.

SYNTAX

GETSTAT:MAXIMUM (*variable*) ;

DESCRIPTION

Computes the maximum value taken by a variable.

The variable must be WATCHED declared and its nature, discrete or continuous, specified.

Maximum values on standard results of a queue or a (queue, class) couple are given by the following procedures:

- GETSTAT:SERVICE:MAXIMUM for the service time.
- GETSTAT:RESPONSE:MAXIMUM for the response time.
- GETSTAT:BLOCKED:MAXIMUM for the blocking time.
- GETSTAT:BUSYPCT:MAXIMUM for the busy rate.
- GETSTAT:CUSTNB:MAXIMUM for the customer number.

EVALUATION

During the execution.

WARNING

In simulation, the request of statistical results must have be done explicitly.

SEE ALSO

SETSTAT:DISCRETE - SETSTAT:CONTINUE - SETSTAT:QUEUE - SETSTAT:CLASS

EXAMPLE

```
/DECLARE/ WATCHED INTEGER I;
          PROCEDURE init (j);
            VAR WATCHED INTEGER j;
            BEGIN
              SETSTAT:DISCRETE (j);
            END;

/EXEC/ BEGIN
         init (I);
         ....
         PRINT (GETSTAT:MAXIMUM (I));
       END;
```

GETSTAT:MEAN

NAME

GETSTAT:MEAN - Computes and returns the mean of a statistical variable.

SYNTAX

GETSTAT:MEAN (*variable*) ;

DESCRIPTION

Computes the mean of the values taken by a variable.

The variable must be WATCHED declared and its nature, discrete or continuous, specified.

If the computation has been requested, the standard results on a queue or a (*queue, class*) couple may be got using the functions :

- GETSTAT:SERVICE:MEAN for the service time.
- GETSTAT:RESPONSE:MEAN for the response time.
- GETSTAT:BLOCKED:MEAN for the blocking time.
- GETSTAT:BUSYPCT:MEAN for the busy rate.
- GETSTAT:CUSTNB:MEAN for the customer number.
- GETSTAT:THRUPUT:MEAN for the throughput.

EVALUATION

During the execution.

WARNING

In simulation, the request of statistical results must have be done explicitly.

SEE ALSO

SETSTAT:DISCRETE - SETSTAT:CONTINUE - SETSTAT:QUEUE - SETSTAT:CLASS

EXAMPLE

```
 /DECLARE/ WATCHED INTEGER I;
           PROCEDURE init (j);
             VAR WATCHED INTEGER j;
             BEGIN
               SETSTAT:DISCRETE (j);
             END;

/EXEC/ BEGIN
         init (I);
         ....
         PRINT ( GETSTAT:MEAN (I));
     END;
```

NAME

GETSTAT:MINIMUM - Computes and returns the minimum value of a statistical variable.

SYNTAX

GETSTAT:MINIMUM (*variable*) ;

DESCRIPTION

Computes the minimum value taken by a variable.

The variable must be WATCHED declared and its nature, discrete or continuous, specified.

As the computing has been requested, maximum values on standard results of a queue or a (queue, class) couple are given by the following functions:

- GETSTAT:SERVICE:MINIMUM for the service time.
- GETSTAT:RESPONSE:MINIMUM for the response time.
- GETSTAT:BLOCKED:MINIMUM for the blocking time.
- GETSTAT:BUSYPCT:MINIMUM for the busy rate.
- GETSTAT:CUSTNB:MINIMUM for the customer number.

EVALUATION

During the execution.

WARNING

In simulation, the request of statistical results must have be done explicitly.

SEE ALSO

SETSTAT:DISCRETE - SETSTAT:CONTINUE - SETSTAT:QUEUE - SETSTAT:CLASS

EXAMPLE

```
/DECLARE/ WATCHED INTEGER I;
          PROCEDURE init (j);
            VAR WATCHED INTEGER j;
            BEGIN
              SETSTAT:DISCRETE (j);
            END;

/EXEC/ BEGIN
         init (I);
         ....
         PRINT ( GETSTAT:MINIMUM (I));
       END;
```

GETSTAT:RESPONSE:MEAN

NAME

GETSTAT:RESPONSE:MEAN - Returns the mean reponse time of a queue.

SYNTAX

GETSTAT:RESPONSE:MEAN (*list-of-queues, list-of-classes*);

DESCRIPTION

Returns the mean response time of a queue.

The request of statistical calculation will have been specified by the SETSTAT:RESPONSE:MEAN procedure for the response time, SETSTAT:QUEUE to compute all the standard results on a queue and SETSTAT:CLASS to compute all the standard results for customers classes.

EVALUATION

During the execution.

SEE ALSO

SETSTAT:QUEUE - SETSTAT:CLASS - SETSTAT:PARTIAL - SETSTAT:ACCURACY - SETSTAT:MARGINAL - SETSTAT:CORRELAT - SETSTAT:PRECISION

EXAMPLE

```
/DECLARE/ QUEUE q;
          CLASS c1, c2;
          .....

/EXEC/ BEGIN
         SETSTAT:RESPONSE:MEAN (q);
         ....
         SIMUL;
         PRINT ( GETSTAT:RESPONSE:MEAN (q));
         ....
       END;
```

NAME

GETSTAT:SAMPSIZE - Computes and returns the total number of measures of a discrete sample.

SYNTAX

GETSTAT:SAMPSIZE (*variable*) ;

DESCRIPTION

GETSTAT:SAMPSIZE returns an INTEGER.
Returns the number of measures of a discrete sample. Returns an error if the sample is not discrete.

The variable must be declared as WATCHED; its nature must be discrete in this case.

EVALUATION

During the execution.

WARNING

In simulation, the request of statistical results must have be done explicitly.
An error occurs when invoked on a continuous sample. Use GETSTAT:SAMPTIME in this case.

SEE ALSO

SETSTAT:DISCRETE - SETSTAT:CONTINUE - GETSTAT:SAMPTIME

EXAMPLE

```
 /DECLARE/ WATCHED INTEGER I;
           PROCEDURE init (j);
             VAR WATCHED INTEGER j;
             BEGIN
               SETSTAT:DISCRETE (j);
             END;

/EXEC/ BEGIN
         init (I);
         ....
         PRINT ( GETSTAT:SAMPSIZE (I));
       END;
```

GETSTAT:SAMPTIME

NAME

GETSTAT:SAMPTIME - Computes and returns the total sampling time on a continuous sample.

SYNTAX

GETSTAT:SAMPTIME (*variable*) ;

DESCRIPTION

GETSTAT:SAMPTIME returns a REAL.
Returns the total sampling time on a continuous sample. Returns an error if the sample is not continuous.

The variable must be declared as WATCHED; its nature must be continuous in this case.

EVALUATION

During the execution.

WARNING

In simulation, the request of statistical results must have be done explicitly.
An error occurs when invoked on a discrete sample. Use GETSTAT:SAMPSIZE in this case.

SEE ALSO

SETSTAT:DISCRETE - SETSTAT:CONTINUE - GETSTAT:SAMPSIZE

EXAMPLE

```
 /DECLARE/ WATCHED INTEGER I;
           PROCEDURE init (j);
             VAR WATCHED INTEGER j;
             BEGIN
               SETSTAT:CONTINUE (j);
             END;

/EXEC/ BEGIN
         init (I);
         ....
         PRINT ( GETSTAT:SAMPTIME (I));
       END;
```

NAME

GETSTAT:SERVICE:MEAN - Computes and returns the mean service time of a queue.

SYNTAX

GETSTAT:SERVICE:MEAN (*list-of-queues, list-of-classes*);

DESCRIPTION

Computes the mean service time of a queue .

The request of statistical calculation will have been specified by the SETSTAT:SERVICE:MEAN procedure for the service time, SETSTAT:QUEUE to compute all the standard results on a queue and SETSTAT:CLASS to compute all the standard results for customers classes.

EVALUATION

During the execution.

SEE ALSO

SETSTAT:QUEUE - SETSTAT:CLASS - SETSTAT:PARTIAL - SETSTAT:ACCURACY - SETSTAT:MARGINAL - SETSTAT:CORRELAT - SETSTAT:PRECISION

EXAMPLE

```
/DECLARE/ QUEUE q;
          CLASS c1, c2;
          .....

/EXEC/ BEGIN
         SETSTAT:SERVICE:MEAN (q);
         ....
         SIMUL;
         PRINT ( GETSTAT:SERVICE:MEAN (q));
         ....
       END;
```

GETSTAT:THRUPUT:MEAN

NAME

GETSTAT:THRUPUT:MEAN - Returns the mean throughput of a queue.

SYNTAX

GETSTAT:THRUPUT:MEAN (*list-of-queues, list-of-classes*);

DESCRIPTION

Returns the mean throughput of a queue (REAL type).

The request of statistical calculation will have been specified by the SETSTAT:THRUPUT:MEAN procedure for the throughput, SETSTAT:QUEUE to compute all the standard results on a queue and SETSTAT:CLASS to compute all the standard results for customers classes.

EVALUATION

During the execution.

SEE ALSO

SETSTAT:QUEUE - SETSTAT:CLASS - SETSTAT:PARTIAL - SETSTAT:ACCURACY - SETSTAT:MARGINAL - SETSTAT:CORRELATION - SETSTAT:PRECISION

EXAMPLE

```
/DECLARE/ QUEUE q;
          CLASS c1, c2;
          ...

/EXEC/ BEGIN
         SETSTAT:THRUPUT:MEAN (q);
         ...
         SIMUL;
         PRINT ( GETSTAT:THRUPUT:MEAN (q));
         ...
       END;
```

NAME

GETSTAT:VARIANCE - Computes and returns the variance of a statistical variable.

SYNTAX

GETSTAT:VARIANCE (*variable*) ;

DESCRIPTION

Computes the variance of the values taken by a variable.

The variable must be WATCHED declared and its nature, discrete or continuous, specified.

The variance of the standard results of a queue or a (queue, class) couple are given by the following functions :

- GETSTAT:SERVICE:VARIANCE for the service time.
- GETSTAT:RESPONSE:VARIANCE for the response time.
- GETSTAT:BLOCKED:VARIANCE for the blocking time.
- GETSTAT:BUSYPCT:VARIANCE for the busy rate.
- GETSTAT:CUSTNB:VARIANCE for the customer number.

EVALUATION

During the execution.

WARNING

In simulation, the request of statistical results must have be done explicitly.

SEE ALSO

SETSTAT:DISCRETE - SETSTAT:CONTINUE - SETSTAT:QUEUE - SETSTAT:CLASS

EXAMPLE

```
/DECLARE/ WATCHED INTEGER I;
          PROCEDURE init (j);
            VAR WATCHED INTEGER j;
            BEGIN
              SETSTAT:DISCRETE (j);
            END;

/EXEC/ BEGIN
         init (I);
         ....
         PRINT ( GETSTAT:VARIANCE (I));
       END;
```

GETTRACE

NAME

`GETTRACE` - Returns in a trace treatment procedure the nature and the parameters of the traced operation.

SYNTAX

`GETTRACE:keyword`

DESCRIPTION

This function must be used with a keyword among the following ones:

GETTRACE:CCLASS	Returns the customers class passed as argument to the current traced operation.
GETTRACE:CLISTGET	Returns one by one the references to the customers (their number can be greater than 1) considered as partners of the current traced operation.
GETTRACE:CLLISTGET	Returns one by one the references to the classes of the requests (their number can be greater than 1) specified for the current traced operation.
GETTRACE:CODENAME	Converts the symbolic name of an event to the corresponding integer code.
GETTRACE:CPRIOR	Returns the priority of customers passed as argument of a simulation operation during a trace.
GETTRACE:CPROVOKE	Returns a reference on the customer who produced the traced operation.
GETTRACE:CSECONDR	Returns a reference on the customer, when existing, considered as simple partner of the traced operation.
GETTRACE:CSUBJECT	Returns a reference on the customer, when existing, who was subject of the traced operation.
GETTRACE:DELAY	Returns the value of a service time delay during a trace treatment.
GETTRACE:DISTRI	Returns during a trace treatment the distribution law of a service time that has just begun.
GETTRACE:EVCODE	Returns the current event type during a trace treatment.
GETTRACE:EVSTATUS	Returns the completion state of the current traced operation.
GETTRACE:EXCEPTPROV	Returns the exception that has generated the traced operation.
GETTRACE:FLAG	Returns a reference to the FLAG type object concerned by the traced operation.
GETTRACE:FLISTGET	Returns references to the FLAG objects related to the current traced operation when several objects of this type are involved.

GETTRACE:LCLASSNB	Returns the number of classes of the requests specified for the current traced operation when this number is greater than one.
GETTRACE:LCUSTNB	Returns the number of partner customers of the current traced operation when this number is greater than one.
GETTRACE:LFLAGNB	Returns the number of FLAG objects related to the current traced operation, when this number is greater than one.
GETTRACE:LNUMNB	Returns the cardinal of the set of integer numbers (numbers of requests for instance) specified for the current traced operation when this number is greater than one.
GETTRACE:LPRIONB	Returns the number of priorities specified for the current traced operation when this number is greater than one.
GETTRACE:LQUNB	Returns the number of queues concerned by the current traced operation in case this number is *a priori* unknown.
GETTRACE:NAMECODE	Converts the integer code of an event to the corresponding symbolic name.
GETTRACE:NUMBER	Returns the integer numerical parameter of the current traced operation.
GETTRACE:NUMLISTGET	Returns one by one the values of the numbers (those set's cardinal can be greater than 1) specified for the current traced operation.
GETTRACE:PARDISTR	Returns the service time distribution law parameters during a trace treatment.
GETTRACE:PRILISTGET	Returns one by one the priorities of customers (their number can be greater than 1) specified for the current traced operation.
GETTRACE:QLISTGET	Returns references to a set of queues related the current traced operation.
GETTRACE:QPROVOKE	Returns a reference on the queue directly associated to the traced operation cause.
GETTRACE:QSECONDR	Returns a reference on the queue considered as the traced operation partner.
GETTRACE:QSUBJECT	Returns a reference to the queue considered as subject to the traced operation.
GETTRACE:TIMERPROVO	Returns the timer which produced the traced operation.
GETTRACE:WHICHPROVO	Allows to test if the operation being traced was produced by a customer, an exception or a timer.
GETTRACE:TIMERSUBJE	Returns the TIMER object supporting the traced operation.

The `GETTRACE` function, with the pertinent keywords allows to get the parameters of the traced operation except the calls to the `GETTRACE:CODENAME` and `GETTRACE:NAMECODE` functions which perform simple convert operations between symbolic names (see "Event types") and corresponding integer codes.

EVALUATION

During the execution and only in trace treatment in simulation, except `GETTRACE:CODENAME` and `GETTRACE:NAMECODE` which may be used anywhere.

WARNING

`GETTRACE` functions, except `GETTRACE:CODENAME` and `GETTRACE:NAMECODE`, must be called in a trace treatment, otherwise an error occurs.

In addition, the only calls available for every event are `GETTRACE:EVCODE` which gives the event code and `GETTRACE:EVSTATUS` which returns the event state.

The other calls are allowed only if they make sense according to the current traced event. If not, an error occurs and the simulation stops. The possible calls are exhaustively described for each event.

NAME

GETTRACE:CCLASS - Returns the customers class passed as argument to a traced simulation operation.

SYNTAX

```
GETTRACE:CCLASS;
```

DESCRIPTION

This function allows particularly to obtain the customers class passed as an argument to a procedure such as TRANSIT.

EVALUATION

During a trace user treatment in simulation.

WARNING

Any call to this function outside a trace treatment leads to an error.

SEE ALSO

GETTRACE:CPRIOR

EXAMPLE

```
/DECLARE/ PROCEDURE TREATP;
        BEGIN
          PRINT ("customer:", GETTRACE:CPROVOKE," class:", GETTRACE:CCLASS);
        END;

/EXEC/ BEGIN
        SETTRACE:ON;
        SETTRACE:SET(Q1,"TRANSIT",TREATP);
        SIMUL;
       END;
```

GETTRACE:CLISTGET

NAME

GETTRACE:CLISTGET - Returns one by one the references to the customers (their number can be greater than 1) considered as partners of a traced simulation operation.

SYNTAX

GETTRACE:CLISTGET (*integer*);

DESCRIPTION

In the case of a traced operation when several customers may be potential partners, this function gives access to all the concerned customers. The references to these customers are returned one by one depending on the integer value passed as an argument. This one can vary from 1 to the maximum value obtained by calling the GETTRACE:LCUSTNB function.

EVALUATION

During a trace user treatment in simulation.

WARNING

Any call to this function outside a trace treatment leads to an error. It is in the same way if the integer passed as an argument integer is outside the possible limits: negative or null, or upper to the result returned by GETTRACE:LCUSTNB.

SEE ALSO

GETTRACE:LCUSTNB

EXAMPLE

```
/DECLARE/ PROCEDURE TREATP;
          INTEGER N,II;
          BEGIN
            N:= GETTRACE:LCUSTNB;
            FOR II:=1 STEP 1 UNTIL N DO
              PRINT (II," : ", GETTRACE:CLISTGET(II));
          END;

/EXEC/ BEGIN
        SETTRACE:ON;
        SETTRACE:SET(Q1,"JOIN",TREATP);
        SIMUL;
       END;
```

NAME

GETTRACE:CLLISTGET - Returns one by one the references to the classes of the requests (their number can be greater than 1) specified for the current traced operation.

SYNTAX

GETTRACE:CLLISTGET (*integer*);

DESCRIPTION

GETTRACE:CLLISTGET (*integer*) returns a REF CLASS object.

In the case of a traced operation when several request classes may be specified (PMULT for instance), this function gives access to all the concerned classes. The references to these classes are returned one by one depending on the integer value passed as an argument. This one can vary from 1 to the maximum value obtained by calling the GETTRACE:LCLASSNB function.

The requests are requests of pass grants to one or several semaphores or resource units. The request classes are the classes with which the requests are performed (see PMULT mechanism). By default, they are equal to the class of the customer performing them.

EVALUATION

During a trace user treatment in simulation.

WARNING

Any call to this function outside a trace treatment leads to an error. An error occurs if the integer argument is negative or null or greater than GETTRACE:LCLASSNB.

SEE ALSO

GETTRACE:LCLASSNB - PMULT

EXAMPLE

```
/DECLARE/ PROCEDURE treatp;
          INTEGER n,ii;
          REF CLASS @cl;
          BEGIN
            n:= GETTRACE:LCLASSNB;
            FOR ii:=1 STEP 1 UNTIL n DO BEGIN
              @cl:= GETTRACE:CLLISTGET(ii);
              PRINT (ii," : ", @cl);
            END;
          END;

/EXEC/ BEGIN
        SETTRACE:ON;
        SETTRACE:SET(Q1,"PMULT",treatp);
        SIMUL;
```

```
END;
```

NAME

GETTRACE:CODENAME - Converts the symbolic name of an event to the corresponding positive integer code.

SYNTAX

GETTRACE:CODENAME (*string*) ;

DESCRIPTION

The argument of string type supposed to be the symbolic name of an event type (see "Event types") is converted to the corresponding positive integer code.

The GETTRACE:NAMECODE function performs the opposite operation.

EVALUATION

During the execution.

WARNING

A warning occurs if the string in argument contains an unknown event name.

SEE ALSO

GETTRACE:NAMECODE - GETTRACE:EVSTATUS - GETTRACE:EVCODE

EXAMPLE

```
 /DECLARE/ PROCEDURE USTRACE;
           BEGIN
             IF (GETTRACE:EVCODE := GETTRACE:CODENAME ("WAIT")
                         THEN PRINT ("WAITING ON FLAG", GETTRACE:FLAG);
           END ;
```

GETTRACE:CPRIOR

NAME

GETTRACE:CPRIOR - Returns the priority of customers passed as argument of a simulation operation during a trace.

SYNTAX

```
GETTRACE:CPRIOR;
```

DESCRIPTION

During a call to the TRANSIT procedure for instance, allows to obtain the specified customers priority.

EVALUATION

During a trace user treatment in simulation.

WARNING

Any call to this function outside a trace treatment leads to an error.

SEE ALSO

GETTRACE:CCLASS

EXAMPLE

```
/DECLARE/ PROCEDURE TREATP;
          BEGIN
            PRINT (" class:", GETTRACE:CCLASS," priority:", GETTRACE:CPRIOR);
          END;

/EXEC/ BEGIN
          SETTRACE:ON;
          SETTRACE:SET(Q1,"TRANSIT",TREATP);
          SIMUL;
        END;
```

NAME

GETTRACE:CPROVOKE - Returns the reference on the customer who produced the traced simulation operation.

SYNTAX

```
GETTRACE:CPROVOKE;
```

DESCRIPTION

The customer who produced a traced operation during a simulation is the current customer, computing the operation either for himself or for an other customer, for instance in the case of a customer forcing an other one to wait on a FLAG, or to perform a transit, etc ...

EVALUATION

During the execution of a trace computing procedure.

NOTE

The customer producing an operation may not exist if the operation was produced by an exception or a timer treatment. The GETTRACE:WHICHPRO, GETTRACE:EXCEPTPR, and GETTRACE:TIMERPRO functions must sometimes be used instead of GETTRACE:CPROVOKE in these cases.

WARNING

Any call to this function outside a trace treatment leads to an error.

SEE ALSO

GETTRACE:WHICHPRO - GETTRACE:EXCEPTPR - GETTRACE:TIMERPRO

EXAMPLE

```
/DECLARE/ PROCEDURE TREATP;
          BEGIN
            PRINT ("resource :", GETTRACE:QSUBJECT);
            PRINT (" customer:",  GETTRACE:CPROVOKE);
          END;

/EXEC/ BEGIN
        SETTRACE:ON;
        SETTRACE:SET(Q1,"V",TREATP);
        SETTRACE:SET(Q2,"P",TREATP);
        SIMUL;
       END;
```

GETTRACE:CSECONDR

NAME

GETTRACE:CSECONDR - Returns the reference on the customer, when existing, considered as simple partner of the traced simulation operation.

SYNTAX

```
GETTRACE:CSECONDR;
```

DESCRIPTION

The customer partner of a traced operation during a simulation is a customer, when existing, referenced by the operation. For instance, it is the case of the customer before or behind whom one puts itself in a BEFCUST or AFTCUST operation. Its signification comes, the more often, from a convention linked to the treated operation.

EVALUATION

During the execution of a trace treatment procedure.

NOTE

This function returns the reference on the partner customer of a simulation operation when this partner is unique. For some operations, several customers may meanwhile exist and the GETTRACE:LCUSTNB and GETTRACE:CLISTGET functions must then be used.

WARNING

Any call to this function outside a trace treatment leads to an error.

SEE ALSO

GETTRACE:CPROVOKE - GETTRACE:CSUBJECT - GETTRACE:LCUSTNB - GETTRACE:CLISTGET

EXAMPLE

```
/DECLARE/ PROCEDURE TREATP;
          BEGIN
            PRINT (" active customer:", GETTRACE:CPROVOKE);
            PRINT (" concerned customer:", GETTRACE:CSECONDR);
          END;

/EXEC/ BEGIN
        SETTRACE:ON;
        SETTRACE:SET(Q1,"AFTCUST",TREATP);
        SIMUL;
       END;
```

NAME

GETTRACE:CSUBJECT - Returns the reference on the customer, when existing, who was subject of the traced simulation operation.

SYNTAX

```
GETTRACE:CSUBJECT;
```

DESCRIPTION

The customer subject to a traced operation during a simulation is the customer running a service time delay or a synchronization, for example, either for himself or forced by an other customer, an exception or a timer, to synchronize itself, to transit, or any other operation.

EVALUATION

During the execution of a trace treatment procedure.

NOTE

The definition of the customer considered as submitted to an operation can depend in some cases of the traced operation, even making sense only by convention; for instance, it is the case of a customer created by calling the NEW function.

WARNING

Any call to this function outside a trace treatment leads to an error.

SEE ALSO

GETTRACE:CPROVOKE - GETTRACE:CSECONDR - GETTRACE:LCUSTNB - GETTRACE:CLISTGET

EXAMPLE

```
/DECLARE/ PROCEDURE TREATP;
          BEGIN
            PRINT (" station :", GETTRACE:QSUBJECT);
            PRINT (" customer:", GETTRACE:CSUBJECT);
          END;
```

GETTRACE:DELAY

NAME

GETTRACE:DELAY - Returns the value of a service time delay during a trace treatment in simulation.

SYNTAX

```
GETTRACE:DELAY;
```

DESCRIPTION

Allows to know the expected delay for a customer service time after the call to a corresponding procedure.

EVALUATION

During a trace user treatment in simulation.

WARNING

Any call to this function outside a trace treatment leads to an error.

SEE ALSO

GETTRACE:DISTRI - GETTRACE:PARDISTR

EXAMPLE

```
/DECLARE/ PROCEDURE TREATP;
          BEGIN
            PRINT ("service time :", GETTRACE:DELAY);
          END;

/EXEC/ BEGIN
        SETTRACE:ON;
        SETTRACE:SET(Q1,"SERVTIME",TREATP);
        SIMUL;
       END;
```

NAME

GETTRACE:DISTRI - Returns during a simulation trace treatment the distribution law of a service time that has just began.

SYNTAX

```
GETTRACE:DISTRI;
```

DESCRIPTION

When a customer has just requested a service time by calling a procedure such as CST, EXP, ERLANG, HEXP, etc ..., this function allows to know the used distribution law. The nature of this law is returned by an integer and follows the code :

1. CST
2. EXP
3. ERLANG
4. HEXP
5. UNIFOR
6. RANDI
7. COX
8. NORMAL
9. RANDU

The distribution parameters may be obtained by the GETTRACE:PARDISTR function.

EVALUATION

During a trace user treatment in simulation.

WARNING

Any call to this function outside a trace treatment leads to an error.

SEE ALSO

GETTRACE:DELAY - GETTRACE:PARDISTR

EXAMPLE

```
/DECLARE/ PROCEDURE TREATP;
          BEGIN
            PRINT ("service law :", GETTRACE:DISTRI);
          END;

/EXEC/ BEGIN
        SETTRACE:ON;
        SETTRACE:SET(Q1,"SERVTIME",TREATP);
        SIMUL;
       END;
```

NAME

GETTRACE:EVCODE - Returns the current event type traced during the simulation.

SYNTAX

GETTRACE:EVCODE ;

DESCRIPTION

This function returns an integer which represents the current event type traced during the simulation.

This integer code may be associated to the symbolic name of the corresponding event.

The GETTRACE:CODENAME function allows to convert a symbolic name to the corresponding integer code. The GETTRACE:NAMECODE function performs the opposite operation.

EVALUATION

During a trace user treatment in simulation.

NOTES

This call must precede any other call to GETTRACE:key-word in the trace treatment, except if you are sure of the handled event.

WARNING

Any call to this function outside a trace treatment leads to an error.

The integer values of the event codes may be modified in the successive QNAP2 versions. The only certainty is that all the possible values are integer greater than zero. The values returned by GETTRACE:EVCODE do not have to be directly compared with the integer used in the QNAP2 model in form of constants. Please, always use the result of GETTRACE:CODENAME or GETTRACE:NAMECODE, either directly, either after storage of the value in user variables.

SEE ALSO

GETTRACE:CODENAME - GETTRACE:NAMECODE - Event type

EXAMPLE

```
/DECLARE/ PROCEDURE USTRACE;
          BEGIN
            IF (GETTRACE:EVCODE = GETTRACE:CODENAME ("WAIT"))
              THEN PRINT ("WAIT ON FLAG", GETTRACE:FLAG);
          END;
```

GETTRACE:EVSTATUS

NAME

GETTRACE:EVSTATUS - Returns the completion state of a traced simulation operation.

SYNTAX

```
GETTRACE:EVSTATUS;
```

DESCRIPTION

During a trace treatment, returns the completion state of a traced simulation operation. This state allows to test if a waiting state is passing or blocking. The returned value is zero without producing any error, for the operations returning no significant state value.

These codes are proper to each considered event type.

EVALUATION

During a trace user treatment in simulation.

NOTE

The signification of a simulation operation completion state depends on the nature of this operation.

WARNING

Any call to this function outside a trace treatment leads to an error.

SEE ALSO

Event types.

EXAMPLE

```
/DECLARE/ PROCEDURE TREATP;
          BEGIN
            PRINT ("resource :", GETTRACE:QSUBJECT);
            IF (GETTRACE:EVSTATUS < 1) THEN
             PRINT ("Through operation");
          END;

/EXEC/ BEGIN
        SETTRACE:ON;
        SETTRACE:SET(Q2,"P",TREATP);
        SIMUL;
       END;
```

NAME

GETTRACE:EXCEPTPROVOKE - Returns the exception (when existing) that generated the traced simulation operation.

SYNTAX

```
GETTRACE:EXCEPTPROVOKE;
```

DESCRIPTION

In the case when a simulation operation was generated by an exception treatment, and not by a customer or a timer, which is identified by the GETTRACE:WHICHPROVOKE call, this function allows to obtain the reference to this exception.

EVALUATION

During a trace user treatment in simulation.

WARNING

Any call to this function outside a trace treatment leads to an error. It is in the same way when the traced operation was generated by a customer or a timer, which is identified by a GETTRACE:WHICHPROVOKE call.

SEE ALSO

GETTRACE:WHICHPROVOKE - GETTRACE:TIMERPROVOKE - GETTRACE:CPROVOKE

EXAMPLE

```
/DECLARE/ PROCEDURE TREATP;
          BEGIN
           IF (GETTRACE:WHICHPROVOKE = 2) THEN
            PRINT (" exception:", GETTRACE:EXCEPTPROVOKE);
          END;
```

GETTRACE:FLAG

NAME

GETTRACE:FLAG - Returns the reference to the FLAG type object concerned by the traced simulation operation.

SYNTAX

```
GETTRACE:FLAG;
```

DESCRIPTION

Returns the reference to the FLAG type object, when existing, which is the only one concerned by the traced operation: SET, RESET, WAIT, etc ...

EVALUATION

During a trace user treatment in simulation.

NOTE

For operations concerning several FLAG type objects as WAITAND and WAITOR, the GETTRACE:LFLAGNB and GETTRACE:FLISTGET functions must be used.

WARNING

Any call to this function outside a trace treatment leads to an error.

SEE ALSO

GETTRACE:LFLAGNB - GETTRACE:FLISTGET

EXAMPLE

```
/DECLARE/ PROCEDURE TREATP;
          BEGIN
            PRINT ("flag :", GETTRACE:FLAG);
            PRINT (" customer:", GETTRACE:CPROVOKE);
          END;

/EXEC/ BEGIN
        SETTRACE:ON;
        SETTRACE:SET(Q1,"WAIT",TREATP);
        SIMUL;
       END;
```

NAME

GETTRACE:FLISTGET - Returns the references to the FLAG objects concerned by a simulation operation when several objects of this type are involved.

SYNTAX

GETTRACE:FLISTGET (*integer*);

DESCRIPTION

This function, returns one by one all the references to the FLAG objects concerned by an operation such as WAITAND or WAITOR.

The integer argument can vary from 1 to the maximum number returned by GETTRACE:LFLAGNB.

EVALUATION

During a trace user treatment in simulation.

WARNING

Any call to this function outside a trace treatment leads to an error. It is in the same way if the integer argument is negative or null, or upper to the maximum value returned by GETTRACE:LFLAGNB.

SEE ALSO

GETTRACE:LFLAGNB

EXAMPLE

```
/DECLARE/ PROCEDURE TREATP;
          INTEGER N,II;
          BEGIN
            N:= GETTRACE:LFLAGNB;
            FOR II:=1 STEP 1 UNTIL N DO
              PRINT (II," : ", GETTRACE:FLISTGET(II));
          END;

/EXEC/ BEGIN
        SETTRACE:ON;
        SETTRACE:SET(Q1,"WAITAND",TREATP);
        SIMUL;
       END;
```

GETTRACE:LCLASSNB

NAME

GETTRACE:LCLASSNB - Returns the number of classes of the requests specified for the current traced operation when this number is greater than one.

SYNTAX

```
GETTRACE:LCLASSNB;
```

DESCRIPTION

GETTRACE:LCLASSNB returns an INTEGER.

This function allows to know for instance the number of request classes specified for a PMULT operation. The GETTRACE:CLLISTGET function allows then to obtain one by one the references to all these classes.

The requests are requests of pass grants to one or several semaphores or resource units. The request classes are the classes with which the requests are performed (see PMULT mechanism). By default, they are equal to the class of the customer performing them.

EVALUATION

During a trace user treatment in simulation.

WARNING

Any call to this function outside a trace treatment leads to an error.

SEE ALSO

GETTRACE:CLLISTGET - PMULT

EXAMPLE

```
/DECLARE/ PROCEDURE treatp;
            INTEGER n;
          BEGIN
            n:= GETTRACE:LCLASSNB;
            PRINT ("number of requests classes:", n);
          END;

/EXEC/ BEGIN
        SETTRACE:ON;
        SETTRACE:SET(Q1,"PMULT",treatp);
        SIMUL;
       END;
```

NAME

GETTRACE:LCUSTNB - Returns the number of partner customers of a traced simulation operation when this number is greater than one.

SYNTAX

```
GETTRACE:LCUSTNB;
```

DESCRIPTION

This function allows to know for instance the number of customers to wait for by a JOIN operation. The GETTRACE:CLISTGET function allows then to obtain one by one the references to all these customers.

EVALUATION

During a trace user treatment in simulation.

WARNING

Any call to this function outside a trace treatment leads to an error.

SEE ALSO

GETTRACE:CLISTGET

EXAMPLE

```
/DECLARE/ PROCEDURE TREATP;
          BEGIN
            PRINT ("number of customers :", GETTRACE:LCUSTNB);
          END;

/EXEC/ BEGIN
        SETTRACE:ON;
        SETTRACE:SET(Q1,"JOIN",TREATP);
        SIMUL;
       END;
```

GETTRACE:LFLAGNB

NAME

GETTRACE:LFLAGNB - Returns the number of FLAG objects concerned by a traced simulation operation, when this number is greater than one.

SYNTAX

```
GETTRACE:LFLAGNB;
```

DESCRIPTION

This function allows to compute the number of FLAG objects concerned by an operation such as WAITAND ou WAITOR. The GETTRACE:FLISTGET function allows then to obtain the references to those FLAG objects.

EVALUATION

During a trace user treatment in simulation.

WARNING

Any call to this function outside a trace treatment leads to an error.

SEE ALSO

GETTRACE:FLISTGET

EXAMPLE

```
/DECLARE/ PROCEDURE TREATP;
          INTEGER N,II;
          BEGIN
            N:= GETTRACE:LFLAGNB;
            FOR II:=1 STEP 1 UNTIL N DO
              PRINT (II," : ", GETTRACE:FLISTGET(II));
          END;

/EXEC/ BEGIN
        SETTRACE:ON;
        SETTRACE:SET(Q1,"WAITAND",TREATP);
        SIMUL;
       END;
```

NAME

GETTRACE:LNUMNB - Returns the number of integer numbers specified for the current traced operation (number of sets of requests for instance) when this number is greater than one.

SYNTAX

```
GETTRACE:LNUMNB;
```

DESCRIPTION

GETTRACE:LNUMNB returns an INTEGER.

This function allows to know for instance the number of sets of requests specified for a PMULT or VMULT operation. The GETTRACE:NUMLISTGET function allows then to obtain one by one the values of all these numbers.

The requests are requests of pass grants to one or several semaphores or resource units. For example, if a customer performs a PMULT((sem1,sem2),(3,1)), where sem1 and sem2 are SEMAPHORE or RESOURCE queues, and if this operation is traced, GETTRACE:LNUMNB will return 2, because two integer numbers (corresponding to the numbers of pass grants requested to sem1 and sem2) are specified.

EVALUATION

During a trace user treatment in simulation.

WARNING

Any call to this function outside a trace treatment leads to an error.

SEE ALSO

GETTRACE:NUMLISTGET - PMULT - VMULT

EXAMPLE

```
/DECLARE/ PROCEDURE treatp;
            INTEGER n;
          BEGIN
            n:= GETTRACE:LNUMNB;
            PRINT ("number of request's sets :", n);
          END;

/EXEC/ BEGIN
        SETTRACE:ON;
        SETTRACE:SET(Q1,"PMULT",treatp);
        SIMUL;
       END;
```

GETTRACE:LPRIONB

NAME

GETTRACE:LPRIONB - Returns the number of the priorities of the requests specified for the current traced operation when this number is greater than one.

SYNTAX

GETTRACE:LPRIONB;

DESCRIPTION

GETTRACE:LPRIONB returns an INTEGER.

This function allows to know the number of the priorities specified for a PMULT operation. The GETTRACE:PRILISTGET function allows then to obtain one by one the values of all these priorities.

The requests are requests of pass grants to one or several semaphores or resource units. The request priorities are the priorities with which the requests are performed (see PMULT mechanism). By default, they are equal to the priority of the customer performing them.

EVALUATION

During a trace user treatment in simulation.

WARNING

Any call to this function outside a trace treatment leads to an error.

SEE ALSO

GETTRACE:PRILISTGET - PMULT

EXAMPLE

```
/DECLARE/ PROCEDURE treatp;
            INTEGER n;
          BEGIN
            n:= GETTRACE:LPRIONB;
            PRINT ("number of priorities specified :", n);
          END;

/EXEC/ BEGIN
        SETTRACE:ON;
        SETTRACE:SET(Q1,"PMULT",treatp);
        SIMUL;
       END;
```

NAME

GETTRACE:LQUNB - Returns the number of queues concerned by a traced simulation operation in the case where this number is apriori unknown.

SYNTAX

```
GETTRACE:LQUNB;
```

DESCRIPTION

This function was introduced to compute the number of queues concerned by an operation such as BLOCK ou UNBLOCK.

The GETTRACE:QLISTGET function allows to obtain one by one the references to these queues.

EVALUATION

During a trace user treatment in simulation.

WARNING

Any call to this function outside a trace treatment leads to an error.

SEE ALSO

GETTRACE:QLISTGET

EXAMPLE

```
/DECLARE/ PROCEDURE TREATP;
          INTEGER N,II;
          BEGIN
            N:=GETTRACE:LQUNB;
            FOR II:=1 STEP 1 UNTIL N DO
              PRINT (II," : ", GETTRACE:QLISTGET(II));
          END;

/EXEC/ BEGIN
        SETTRACE:ON;
        SETTRACE:SET(Q1,"BLOCK","UNBLOCK",TREATP);
        SIMUL;
       END;
```

GETTRACE:NAMECODE

NAME

GETTRACE:NAMECODE - Converts the integer code of an event to the corresponding symbolic name.

SYNTAX

GETTRACE:NAMECODE (*integer*) ;

DESCRIPTION

The integer passed in argument represents the numerical code of the event (such as returned by GETTRACE:EVCODE) to be converted.

If the string returned by the function is stored in a variable, this variable must have a length greater than eight in order to never have to truncate the returned value.

EVALUATION

During a trace user treatment in simulation.

WARNING

An error occurs if the argument is negative or zero or does not represent a valid event code.

SEE ALSO

GETTRACE:CODENAME - GETTRACE:EVCODE - Event type.

EXAMPLE

```
/EXEC/ BEGIN
         IF ( GETTRACE:NAMECODE (GETTRACE:CODENAME("WAIT")) <> "WAIT")
           THEN PRINT ("CODING ERROR") ;
       END ;
```

NAME

GETTRACE:NUMBER - Returns the integer numerical parameter, when existing and unique, of a traced simulation operation.

SYNTAX

GETTRACE:NUMBER;

DESCRIPTION

This function allows for instance to obtain the integer argument of the JOIN (*number_of_customers*) procedure.

In other cases, a numerical parameter characteristic of the traced operation is conventionnaly returned.

EVALUATION

During a trace user treatment in simulation.

WARNING

Any call to this function outside a trace treatment leads to an error.

EXAMPLE

```
/DECLARE/ PROCEDURE TREATP;
          BEGIN
            PRINT ("JOIN on ", GETTRACE:NUMBER," clients.");
          END;

/EXEC/ BEGIN
        SETTRACE:ON;
        SETTRACE:SET(Q1,"JOIN",TREATP);
        SIMUL;
       END;
```

GETTRACE:NUMLISTGET

NAME

GETTRACE:NUMLISTGET - Returns one by one the values of the numbers specified for the current traced operation.

SYNTAX

GETTRACE:NUMLISTGET (*integer*);

DESCRIPTION

GETTRACE:NUMLISTGET (*integer*) returns an INTEGER.

In the case of a traced operation when several numbers may be specified (numbers of requests on different semaphores or resources on a PMULT or VMULT operation for instance), this function gives access to all the values of these specified numbers. The values of these numbers are returned one by one depending on the integer value passed as an argument. This one can vary from 1 to the maximum value obtained by calling the GETTRACE:LNUMNB function.

The requests are requests of pass grants to one or several semaphores or resource units. For example, if a customer performs a PMULT((sem1,sem2),(3,1)), where sem1 and sem2 are SEMAPHORE or RESOURCE queues, and if this operation is traced, GETTRACE:LNUMNB will return 2, GETTRACE:NUMLISTGET (1) will return 3 and GETTRACE:NUMLISTGET (2) will return 1.

EVALUATION

During a trace user treatment in simulation.

WARNING

Any call to this function outside a trace treatment leads to an error. An error occurs if the integer argument is negative or null or greater than GETTRACE:LNUMNB.

SEE ALSO

GETTRACE:LNUMNB - PMULT - VMULT

EXAMPLE

```
/DECLARE/ PROCEDURE treatp;
          INTEGER n,ii;
          INTEGER num;
          BEGIN
            n:= GETTRACE:LNUMNB;
            FOR ii:=1 STEP 1 UNTIL n DO BEGIN
              num:= GETTRACE:NUMLISTGET(ii);
              PRINT (ii," : ", num);
            END;
          END;

/EXEC/ BEGIN
        SETTRACE:ON;
```

```
 SETTRACE:SET(Q1,"PMULT","VMULT",treatp);
 SIMUL;
END;
```

GETTRACE:PARDISTR

NAME

GETTRACE:PARDISTR - Returns the service time distribution law parameters during a trace treatment in simulation.

SYNTAX

GETTRACE:PARDISTR (*integer*);

DESCRIPTION

Allows, during a user trace treatment in simulation, to obtain the service time distribution parameters (maximum three) that have just been used, when current treated event is a time delay request. These parameters are the values of the call arguments, in the right order, of the service time request procedure used. The non-existent parameters are supposed to be null and in that case, no mistake happens.

EVALUATION

During a trace user treatment in simulation.

WARNING

Any call to this function outside a trace treatment leads to an error. It is in the same way if call argument to this function is not an integer in the range between 1 and 3.

SEE ALSO

GETTRACE:DISTRI - GETTRACE:DELAY

EXAMPLE

```
/DECLARE/ PROCEDURE TREATP;
          BEGIN
            PRINT ("service law :", GETTRACE:DISTRI);
            PRINT ("first parameter :", GETTRACE:PARDISTR(1));
          END;

/EXEC/ BEGIN
        SETTRACE:ON;
        SETTRACE:SET(Q1,"SERVTIME",TREATP);
        SIMUL;
       END;
```

NAME

GETTRACE:PRILISTGET - Returns one by one the values of the priorities of the requests specified for the current traced operation.

SYNTAX

GETTRACE:PRILISTGET (*integer*);

DESCRIPTION

GETTRACE:PRILISTGET (*integer*) returns an INTEGER.

In the case of a traced operation when several customer's priorities may be specified (on a PMULT operation for instance), this function gives access to all the values of these priorities. These values are returned one by one depending on the integer value passed as an argument. This one can vary from 1 to the maximum value obtained by calling the GETTRACE:LPRIONB function.

The requests are requests of pass grants to one or several semaphores or resource units. The request priorities are the priorities with which the requests are performed (see PMULT mechanism). By default, they are equal to the priority of the customer performing them.

EVALUATION

During a trace user treatment in simulation.

WARNING

Any call to this function outside a trace treatment leads to an error. It is in the same way if the integer passed as an argument integer is outside the possible limits: negative or null, or upper to the result returned by GETTRACE:LPRIONB.

SEE ALSO

GETTRACE:LPRIONB - PMULT

EXAMPLE

```
/DECLARE/ PROCEDURE treatp;
          INTEGER n,ii;
          INTEGER num;
          BEGIN
            n:= GETTRACE:LPRIONB;
            FOR ii:=1 STEP 1 UNTIL n DO BEGIN
              num:= GETTRACE:PRILISTGET(ii);
              PRINT (ii," : ", num);
            END;
          END;

/EXEC/ BEGIN
        SETTRACE:ON;
        SETTRACE:SET(Q1,"PMULT",treatp);
```

```
 SIMUL;
END;
```

NAME

GETTRACE:QLISTGET - Returns the references to a set of queues concerned by a traced simulation operation.

SYNTAX

GETTRACE:QLISTGET (*integer*);

DESCRIPTION

This function, linked to GETTRACE:LQUNB which returns the number of these queues, returns one by one the references to all the queues concerned for instance by a BLOCK or UNBLOCK operation. The integer passed as argument can vary from 1 to the number returned by GETTRACE:LQUNB.

EVALUATION

During a trace user treatment in simulation.

WARNING

Any call to this function outside a trace treatment leads to an error. It is in the same way if the passed as an argument integer is outside the possible limits: negative or null, or superior to the result returned by GETTRACE:LQUNB.

SEE ALSO

GETTRACE:LQUNB

EXAMPLE

```
/DECLARE/ PROCEDURE TREATP;
          INTEGER N,II;
          BEGIN
            N:= GETTRACE:LQUNB;
            FOR II:=1 STEP 1 UNTIL N DO
              PRINT (II," : ", GETTRACE:QLISTGET(II));
          END;

/EXEC/ BEGIN
        SETTRACE:ON;
        SETTRACE:SET(Q1,"BLOCK","UNBLOCK",TREATP);
        SIMUL;
       END;
```

GETTRACE:QPROVOKE

NAME

GETTRACE:QPROVOKE - Returns the reference on the queue directly associated to the traced simulation operation cause.

SYNTAX

```
GETTRACE:QPROVOKE;
```

DESCRIPTION

The queue which produced the operation is the more often the one containing the current customer, but its definition may result of a convention. In the case of a customer transition, this queue is considered as equal to the departure queue.

EVALUATION

During the execution of a trace treatment procedure.

WARNING

Any call to this function outside a trace treatment leads to an error.

SEE ALSO

GETTRACE:QSUBJECT - GETTRACE:QSECONDR - GETTRACE:LQUNB - GETTRACE:QLISTGET

EXAMPLE

```
/DECLARE/ PROCEDURE TREATP;
          BEGIN
            PRINT (" station :", GETTRACE:QPROVOKE);
            PRINT (" customer:", GETTRACE:CPROVOKE);
          END;

/EXEC/ BEGIN
        SETTRACE:ON;
        SETTRACE:SET(Q1,TREATP);
        SETTRACE:SET(Q2,TREATP);
        SIMUL;
       END;
```

NAME

GETTRACE:QSECONDR - Returns the reference on the queue, when existing, considered as the traced simulation operation partner.

SYNTAX

```
GETTRACE:QSECONDR;
```

DESCRIPTION

The partner queue of a simulation operation is always defined by a convention adapted to the traced operation. When several queues can be considered as partners, the GETTRACE:LQUNB and GETTRACE:QLISTGET functions must be used.

EVALUATION

During the execution of a trace computing procedure.

WARNING

Any call to this function outside a trace treatment leads to an error.

SEE ALSO

GETTRACE:QPROVOKE - GETTRACE:QSUBJECT - GETTRACE:LQUNB - GETTRACE:QLISTGET

EXAMPLE

```
/DECLARE/ PROCEDURE TREATP;
          BEGIN
            PRINT ("departure station :", GETTRACE:QPROVOKE);
            PRINT ("arrival station :", GETTRACE:QSECONDR);
          END;

/EXEC/ BEGIN
        SETTRACE:ON;
        SETTRACE:SET(Q1,"TRANSIT",TREATP);
        SETTRACE:SET(Q2,"TRANSIT",TREATP);
        SIMUL;
       END;
```

GETTRACE:QSUBJECT

NAME

GETTRACE:QSUBJECT - Returns the reference to the queue (when existing) considered as submitted to the traced simulation operation.

SYNTAX

```
GETTRACE:QSUBJECT;
```

DESCRIPTION

The queue submitted to a simulation operation is the more often defined by convention. It is the arrival queue for a transition, the semaphore or resource in the case of a P or V operation, etc ...

EVALUATION

During the execution of a trace computing procedure.

WARNING

Any call to this function outside a trace treatment leads to an error.

SEE ALSO

GETTRACE:QPROVOKE - GETTRACE:QSECONDR - GETTRACE:LQUNB - GETTRACE:QLISTGET

EXAMPLE

```
/DECLARE/ PROCEDURE TREATP;
          BEGIN
            PRINT ("resource :", GETTRACE:QSUBJECT);
            PRINT (" customer:", GETTRACE:CPROVOKE);
          END;

/EXEC/ BEGIN
        SETTRACE:ON;
        SETTRACE:SET(Q1,"V",TREATP);
        SETTRACE:SET(Q2,"P",TREATP);
        SIMUL;
       END;
```

NAME

GETTRACE:TIMERPROVOKE - Returns the timer which has produced the traced simulation operation.

SYNTAX

```
GETTRACE:TIMERPROVOKE;
```

DESCRIPTION

In the case when a simulation operation during a trace treatment was generated by a timer, this function returns the reference to this timer.

EVALUATION

During a trace user treatment in simulation.

WARNING

Any call to this function outside a trace treatment leads to an error. It is in the same way when the traced operation was generated by a customer or an exception, which is identified by a GETTRACE:WHICHPROVOKE call.

SEE ALSO

GETTRACE:WHICHPROVOKE - GETTRACE:EXCEPTPROVOKE - GETTRACE:CPROVOKE

EXAMPLE

```
/DECLARE/ PROCEDURE TREATP;
          BEGIN
           IF (GETTRACE:WHICHPROVOKE =3) THEN
            PRINT ("timer:", GETTRACE:TIMERPROVOKE);
          END;
```

GETTRACE:TIMERSUBJECT

NAME

GETTRACE:TIMERSUBJECT - Returns the TIMER object supporting the traced simulation operation.

SYNTAX

```
GETTRACE:TIMERSUBJECT;
```

DESCRIPTION

Returns the reference to the timer object having supported an operation in simulation, particularly an activation or a pause.

EVALUATION

During a trace user treatment in simulation.

WARNING

Any call to this function outside a trace treatment leads to an error.

EXAMPLE

```
/DECLARE/ PROCEDURE TREATP;
          BEGIN
           PRINT (" timer:", GETTRACE:TIMERSUBJECT);
          END;
```

NAME

GETTRACE:WHICHPROVOKE - Allows to test if the simulation operation being traced was produced by a customer, an exception or a timer.

SYNTAX

```
GETTRACE:WHICHPROVOKE;
```

DESCRIPTION

This function returns an integer code depending on the nature (customer, exception or timer) of the object having produced the simulation operation during the trace. The returned integer code values are:

- 1 : customer produced operation.
- 2 : exception produced operation.
- 3 : timer produced operation.

EVALUATION

During a trace user treatment in simulation.

NOTE

The test of the value returned by this function must normally be a preliminary to the use of GETTRACE:CPROVOKE, GETTRACE:EXCEPTPROVOKE or GETTRACE:TIMERPROVOKE in order to avoid execution errors.

WARNING

Any call to this function outside a trace treatment leads to an error.

SEE ALSO

GETTRACE:CPROVOKE - GETTRACE:EXCEPTPROVOKE - GETTRACE:TIMERPROVOKE

EXAMPLE

```
/DECLARE/ PROCEDURE TREATP;
          BEGIN
           IF (GETTRACE:WHICHPROVOKE=2) THEN
            PRINT (" exception:", GETTRACE:EXCEPTPROVOKE);
           IF (GETTRACE:WHICHPROVOKE=3) THEN
            PRINT (" timer:", GETTRACE:TIMERPROVOKE);
          END;
```

SETEXCEPT

NAME

SETEXCEPT - Exception processing description.

SYNTAX

SETEXCEPT:key_word

DESCRIPTION

SETEXCEPT:key_word can only be used with the following key_words :

SETEXCEPT:CANCELTIM	Real time or CPU asynchronous timer cancel.
SETEXCEPT:CONNECT	Exception connection with an asynchronous interruption.
SETEXCEPT:DISCONNECT	Asynchronous interruption release.
SETEXCEPT:HANDLER	Exception processing specifying.
SETEXCEPT:LAUNCHTIM	Real time or CPU asynchronous timer start.
SETEXCEPT:MASK	Masks an exception.
SETEXCEPT:UNMASK	Unmasks an exception.

NAME

SETEXCEPT:CANCELTIMER - Cancels a real time or CPU asynchronous timer.

SYNTAX

SETEXCEPT:CANCELTIMER (*exception*) ;

DESCRIPTION

Cancels an asynchronous timer connected to an exception and launched before using the SETEXCEPT:LAUNCHTIMER procedure.

exception is an argument having the EXCEPTION type.

EVALUATION

During the execution.

WARNING

An error occurs during the execution if the exception is not connected to an asynchronous timer.

SEE ALSO

SETEXCEPT:LAUNCHTIMER

EXAMPLE

```
/DECLARE/ EXCEPTION HORLOGE;
          PROCEDURE ALARME;
          BEGIN
            ...
          END;

/EXEC/ BEGIN
         SETEXCEPT:CONNECT (HORLOGE, "ALARMECLOCK", ALARME);
         SETEXCEPT:LAUNCHTIMER (HORLOGE, 10.0);
         ...
         SETEXCEPT:CANCELTIMER (HORLOGE);
         & Then, the simulation continues without the timer
         ...
       END;
```

SETEXCEPT:CONNECT

NAME

SETEXCEPT:CONNECT - Connection of an exception to an asynchronous signal.

SYNTAX

SETEXCEPT:CONNECT (*exception, signalname*) ;
SETEXCEPT:CONNECT (*exception, signalname, procedure*);

DESCRIPTION

Connects an EXCEPTION object to an asynchronous signal and optionaly defines a procedure to be called when the signal arrives. The asynchronous signal is defined using a symbol (QNAP2 string).

exception is a pointer to the considered EXCEPTION object.

signalname is the symbolic name of the asynchronous signal to be connected to the EXCEPTION objects. It is defined using a QNAP2 string. Blanks located at the beginning and the end of the string are ignored.

procedure defines the user procedure (without argument) to be called when the signal arrives. This procedure will be called, during the simulation, before the next event to be handled (and before the corresponding time progress). All the operations allowed during the simulation may be used (except working demands and blocking of the exception on a synchronization).

EVALUATION

During the execution.

NOTES

This facility is only available if the signal management of QNAP2 has been installed on your machine. The different signals depend on the operating system of your machine.

For example, on UNIX, the signals are :

QNAP2 Name (string)	UNIX Name (UNIX manual)	Description
"INTERRUPT"	SIGINT	control C on keyboard
"QUIT"	SIGQUIT	control / on keyboard
"ALARMCLOCK"	SIGALRM	real time clock
"CTRLZ"	SIGTSTP	control Z on keyboard
"CONTINUE"	SIGCONT	SIGCONT signal on UNIX
"CPUCLOCK"	SIGVTALRM	CPU time clock
"USERIT1"	SIGUSR1	user signal 1, to be sent between processus using KILL command of UNIX
"USERIT2"	SIGUSR2	user signal 2, to be sent between processus using KILL command of UNIX

WARNING

If the procedure defined in argument is no valid then an error occurs during the execution.

If the signal is unknown then an error occurs during the execution.

SEE ALSO

SETEXCEPT:CONNECT - SETEXCEPT:DISCONNECT - SETEXCEPT:MASK - SETEXCEPT:UNMASK - SETEXCEPT:HANDLER - SETEXCEPT:LAUNCHTIMER - SETEXCEPT:CANCELTIMER - EXCEPTION

EXAMPLE

```
/DECLARE/ PROCEDURE HANDLER;
          BEGIN
            PRINT ("AN INTERRUPTION OCCURED");
          END;
          EXCEPTION INTERRUPT; & exception definition

/EXEC/ BEGIN
         SETEXCEPT:CONNECT (INTERRUPT, "INTERRUPT", HANDLER);
         SIMUL;
       END;
```

Prints on UNIX system the "AN INTERRUPTION OCCURED" message if the user types CTRL C during the simulation.

SETEXCEPT:DISCONNECT

NAME

SETEXCEPT:DISCONNECT - Disconnection of an asynchronous exception.

SYNTAX

SETEXCEPT:DISCONNECT (*exception*) ;

DESCRIPTION

Disconnects an EXCEPTION object from the external signal connected before with the SETEXCEPT:CONNECT procedure. After the call of this procedure, each occurence of the signal will be handled according the default specification of the operating system.

exception is a pointer on an EXCEPTION object.

EVALUATION

During the execution.

WARNING

The disconnection of an exception fails (message error) if the exception was not connected before with the SETEXCEPT:CONNECT procedure.

SEE ALSO

SETEXCEPT:CONNECT - SETEXCEPT:DISCONNECT - SETEXCEPT:MASK - SETEXCEP:UNMASK - SETEXCEPT:HANDLER - SETEXCEPT:LAUNCHTIMER - SETEXCEPT:CANCELTIMER - EXCEPTION

EXAMPLE

```
 /DECLARE/ PROCEDURE HANDLER;
           BEGIN
             PRINT ("AN INTERRUPTION OCCURED");
           END;
           EXCEPTION INTERRUPT;

 /EXEC/ BEGIN
          SETEXCEPT:CONNECT (INTERRUPT, "INTERRUPT", HANDLER);
          SIMUL;
& The interruption may be handled
          SETEXCEPT:DISCONNECT (INTERRUPT);
& Now, the interruption cannot be handled
          SIMUL;
        END;
```

NAME

SETEXCEPT:HANDLER - Definition of the treatment in case of exception.

SYNTAX

SETEXCEPT:HANDLER (*exception, procedure*) ;

DESCRIPTION

Defines the procedure to be called in case of exception, without performing other operations. The current treatment procedure of an exception may be read using the DEFHANDLER attribute.

exception defines the EXCEPTION object.

procedure defines the procedure (without argument) performing the treatment of the exception.

EVALUATION

During the execution.

NOTES

This call is the only mean to specify the treatment for the predeclared exceptions : SIMSTART, SIMSTOP, SIMACCUR.

WARNING

When using an exception already connected to an asynchronous signal, the specification of a non valid procedure (or NIL procedure) leads to an error.

SEE ALSO

EXCEPTION - SETEXCEPT:CONNECT - SIMSTART - SIMSTOP - SIMACCUR - DEFHANDLER

EXAMPLE

```
/DECLARE/ PROCEDURE STARTSIM;
          BEGIN               & operations to perform at the
                              & beginning of the simulation
            ...
          END;

/EXEC/ BEGIN
         SETEXCEPT:HANDLER  (SIMSTART,  STARTSIM);
         SIMUL;
       END;
```

SETEXCEPT:LAUNCHTIMER

NAME

SETEXCEPT:LAUNCHTIMER - Real time asynchronous or CPU timer launching.

SYNTAX

SETEXCEPT:LAUNCHTIMER (*exception, real_value*) ;

DESCRIPTION

When connected to an asynchronous signal corresponding to a real time or CPU timer, an exception may support this procedure which initializes the delay before the corresponding signal will be sent.

exception defines the considered EXCEPTION object supposed to be connected to an asynchronous timer signal.

real_value is a real; it represents the delay (real time or CPU time) before the interruption will be sent. This delay must be expressed in seconds. It is taken into account with the best precision given by the clock of the machine.

EVALUATION

During the execution.

NOTES

In case of multiple calls of this procedure, only the last one is taken into account. The effect of this procedure may be cancelled using SETEXCEPT:CANCELTIMER. The treatment of such an exception may be a new call of SETEXCEPT:LAUNCHTIMER.

WARNING

If this procedure is called for an exception which is not connected to an asynchronous signal or connected to a signal which is not a timer, an error occurs during the execution.

SEE ALSO

SETEXCEPT:CANCELTIMER - SETEXCEPT:CONNECT - SETEXCEPT:DISCONNECT

EXAMPLE

```
 /DECLARE/ EXCEPTION HORLOGE;
           PROCEDURE ALARME;
             BEGIN
               PRINT ("ALARM EACH 10 SECONDS");
               SETEXCEPT:LAUNCHTIMER (HORLOGE, 10.0);
             END;

/EXEC/   BEGIN
           SETEXCEPT:CONNECT (HORLOGE, "ALARMCLOCK", ALARME);
           SETEXCEPT:LAUNCHTIMER (HORLOGE, 10.0);
           SIMUL;
         END;
```

SETEXCEPT:MASK

NAME

SETEXCEPT:MASK - Masks an exception.

SYNTAX

SETEXCEPT:MASK (*exception*) ;

DESCRIPTION

Masks the exception given in argument. This exception will keep memorizing and waiting until it is unmasked with SETEXCEPT:UNMASK.

A counter is associated to the exception (initial value is zero). It is incremented by one at each call of SETEXCEPT:MASK. The exception is automatically unmasked when the value of the counter becomes zero after successive calls (if several successive SETEXCEPT:MASK have been done).

exception is a pointer on a EXCEPTION object.

EVALUATION

During the execution.

SEE ALSO

SETEXCEPT:UNMASK

EXAMPLE

```
/DECLARE/ EXCEPTION INTERRUPT;
          PROCEDURE HANDLER;
          BEGIN
            ...
          END;
          QUEUE Q;

/STATION/ NAME = Q;
          SERVICE = BEGIN
                      SETEXCEPT:MASK (INTERRUPT);
& Critical sequence which cannot be interrupted by the INTERRUPT interruption,
& even if the server is temporarily allocated to customers having a higher
& priority
                      SETEXCEPT:UNMASK (INTERRUPT);
& The INTERRUPT interruption may be again handled and will take immediately
& the control if the corresponding signal arrived during the masked critical
& sequence
                    END;
/EXEC/ BEGIN
         SETEXCEPT:CONNECT (INTERRUPT, "INTERRUPT", "HANDLER");
         SIMUL;
       END;
```

SETEXCEPT:UNMASK

NAME

SETEXCEPT:UNMASK - Unmasks an exception.

SYNTAX

SETEXCEPT:UNMASK (*exception*) ;

DESCRIPTION

The counter associated to the exception given in argument is decremented by one. If the value of the counter becomes zero, the unmasked exception will be handled in case of signal waiting.

exception is a pointer on a EXCEPTION object.

EVALUATION

During the execution.

SEE ALSO

SETEXCEPT:MASK

EXAMPLE

```
 /DECLARE/ EXCEPTION INTERRUPT;
           PROCEDURE HANDLER;
           BEGIN
             ...
           END;
           QUEUE Q;

 /STATION/ NAME = Q;
           SERVICE = BEGIN
                       ...
                       SETEXCEPT:MASK (INTERRUPT);
& Critical sequence which cannot be interrupted by the INTERRUPT interruption,
& even if the server is temporarily allocated to customers having a higher
& priority
                       SETEXCEPT:UNMASK (INTERRUPT);
& The INTERRUPT interruption may be again handled and will take immediately
& the control if the corresponding signal arrived during the masked critical
& sequence
                     END;

/EXEC/ BEGIN
         SETEXCEPT:CONNECT (INTERRUPT, "INTERRUPT", "HANDLER");
         ...
         SIMUL;
       END;
```

SETPROFILE

NAME

SETPROFILE - Measurement of the CPU times used by QNAP2 procedures.

SYNTAX

SETPROFILE:*keyword*

DESCRIPTION

Is used to measure automatically the numbers of calls and CPU time used by some or all the user procedures and/or functions in a QNAP2 model. Is also used to improve the algorithmic code of a model by revealing the parts which need to be worked on in order to reduce the computation time taken. Results are also provided on cross references between procedures observed on running, as well as the corresponding performances (number of calls and total CPU time used for a procedure on behalf of a given calling procedure). After measurement, the results are obtained by calling the GETPROFILE:*keyword* function (see this function and the description of the corresponding keywords). Standardized procedures written entirely in QNAP2 language can show and edit the results in full without the need to know the structure of the algorithmic code in advance. An example of such a procedure, which has been used unchanged on several different models, is shown in this manual in the paragraph which describes the GETPROFILE:*keyword* function.

EVALUATION

During the running of an algorithmic code sequence called by an /EXEC/ command, or of a user procedure called in such a sequence. By contrast with comparable measurement tools linked to standard programming languages and under standard operating systems, no special compilation prior to a QNAP2 model is necessary before carrying out measurement.

NOTES

It is perfectly possible to start measuring on a model previously saved to file by a SAVE command, as long as the required commands are added after the RESTORE command. This illustrates the fact that no prior preparation of the procedures to be measured is needed.

WARNING

The measurements cannot be started after the beginning of a simulation or of a mathematical solution, but only before these are begun; in this way, simulation or mathematical solution will be considered as a whole in relation to the measurement of numbers of calls and CPU time used by each procedure.

SEE ALSO

The available calls with keywords are (see each specific description):
SETPROFILE:METERALL - SETPROFILE:METERPROC - SETPROFILE:STARTMETER -
SETPROFILE:STOPMETER - SETPROFILE:CLEAR

See also the GETPROFILE function for obtaining results.

EXAMPLE

```
 ...
/EXEC/ BEGIN
 ...
SETPROFILE:METERALL;       & Asks for all the procedures of the
                           & model to be metered
SETPROFILE:STARTMETER;     & Starts measurement
SIMUL;                     & Launches the simulation which is
                           & to be measured
SETPROFILE:STOPMETER;      & Stops metering before editing
EDITPROF;                  & Edit measurements by means of the
                           & EDITPROF user procedure (see
                           & a GETPROFILE:keyword for an example
                           & of such a procedure).
```

SETPROFILE:CLEAR

NAME

SETPROFILE:CLEAR - Completely resets to zero (after any stop) the system of measuring numbers of calls and CPU time taken for QNAP2 user procedures and/or functions.

SYNTAX

```
SETPROFILE:CLEAR;
```

with no argument at the call

DESCRIPTION

Completely resets the system of measuring numbers of calls and CPU time taken for QNAP2 user procedures and/or functions. After such a reset, any metering that may be in progress is immediately cut off. The metering results are no longer available. Any previous specifications for metering requests on some (or all) user procedures and/or functions are wiped out and can then be completely reformulated, just as after the first startup of a model.

EVALUATION

When the algorithmic code is running without any restriction.

NOTE

This operation is optional if you want to implement several series of measurements on a single model with different sets of parameters. In such a case, a new measurement startup by SETPROFILE:STARTMETER removes the old metering results and and automatically reapplies the previous specifications for user procedures and/or functions to meter.

WARNING

This operation removes all the metering previously done. Do not use instead of SETPROFILE:STOPMETER.

SEE ALSO

SETPROFILE - SETPROFILE:METERALL - SETPROFILE:METERPROC - SETPROFILE:STARTMETER - SETPROFILE:STOPMETER

EXAMPLE

```
 ...
/EXEC/ BEGIN
 ...
SETPROFILE:METERALL;       & Asks for all the procedures of the
                           & model to be metered
SETPROFILE:STARTMETER;     & Start metering
SIMUL;                     & Start the simulation which you
                           & intend to measure
SETPROFILE:STOPMETER;      & Stop metering before editing
EDITPROF;                  & Edition of measurements by means of the
                           & EDITPROF user procedure (see
                           & a GETPROFILE:keyword for an example
                           & of such a procedure).
...
SETPROFILE:CLEAR;          & Reset previous measurements to zero
SETPROFILE:METERALL;
SETPROFILE:STARTMETER;     & New metering
...
```

SETPROFILE:METERALL

NAME

SETPROFILE:METERALL - Asks for metering of CPU time taken on all the user procedures and/or functions of a model.

SYNTAX

SETPROFILE:METERALL;
no argument.

DESCRIPTION

Before starting measurement of CPU time taken on the procedures of the QNAP2 model processed, specifies in advance (by contrast with SETPROFILE:METERPROC which would specify only certain procedures) that all the procedures which will be encountered during running be metered entirely automatically, whatever these procedures may be and even where written by another person and unfamiliar.

EVALUATION

When the algorithmic code is running (/EXEC/ sequence or user procedure called by such a sequence).

NOTE

This operation (or SETPROFILE:METERPROC) must be done before any attempt to start the metering system with SETPROFILE:STARTMETER.

WARNING

Calls to SETPROFILE:METERALL and SETPROFILE:METERPROC are mutually exclusive, except for an intermediate call to SETPROFILE:CLEAR which performs a general reset of the metering system.

SEE ALSO

SETPROFILE - SETPROFILE:METERPROC - SETPROFILE:STARTMETER - SETPROFILE:STOPMETER - SETPROFILE:CLEAR

EXAMPLE

```
 ...
/EXEC / BEGIN
 ...
SETPROFILE:METERALL;        & Asks for all the procedures of the
                            & model to be metered
SETPROFILE:STARTMETER;      & Start metering
SIMUL;                      & Start the simulation which you
                            & intend to measure
SETPROFILE:STOPMETER;       & Stop metering before editing
EDITPROF;                   & Edition of measurements by means of the
                            & EDITPROF user procedure (see
                            & a GETPROFILE:keyword for an example
                            & of such a procedure).
```

SETPROFILE:METERPROC

NAME

SETPROFILE:METERPROC - Asks for metering of CPU times taken on certain procedures of a QNAP2 model only.

SYNTAX

SETPROFILE:METERPROC (*procedure*);
where the argument represents a procedure or function (in this case, use the user directive ADDRESS). A procedure or function pointer is allowed, as is a character string which gives the name of the procedure or function.

DESCRIPTION

Specifies that the user procedure or function used as an argument must be metered. For several different user procedures or functions, make as many calls as there are procedures or functions.

EVALUATION

When the algorithmic code is running (/EXEC/ sequence or user procedure called by such a sequence).

NOTE

This operation (or SETPROFILE:METERALL) must be performed before attempting to start the metering system with SETPROFILE:STARTMETER.

WARNING

Calls to SETPROFILE:METERALL and SETPROFILE:METERPROC are mutually exclusive, except for an intermediate call to SETPROFILE:CLEAR which performs a general reset of the metering system.

SEE ALSO

SETPROFILE - SETPROFILE:METERALL - SETPROFILE:STARTMETER - SETPROFILE:STOPMETER - SETPROFILE:CLEAR

EXAMPLE

```
/DECLARE/
PROCEDURE FOO (X,Y,Z);
 ...
BEGIN
 ...
END;

INTEGER FUNCTION BAR;
 ...
BEGIN
 ...
END;
 ...
/EXEC/ BEGIN
 ...
SETPROFILE:METERPROC (FOO);
SETPROFILE:METERPROC (BAR ADDRESS);
SETPROFILE:STARTMETER;
SIMUL;
SETPROFILE:STOPMETER;
 ...
END;
```

The following form also allowed:

```
/EXEC/ BEGIN
 ...
SETPROFILE:METERPROC ("FOO");
SETPROFILE:METERPROC ("BAR");
SETPROFILE:STARTMETER;
SIMUL;
SETPROFILE:STOPMETER;
 ...
END;
```

by specifying the procedures by their name as character strings.

SETPROFILE:STARTMETER

NAME

SETPROFILE:STARTMETER - Starts metering CPU time taken (and number of calls) on the specified procedures (perhaps all if SETPROFILE:METERALL was previously called).

SYNTAX

SETPROFILE:STARTMETER; with no argument at the call

DESCRIPTION

Starts metering the numbers of calls and CPU time taken on the user procedures or functions (perhaps all if SETPROFILE:METERALL was previously called) previously specified.

EVALUATION

When the algorithmic code is running without any restriction. (/EXEC/ sequence or user procedure called by such a sequence).

NOTE

This operation must be carried out after the user procedures and/or functions to be metered have been specified by SETPROFILE:METERALL or SETPROFILE:METERPROC.

WARNING

The metering system must be started before the start of the simulation or the mathematical solution so that metering can be done on such a simulation or solution; otherwise, it is impossible to start metering during running in such cases.

SEE ALSO

SETPROFILE - SETPROFILE:METERALL - SETPROFILE:METERPROC - SETPROFILE:STOPMETER - SETPROFILE:CLEAR

EXAMPLE

```
 ...
/EXEC / BEGIN
 ...
SETPROFILE:METERALL;       & Asks for all the procedures of the
                           & model to be metered
SETPROFILE:STARTMETER;     & Start metering
SIMUL;                     & Start the simulation which you
                           & intend to measure
SETPROFILE:STOPMETER;      & Stop metering before editing
EDITPROF;                  & Edition of measurements by means of the
                           & EDITPROF user procedure (see
                           & a GETPROFILE:keyword for an example
                           & of such a procedure).
...
SETPROFILE:CLEAR;          & Reset previous measurements to zero
SETPROFILE:METERALL;
SETPROFILE:STARTMETER;     & New metering
```

SETPROFILE:STOPMETER

NAME

SETPROFILE:STOPMETER - Stops metering in progress of numbers of calls and CPU time taken by the procedures of the model, prior to any attempt to get the results.

SYNTAX

SETPROFILE:STOPMETER;
with no argument at the call

DESCRIPTION

Stops the metering of the numbers of calls and CPU time taken previously started by SETPROFILE:STARTMETER on the user procedures or functions specified. The metering results can then be accessed using the appropriate access functions (see GETPROFILE:*keyword*).

EVALUATION

When the algorithmic code is running.

NOTE

This operation must be carried out before the result of metering can be accessed.

SEE ALSO

SETPROFILE - SETPROFILE:METERALL - SETPROFILE:METERPROC - SETPROFILE:STARTMETER - SETPROFILE:CLEAR - GETPROFILE

EXAMPLE

```
 ...
/EXEC/ BEGIN
 ...
SETPROFILE:METERALL;       & Asks for all the procedures of the
                           & model to be metered
SETPROFILE:STARTMETER;     & Start metering
SIMUL;                     & Start the simulation which you
                           & intend to measure
SETPROFILE:STOPMETER;      & Stop metering before editing
EDITPROF;                  & Edition of measurements by means of the
                           & EDITPROF user procedure (see
                           & a GETPROFILE:keyword for an example
                           & of such a procedure).
...
```

NAME

`SETSTAT` - Description of the searched statistical results on a variable.

SYNTAX

`SETSTAT:key-word`

DESCRIPTION

`SETSTAT:key-word` may be used only with the following key-words :

SETSTAT:ACCURACY	Accuracy computing request.
SETSTAT:BLOCKED:MEAN	Statistical results request on the blocking time of queue.
SETSTAT:BUSYPCT:MEAN	Statistical results request on the busy percentage of queue servers.
SETSTAT:CANCEL	Cancels the requested statistical results.
SETSTAT:CLASS	Per class standard statistical results request for one or several queues.
SETSTAT:CONTINUE	Defines as continuous a variable on WATCHED declared.
SETSTAT:CORRELATION	Computing auto-correlation coefficients request.
SETSTAT:CUSTNB:MEAN	Statistical results request on the number of customers in a queue.
SETSTAT:DISCRETE	Defines as discrete a variable on WATCHED declared.
SETSTAT:MARGINAL	Computing marginal probabilities request.
SETSTAT:OFF	Stops taking measures into account.
SETSTAT:ON	Takes measures into account.
SETSTAT:PARTIAL	Partial results request.
SETSTAT:PRECISION	Request simulation stop as soon as the mean of the concerned variable has reached a precision.
SETSTAT:QUEUE	Standard statistical results request for a queue.
SETSTAT:RESPONSE:MEAN	Request of statistical results on the time queue response.
SETSTAT: SAMPLE	Generates computing of all basical statistical criteria concerning variable, mean, variance, maximum and minimum.
SETSTAT:SERVICE:MEAN	Statistical results on the service time of a queue request.
SETSTAT:THRUPUT:MEAN	Statistical results on the throughput of a queue request.

NOTES

The results are available by means of the `GETSTAT:key-word` functions.

WARNING

This procedure can only be called in an algorithmic sequence before the resolution (not possible in a service of a station).

This procedure can only be used for simulation.

SETSTAT:ACCURACY

NAME

SETSTAT:ACCURACY - Accuracy computing request.

SYNTAX

SETSTAT:ACCURACY *(list_of_variables)* ;

DESCRIPTION

Specifies that an accuracy computing is requested on the variables means.

The variables can be :
- integers or reals declared in WATCHED
- queues
- queues followed by a list of classes.

Use SETSTAT:result:ACCURACY to compute an accuracy on a statistical result for a queue.

Results are available by means of the following functions :
GETSTAT:ACCURACY (*variable*) ; or GETSTAT:result:ACCURACY (*queue*, [*class*]) ; .

EVALUATION

At the beginning of the resolution.

NOTES

The estimation method is common to all statistical variables. It is defined by means of the ESTIMATION parameter.

If the estimation method is not the spectral one, periodical results can be obtained , by means of SETSTAT:PARTIAL (list of variables) ;

WARNING

This procedure must be preceded by one of the following procedures :
- For a queue : SETSTAT:QUEUE *(queue)* ;
- For a integer or real : SETSTAT:CONTINUE (*liste_of_variables*) or SETSTAT:DISCRETE (*liste_of_variables*)
- For a (queue,class) couple : SETSTAT:CLASS (*queue, class*);

This procedure can only be called in an algorithmic sequence before the resolution (not possible in a service of a station).

This procedure can only be used for simulation.

SEE ALSO

GETSTAT:ACCURACY - WATCHED - SETSTAT:PARTIAL - SETSTAT:CORRELATION - SETSTAT:PRECISION - SETSTAT:CLASS - SETSTAT:QUEUE - SETSTAT:SERVICE - SETSTAT:RESPONSE - SETSTAT:BLOCKED - SETSTAT:CUSTNB - SETSTAT:BUSYPCT - SETSTAT:THRUPUT

EXAMPLE

```
/DECLARE/ QUEUE q;
          CLASS c1, c2;

/CONTROL/ TMAX = 10000;
          PERIOD = 2000;
          ESTIMATION = REGENERATION;
          TEST = BEGIN
                   SAMPLE;
                   result := GETSTAT:SERVICE:MEAN (q, c1);
                 END;

/EXEC/ BEGIN
         SETSTAT:QUEUE (q, c1); & request of statistical results
         SETSTAT:ACCURACY (q, c1);
         ....
         SIMUL;
         PRINT (GETSTAT:RESPONSE:ACCURACY (q, c1));
         ....
       END;
```

SETSTAT:BLOCKED:MEAN

NAME

SETSTAT:BLOCKED:MEAN - Statistical results on the blocking time of queue request.

SYNTAX

SETSTAT:BLOCKED:MEAN (*list_of_queues*, *list_of_classes*) ;

DESCRIPTION

Constitutes an explicit demand of standard statistical results : mean, variance, minimum and maximum on the blocking time for each specified (*queue*, *class*) couple.

These results are avalaible by means of the functions:
GETSTAT:BLOCKED:MEAN
GETSTAT:BLOCKED:MAXIMUM
GETSTAT:BLOCKED:MINIMUM
GETSTAT:BLOCKED:VARIANCE

More complex results can be requested by means of the following procedures:
SETSTAT:BLOCKED:ACCURACY for an accuracy.
SETSTAT:BLOCKED:PRECISION for a stop on a fixed precision.
SETSTAT:BLOCKED:MARGINAL for marginal probabilities.
SETSTAT:BLOCKED:CORRELATION for auto-correlation coefficients.
Periodical results can be computed by calling the SETSTAT:PARTIAL procedure.

EVALUATION

At the beginning of the resolution.

WARNING

No more complex statistical result on the service time can be requested if not preceded by SETSTAT:BLOCKED:MEAN (*queue*) or SETSTAT:QUEUE (*queue*).

This procedure can only be called in an algorithmic sequence before the resolution (not possible in a service of a station).

This procedure has got a sens only for simulation.

SEE ALSO

SETSTAT:QUEUE - SETSTAT:CLASS - GETSTAT:BLOCKED:MEAN - SETSTAT:PARTIAL - SETSTAT:ACCURACY - SETSTAT:MARGINAL - SETSTAT:CORRELAT - SETSTAT:PRECISION - SETSTAT:CANCEL

EXAMPLE

```
/DECLARE/ QUEUE q;
          CLASS c1, c2;
          .....

/EXEC/ BEGIN
         SETSTAT:BLOCKED:MEAN (q);
         SETSTAT:BLOCKED:MEAN (q, c1);
         SETSTAT:BLOCKED:ACCURACY (q, c1);
         ....
         SIMUL;
         ....
       END;
```

SETSTAT:BUSYPCT:MEAN

NAME

SETSTAT:BUSYPCT:MEAN - Statistical results request on the occupation rate of servers of a queue.

SYNTAX

SETSTAT:BUSYPCT:MEAN (*list_of_queues*, *list_of_classes*) ;

DESCRIPTION

Constitutes an explicit request of standard statistical results : mean, variance, minimum and maximum on the busy percentage for each specified (*queue*, *class*) couple.

These results are available by means of the following functions :
GETSTAT:BUSYPCT:MEAN
GETSTAT:BUSYPCT:MAXIMUM
GETSTAT:BUSYPCT:MINIMUM
GETSTAT:BUSYPCT:VARIANCE

More complex results may be requested by the following procedures :
SETSTAT:BUSYPCT:ACCURACY for an accuracy.
SETSTAT:BUSYPCT:PRECISION for a stop on a fixed precision.
SETSTAT:BUSYPCT:MARGINAL for marginal probabilities.
SETSTAT:BUSYPCT:CORRELATION for auto-correlation coefficients.
Periodical results may be computed in calling the SETSTAT:PARTIAL procedure.

EVALUATION

At the beginning of the resolution.

WARNING

No more complex statistical result can be requested on the service time if not preceded by SETSTAT:BUSYPCT:MEAN (*queue*) or SETSTAT:QUEUE (*queue*).

This procedure can only be called in an algorithmic sequence before the resolution (not possible in a service of a station).

This procedure has got a sens only for simulation.

SEE ALSO

SETSTAT:QUEUE - SETSTAT:CLASS - GETSTAT:BUSYPCT:MEAN - SETSTAT:PARTIAL - SETSTAT:ACCURACY - SETSTAT:MARGINAL - SETSTAT:CORRELAT - SETSTAT:PRECISION - SETSTAT:CANCEL

EXAMPLE

```
/DECLARE/ QUEUE q;
          CLASS c1, c2;
          .....

/EXEC/ BEGIN
         SETSTAT:BUSYPCT:MEAN (q );
         SETSTAT:BUSYPCT:MEAN (q, c1);
         SETSTAT:BUSYPCT:ACCURACY (q, c1);
         ....
         SIMUL;
         ....
       END;
```

SETSTAT:CANCEL

NAME

SETSTAT:CANCEL - Cancels the request of statistical results.

SYNTAX

SETSTAT:CANCEL (*list_of_variables*) ;

DESCRIPTION

Cancels all requested statistical results.

Variables types can be :
- integer or real (WATCHED declared)
- queue
- queue followed by a list of classes.

A SETSTAT:SERVICE:CANCEL procedure cancels only the specified standard variables.

EVALUATION

At the beginning of the resolution.

WARNING

This procedure inhibits the statistics requested previously by procedures such as :
- For a queue : SETSTAT:QUEUE (*queue*) ;
- For an integer or real : SETSTAT:CONTINUE (*list_of_variables*) or SETSTAT:DISCRETE (*list_of_variables*)
- For a (*queue,class*) couple : SETSTAT:CLASS (*queue, class*) ;

This procedure can only be called in an algorithmic sequence before the resolution (not possible in a service of a station).

This procedure can only be used for simulation.

SEE ALSO

SETSTAT:DISCRETE - SETSTAT:CONTINUE - SETSTAT:CLASS - SETSTAT:QUEUE - SETSTAT:SERVICE - SETSTAT:RESPONSE - SETSTAT:BLOCKED - SETSTAT:CUSTNB - SETSTAT:BUSYPCT

EXAMPLE

```
/DECLARE/ QUEUE q;
          CLASS c1, c2;
          .....

/CONTROL/ TMAX = 10000;

/EXEC/ BEGIN
         SETSTAT:QUEUE (q, c1); & statistical results request
         ....
         SIMUL;
         SETSTAT:RESPONSE:CANCEL (q, c1); & cancels the request for the
         ....                             & response time
       END;

/CONTROL/ ESTIMATION = SPECTRAL;

/EXEC/ BEGIN
         SETSTAT:QUEUE (q, c2);
         SETSTAT:ACCURACY (q, c2);
       END;
```

SETSTAT:CLASS

NAME

SETSTAT:CLASS - Standard statistical results request per class for one or several queues.

SYNTAX

SETSTAT:CLASS (*list_of_queues*, *list_of_classes*) ;

DESCRIPTION

Constitutes an explicit request of standard statistical results per class for the specified queues. Then, it is possible to get the mean, variance, minimum and maximum value on the service time, response time, blocking time, servers occupation rate and mean number of customers.

These results are available by means of the following functions :
GETSTAT:SERVICE: ...
GETSTAT:RESPONSE: ...
GETSTAT:BLOCKED: ...
GETSTAT:BUSYPCT: ...
GETSTAT:CUSTNB:MEAN
GETSTAT:CUSTNB:MAXIMUM
GETSTAT:CUSTNB:MINIMUM
GETSTAT:CUSTNB:VARIANCE

More complex results may be requested by the following procedures :
SETSTAT:ACCURACY for an accuracy.
SETSTAT:PRECISION for a stop on a fixed precision.
SETSTAT:MARGINAL for marginal probabilities.
SETSTAT:CORRELATION for auto-correlation coefficients

EVALUATION

At the beginning of the resolution.

WARNING

This procedure can only be called in an algorithmic sequence before the resolution (not possible in a service of a station).

This procedure has got a sens only for simulation.

SEE ALSO

SETSTAT:QUEUE - SETSTAT:SERVICE:MEAN - SETSTAT:RESPONSE:MEAN - SETSTAT:BLOCKED:MEAN - SETSTAT:BUSYPCT:MEAN - GETSTAT:SERVICE:MEAN - GETSTAT:RESPONSE:MEAN - GETSTAT:BLOCKED:MEAN - GETSTAT:BUSYPCT:MEAN - GETSTAT:CUS NB: MEAN - SETSTAT:CANCEL

EXAMPLE

```
/DECLARE/ QUEUE q1,q2;
          CLASS c1, c2;
          .....

/EXEC/ BEGIN
         SETSTAT:CLASS  (ALL QUEUE, c1);
         ....
         SIMUL;
         PRINT (GETSTAT:CUSTNB:MAXIMUM (q1, c1));
         ....
       END;
```

SETSTAT:CONTINUE

NAME

SETSTAT:CONTINUE - Defines as continuous a WATCHED declared variable.

SYNTAX

SETSTAT:CONTINUE (*list_of_variables*) ;

DESCRIPTION

This procedure requests simple statistics on the variable : mean, variance, minimum and maximum.

It also gives access to more complex statistics : accuracy or stop on a fixed precision.

Variables are integers or reals, global variables, attribute of object or elements of WATCHED declared array.

The statistical results can then be got using the following functions :
GETSTAT:MEAN for the mean
GETSTAT:VARIANCE for the variance
GETSTAT:MINIMUM for the minimum value
GETSTAT:MAXIMUM for the maximum value.

More complex results can be requested by means of the procedures :
SETSTAT:ACCURACY for an accuracy.
SETSTAT:PRECISION for a stop on a fixed precision.
SETSTAT:MARGINAL for marginal probabilities.
SETSTAT:CORRELATION for auto-correlation coefficients.

EVALUATION

At the beginning of the resolution.

NOTES

A variable is continuous if the delay when the same value is kept must be taken into account.
The mean is equal to :

$$\sum_{i=1}^{i=N} \frac{x_i t_i}{T}$$

with
- T represents the observation time.
- x_i the values taken by the observed variable.
- t_i the delay where the variable keeps the value .

Such variables are also called integrators. In the case of QNAP2 standard results, the mean number of customers in a station and the occupation rate of the servers are continuous variables.

WARNING

This procedure can only be called in an algorithmic sequence before the resolution (not possible in a service of a station).

This procedure can only be used for simulation.

SEE ALSO

WATCHED - SETSTAT:CONTINUE - SETSTAT:DISCRETE - SETSTAT:PARTIAL - SETSTAT:CORELATION - SETSTAT:PRECISION - GETSTAT:MEAN - GETSTAT:VARIANCE - GETSTAT:MINIMUM - GETSTAT:MAXIMUM - SETSTAT:CANCEL

EXAMPLE

```
/DECLARE/ WATCHED INTEGER I;

/EXEC/ BEGIN
         SETSTAT:CONTINUE (I) ;
       END ;
```

SETSTAT:CORRELATION

NAME

SETSTAT:CORRELATION - Auto-correlation coefficients computing request.

SYNTAX

SETSTAT:CORRELATION *(list_of_variables, order)* ;

DESCRIPTION

It provides the auto-correlation coefficients up to requested order, which must be less or equal to 20. This functionality can only be used with the regeneration method.

order is an integer greater than 0.

Variables may be of the following type :
- integer or real (WATCHED declared)
- queue
- queue followed of a list of classes.

To compute the auto-correlation coefficients on objects relative to a queue, use SETSTAT:result:CORRELATION.

Results are avalaible by means of the function : GETSTAT:CORRELATION (*variable, order*); or GETSTAT:result:CORRELATION (*queue*, [*class* ,] *order*) ;.

EVALUATION

At the beginning of the resolution.

WARNING

- This procedure can only be called in an algorithmic sequence before the resolution (not possible in a service of a station).
- This procedure can only be used for simulation.
- This procedure is identified only if the regeneration method is used.
- The results are available only at the end of the simulation.
- This procedure must be preceded by the SETSTAT:ACCURACY procedure

SEE ALSO

GETSTAT:CORRELATION - WATCHED - SETSTAT:CLASS - SETSTAT:QUEUE - SETSTAT:ACCURACY - SETSTAT:SERVICE - SETSTAT:RESPONSE - SETSTAT:BLOCKED - SETSTAT:CUSTNB - SETSTAT:BUSYPCT - ESTIMATION - SAMPLE

EXAMPLE

```
/DECLARE/ QUEUE q;
          CLASS c1, c2;
          .....

/CONTROL/ TMAX = 10000;
          PERIOD = 2000;
          ESTIMATION = REGENERATION;
          TEST = BEGIN
                   SAMPLE;
                 END;

/EXEC/ BEGIN
         SETSTAT:QUEUE (q, c1); & request of statistical results
         SETSTAT:ACCURACY (q, c1);
         SETSTAT:RESPONSE:CORRELATION (q, c1, 5);
         ....
         SIMUL;
         PRINT (GETSTAT:RESPONSE:CORRELATION (q, c1, 2));
         ....
       END;
```

SETSTAT:CUSTNB:MEAN

NAME

SETSTAT:CUSTNB:MEAN - Statistical results request on the number of customers in a queue.

SYNTAX

SETSTAT:CUSTNB:MEAN (*list_of_queues*, *list_of_classes*) ;

DESCRIPTION

Constitutes an explicit request of standard statistical results: mean, variance, minimum and maximum on the number of customers for each specified (*queue*, *class*) couple.

These results are available by means of the following functions :
GETSTAT:CUSTNB:MEAN
GETSTAT:CUSTNB:MAXIMUM
GETSTAT:CUSTNB:MINIMUM
GETSTAT:CUSTNB:VARIANCE

More complex results may be requested by means of the following procedures :
SETSTAT:CUSTNB:ACCURACY for an accuracy.
SETSTAT:CUSTNB:PRECISION for a stop on a fixed precision.
SETSTAT:CUSTNB:MARGINAL for marginal probabilities.
SETSTAT:CUSTNB:CORRELATION for the auto-correlation coefficients

Periodical results may be computed in calling the SETSTAT:PARTIAL procedure.

EVALUATION

At the beginning of the resolution.

WARNING

No more complex result can be requested on the service time if not preceded by SETSTAT:CUSTNB:MEAN (*queue*) or SETSTAT:QUEUE (*queue*).

This procedure can only be called in an algorithmic sequence before the resolution (not possible in a service of a station).

This procedure has got a sens only for simulation.

SEE ALSO

SETSTAT:QUEUE - SETSTAT:CLASS - GETSTAT:CUSTNB:MEAN - SETSTAT:PARTIAL - SETSTAT:ACCURACY - SETSTAT:MARGINAL - SETSTAT:CORRELAT - SETSTAT:PRECISION - SETSTAT:CANCEL

EXAMPLE

```
/DECLARE/ QUEUE q;
          CLASS c1, c2;
          .....

/EXEC/ BEGIN
         SETSTAT:CUSTNB:MEAN (q );
         SETSTAT:CUSTNB:MEAN (q, c1);
         SETSTAT:CUSTNB:ACCURACY (q, c1);
         ....
         SIMUL;
         ....
       END;
```

SETSTAT:DISCRETE

NAME

SETSTAT:DISCRETE - Defines as discrete a WATCHED declared variable.

SYNTAX

SETSTAT:DISCRETE (*list_of_variables*) ;

DESCRIPTION

This procedure requests simple statistics on the variable: mean, variance, minimum and maximum. It also gives access to more complex statistics: confidence intervals or stop on fixed precision.

The variables are integers or reals, global variables, attributes of objets or elements of a WATCHED declared array.

The statistical results can then be got using the following functions :
GETSTAT:MEAN for the mean
GETSTAT:VARIANCE for the variance
GETSTAT:MINIMUM for the minimum value
GETSTAT:MAXIMUM for the maximum value

More complex results may be requested by means of the following procedures :
SETSTAT:ACCURACY for an accuracy.
SETSTAT:PRECISION for a stop on a fixed precision.
SETSTAT:MARGINAL for marginal probabilities.
SETSTAT:CORRELATION for auto-correlation coefficients

EVALUATION

At the beginning of the resolution.

NOTES

A variable is discrete if any value, taken by the variable, enters in the account of the sample independently of the delay when the variable keeps the same value.
The mean in that case is :

$$\sum_{i=1}^{i=N} \frac{x_i}{N}$$

- x_i are the values taken by the variable.
- N is the number of values taken.

Among the standard results of QNAP2, the discrete statistical variables are the response, the service and the blocking times.

WARNING

This procedure can only be called in an algorithmic sequence before the resolution (not possible in a service of a station).

This procedure can only be used for simulation.

SEE ALSO

WATCHED - SETSTAT:CONTINUE - SETSTAT:DISCRETE - SETSTAT:PARTIAL - SETSTAT:CORRELATION - SETSTAT:PRECISION - GETSTAT:MEAN - GETSTAT:VARIANCE - GETSTAT:MINIMUM - GETSTAT:MAXIMUM - SETSTAT:CANCEL

EXAMPLE

```
/DECLARE/ WATCHED INTEGER I;

/EXEC/ BEGIN
         SETSTAT:DISCRETE (I) ;
       END ;
```

SETSTAT:MARGINAL

NAME

SETSTAT:MARGINAL - Computation of marginal probabilities request.

SYNTAX

SETSTAT:MARGINAL *(list_of_variables, integer, value, size)* ;

DESCRIPTION

A marginal probability returns the probability that a variable belongs to a specified by the user interval.

This facility allows especially to build histograms.

Variables can take one of the following types :
- integer or real (WATCHED declared)
- queue
- queue followed by a list of classes.

The other parameters specify the limits and the size of the intervals :
integer : number of intervals
value : lower limit of the intervals
size : size of an interval

Probabilities are also computed to be out of the interval by lower or upper bound.

To compute marginal probabilities on objects relative to a queue , use SETSTAT:result:MARGINAL .

Results are available by means of the functions : GETSTAT:MARGINAL (*variable, order*) ; or GETSTAT:result:MARGINAL (*queue*, [*class*], *order*) ;.

EVALUATION

At the beginning of the resolution.

NOTES

The SETSTAT:MARGINAL (*file*,....) procedure computes the marginal probability on the mean number of customers in the station.

WARNING

This procedure must be preceded by one of the following procedures :
- For a queue : SETSTAT:QUEUE (*queue*) ;
- For an integer or real variable : SETSTAT:CONTINUE (*list_of_variables*) or SETSTAT:DISCRETE *(list_of_variables)*
- For a (*queue, class*) couple : SETSTAT:CLASS (*queue, class*);

This procedure can only be called in an algorithmic sequence before the resolution (not possible in a service of a station).

This procedure can only be used for simulation.

SEE ALSO

GETSTAT:MARGINAL - WATCHED - SETSTAT:CLASS - SETSTAT:QUEUE - SETSTAT:SERVICE - SETSTAT:RESPONSE - SETSTAT:BLOCKED - SETSTAT:CUSTNB - SETSTAT:BUSYPCT

EXAMPLE

```
 /DECLARE/ QUEUE q;
           .....

 /CONTROL/ TMAX = 10000;

 /EXEC/ BEGIN
& request of statistical results
           SETSTAT:QUEUE (q);
           SETSTAT:RESPONSE:MARGINAL (q, 5, 1., 1.5);
           ....
           SIMUL;
& probability to be in the fourth interval
           PRINT (GETSTAT:RESPONSE:MARGINAL(q ,4)) ;
           ....
         END;
```

SETSTAT:OFF

NAME

SETSTAT:OFF - Commands the interruption of the measurements.

SYNTAX

SETSTAT:OFF (*list_of_variables*) ;

DESCRIPTION

Defines the end or the interruption of the sample.

Variables may be one of the following types :
- integer or real (WATCHED declared)
- queue
- queue followed by a list of classes.

EVALUATION

At the beginning of the resolution.

NOTES

By default, the measurements are active from the beginning of the simulation (at the end of the TSTART period) and remain active until the end of the simulation.

By default, when a TSTART period is defined, the measurements become active at the end of the TSTART period.

WARNING

This procedure can only be called in an algorithmic sequence before the resolution (not possible in a service of a station).

This procedure can only be used for simulation.

SEE ALSO

WATCHED - SETSTAT:CLASS - SETSTAT:QUEUE - SETSTAT:SERVICE - SETSTAT:RESPONSE - SETSTAT:BLOCKED - SETSTAT:CUSTNB - SETSTAT:BUSYPCT - TSTART - SETSTAT:ON

EXAMPLE

```
/DECLARE/ WATCHED INTEGER I;

/EXEC/ BEGIN
         SETSTAT:DISCRETE (I);
         SETSTAT:OFF (I);
       END ;
```

NAME

SETSTAT:ON - Commands the measurements.

SYNTAX

SETSTAT:ON (*list_of_variables*) ;

DESCRIPTION

Defines the beginning or the rerun of the sample.

Variables may be one of the following types :
- integer or real (WATCHED declared)
- queue
- queue followed by a list of classes.

EVALUATION

At the beginning of the resolution.

NOTES

By default, the measurements are active from the beginning of the simulation (at the end of the TSTART period) and remain active until the end of the simulation.

WARNING

This procedure can only be called in an algorithmic sequence before the resolution (not possible in a service of a station).

This procedure can only be used for simulation.

SEE ALSO

WATCHED - SETSTAT:CLASS - SETSTAT:QUEUE - SETSTAT:SERVICE - SETSTAT:RESPONSE - SETSTAT:BLOCKED - SETSTAT:CUSTNB - SETSTAT:BUSYPCT - TSTART - SETSTAT:OFF

EXAMPLE

```
 /DECLARE/ WATCHED INTEGER I;

/EXEC/ BEGIN
         SETSTAT:DISCRETE (I);
         SETSTAT:ON (I);
       END ;
```

SETSTAT:PARTIAL

NAME

SETSTAT:PARTIAL - Partial results request.

SYNTAX

SETSTAT:PARTIAL (*list_of_variables*) ;

DESCRIPTION

The notion of partial results is the opposite of the notion of global results. Global results take into account all the simulation from the beginning of the measurement (beginning of the simulation or end of the TSTART period).
Partial results only concern the last period (remember that a period is specified by the PERIOD key word or by a call to the SETSTAT:SAMPLE (*variable*) procedure).

Variables may be one of the following types :
- integer or real (WATCHED declared)
- queue
- queue followed by a list of classes.

To compute a queue partial statistic, use
SETSTAT:result:PARTIAL.

Results are available by means of following types of functions : GETSTAT:MEAN (*variable*) ; or GETSTAT:result:MAXIMUM (*queue*, [*class*]) ;.

Statistical results returned during the simulation concern the last period. On the other hand, at the end of the simulation, the results are global.

EVALUATION

At the beginning of the resolution.

NOTES

By default, the periodical results computed are global, ie they take into account all the values of a variable from the beginning of the simulation at the end of the TSTART period.

WARNING

This procedure must be preceded by one of th following procedures :
- For a queue : SETSTAT:QUEUE (queue) ;
- For an integer or real variable : SETSTAT:CONTINUE (*list_of_variables*) or SETSTAT:DISCRETE *(list_of_variables)*
- For a (*queue, class*) couple : SETSTAT:CLASS (*queue, class*) ;

This procedure can only be called in an algorithmic sequence before the resolution (not possible in a service of a station).

This procedure can only be used for simulation.

SEE ALSO

GETSTAT:MEAN - WATCHED - SETSTAT:PARTIAL - SETSTAT:CORRELAT - SETSTAT:PRECISION - SETSTAT:CLASS - SETSTAT:QUEUE - SETSTAT:SERVICE - SETSTAT:RESPONSE - SETSTAT:BLOCKED - SETSTAT:CUSTNB - SETSTAT:BUSYPCT - PERIOD - ESTIMATION

EXAMPLE

```
 /DECLARE/ QUEUE q;
           CLASS c1, c2;
           REAL result;

/CONTROL/ TMAX = 10000;
          PERIOD = 2000;
          TEST = BEGIN
                   SAMPLE;
                   result := GETSTAT:SERVICE:MEAN (q, c1);
                 END;

 /EXEC/ BEGIN
          SETSTAT:QUEUE (q); & request of statistical results
          SETSTAT:PARTIAL (q, c1);
          ....
          SIMUL;
          PRINT (GETSTAT:RESPONSE:ACCURACY (q, c1));
          ....
        END;
```

SETSTAT:PRECISION

NAME

SETSTAT:PRECISION - Requests the end of the simulation as soon as the mean of the concerned variable has reached a fixed precision.

SYNTAX

SETSTAT:PRECISION (*list_of_variables*, *relative_precision*) ;

DESCRIPTION

This procedure allows the end of the simulation as soon as the means of the concerned variables have reached a given precision.

The precision is defined by a real *relative_precision* between 0.05 and 1, which represents the admissible size for the accuracy.

If the simulation is stopped before the time defined by TMAX, the size of each accuracy will then be lower or equal to the precision multiplied by the current mean of the variable.

Variables can be one of the following types :
- integer or real (WATCHED) declared
- queue
- queue followed by a list of classes.

EVALUATION

At the beginning of the resolution.

NOTES

A simulation may be stopped by a STOP procedure or when the TMAX, maximum simulation run length specified by the user, is reached.

To request a stop of the simulation on a fixed precision of objects concerning a queue, use SETSTAT:result:PRECISION.

WARNING

This procedure must be preceded by the SETSTAT:ACCURACY procedure.

This procedure can only be called in an algorithmic sequence before the resolution (not possible in a service of a station).

This procedure can only be used for simulation.

SEE ALSO

WATCHED - SETSTAT:CLASS - SETSTAT:QUEUE - SETSTAT:SERVICE - SETSTAT:RESPONSE - SETSTAT:BLOCKED - SETSTAT:CUSTNB - SETSTAT:BUSYPCT - ESTIMATION - STOP - ABORT

EXAMPLE

```
/DECLARE/ QUEUE q;
          CLASS c1, c2;
          .....

/CONTROL/ TMAX = 10000;
          ESTIMATION = SPECTRAL;

/EXEC/ BEGIN
         SETSTAT:QUEUE (q, c1); & request of statistical results
         SETSTAT:ACCURACY (q, c1);
         SETSTAT:PRECISION (q, c1, 0.2);
         ....
         SIMUL;
         PRINT (GETSTAT:RESPONSE:ACCURACY (q, c1));
         ....
       END;
```

SETSTAT:QUEUE

NAME

SETSTAT:QUEUE - Standard statistical results on a queue request.

SYNTAX

SETSTAT:QUEUE (*list_of_queues*) ;

DESCRIPTION

Constitutes an explicit request on standard statistical results on the specified queues.
It is then possible to compute the mean, the variance, the minimum and the maximum on the service time, response time, blocking time, busy percentage of the servers, the number of customers and the throughput of the queue.

The results are available by means of the following functions :
GETSTAT:SERVICE: ...
GETSTAT:RESPONSE: ...
GETSTAT:BLOCKED: ...
GETSTAT:BUSYPCT: ...
GETSTAT:CUSTNB:MEAN
GETSTAT:CUSTNB:MAXIMUM
GETSTAT:CUSTNB:MINIMUM
GETSTAT:CUSTNB:VARIANCE
GETSTAT:THRUPUT:MEAN

More complex results may be requested by means of the following procedures :
SETSTAT:ACCURACY for an accuracy.
SETSTAT:PRECISION for a stop on a fixed precision.
SETSTAT:MARGINAL for marginal probabilities.
SETSTAT:CORRELATION for auto-correlation coefficients

EVALUATION

At the beginning of the resolution.

WARNING

This procedure can only be called in an algorithmic sequence before the resolution (not possible in a service of a station).

This procedure has got a sens only for simulation.

SEE ALSO

SETSTAT:CLASS - SETSTAT:SERVICE:MEAN - SETSTAT:RESPONSE:MEAN -
SETSTAT:BLOCKED:MEAN - SETSTAT:BUSYPCT:MEAN - SETSTAT:THRUPUT:MEAN -
GETSTAT:SERVICE:MEAN - GETSTAT:RESPONSE:MEAN - GETSTAT:BLOCKED:MEAN -
GETSTAT:BUSYPCT:MEAN - GETSTAT:CUSTNB :MEAN - GETSTAT:THRUPUT:MEAN -
SETSTAT:CANCEL

EXAMPLE

```
/DECLARE/ QUEUE q1, q2;
            .....

/EXEC/ BEGIN
         SETSTAT:QUEUE (ALL QUEUE);
         ......
         SIMUL;
         PRINT (GETSTAT:SERVICE:VARIANCE (q1)) ;
       END;
```

SETSTAT:RESPONSE:MEAN

NAME

SETSTAT:RESPONSE:MEAN - Statistical results request on the response time of a queue.

SYNTAX

SETSTAT:RESPONSE:MEAN (*list_of_queues*, *list_of_classes*) ;

DESCRIPTION

Constitutes an explicit request on standard statistical results: mean, variance, minimum and maximum on the response time for each (*queue*, *class*) specified couple.

These results are available by means of the following functions :
GETSTAT:RESPONSE:MEAN
GETSTAT:RESPONSE:MAXIMUM
GETSTAT:RESPONSE:MINIMUM
GETSTAT:RESPONSE:VARIANCE

More complex results may be requested by means of the following procedures :
SETSTAT:RESPONSE:ACCURACY for an accuracy.
SETSTAT:RESPONSE:PRECISION for a stop on a fixed precision.
SETSTAT:RESPONSE:MARGINAL for marginal probabilities.
SETSTAT:RESPONSE:CORRELATION for auto-correlation coefficients.

Periodical results may be computed by calling the SETSTAT:PARTIAL procedure.

EVALUATION

At the beginning of the resolution.

WARNING

No more complex statistical result can be requested on the service time if not preceded by SETSTAT:RESPONSE:MEAN (*queue*) or SETSTAT:QUEUE (*queue*).

This procedure can only be called in an algorithmic sequence before the resolution (not possible in a service of a station).

This procedure has got a sens only for simulation.

SEE ALSO

SETSTAT:QUEUE - SETSTAT:CLASS - GETSTAT:RESPONSE:MEAN - SETSTAT:PARTIAL - SETSTAT:ACCURACY - SETSTAT:MARGINAL - SETSTAT:CORRELATION - SETSTAT:PRECISION - SETSTAT:CANCEL

EXAMPLE

```
/DECLARE/ QUEUE q;
          CLASS c1, c2;
          .....

/EXEC/ BEGIN
         SETSTAT:RESPONSE:MEAN (q );
         SETSTAT:RESPONSE:MEAN (q, c1);
         SETSTAT:RESPONSE:ACCURACY (q, c1);
         ....
         SIMUL;
         ....
       END;
```

SETSTAT:SAMPLE

NAME

SETSTAT:SAMPLE - Computes all the basic statistics concerning the variable.

SYNTAX

SETSTAT:SAMPLE (*list_of_variables*) ;

DESCRIPTION

Computes the basic statistics of the variable concerned : the mean, the variance, the maximum and the minimum.

Variables can be one of the following types :
- integer or real (WATCHED declared)
- queue
- queue followed by a list of classes.

The results are available by means of the functions : GETSTAT:MEAN (*variable*) ; or GETSTAT:SERVICE:MEAN (*queue*, *class*);.

These results are partial if the variable has been the object of a SETSTAT:PARTIAL.

EVALUATION

During the execution.

NOTE

The SAMPLE procedure computes the statistics concerning all the variables of the model.

WARNING

This procedure can not be used if an accuracy, computed with the replication method or the spectral method, was requested on the variable.

This procedure can only be called in an algorithmic sequence before the resolution (not possible in a service of a station).

This procedure can only be used for simulation.

SEE ALSO

WATCHED - SETSTAT:PARTIAL - SETSTAT:CLASS - SETSTAT:QUEUE - SETSTAT:SERVICE - SETSTAT:RESPONSE - SETSTAT:BLOCKED - SETSTAT:CUSTNB - SETSTAT:BUSYPCT - SAMPLE - PERIOD

EXAMPLE

```
 /DECLARE/ QUEUE q;
           CLASS c1, c2;
           REAL result

/CONTROL/ TMAX = 10000;
          PERIOD = 2000;
          ESTIMATION = REGENERATION;
          TEST = BEGIN
                   SETSTAT:SAMPLE (q);
                   result := GETSTAT:SERVICE:MEAN (q, c1);
                 END;

/EXEC/ BEGIN
         SETSTAT:QUEUE (q, c1); & request of statistical results
         SETSTAT:ACCURACY (q, c1);
         ....
         SIMUL;
         PRINT (GETSTAT:RESPONSE:ACCURACY (q, c1);
         ....
    END;
```

SETSTAT:SERVICE:MEAN

NAME

SETSTAT:SERVICE:MEAN - Statistical results request on the service time of a queue.

SYNTAX

SETSTAT:SERVICE:MEAN (*list_of_queues*, *list_of_classes*) ;

DESCRIPTION

Constitutes an explicit request of standard statistical results : mean, variance, minimum and maximum on the service time for each (*queue*, *class*) specified couple.

These results are available by means of the following functions These results are available by means of the following functions:
GETSTAT:SERVICE:MEAN
GETSTAT:SERVICE:MAXIMUM
GETSTAT:SERVICE:MINIMUM
GETSTAT:SERVICE:VARIANCE

More complex results can be requested by means of the following procedures :
SETSTAT:SERVICE:ACCURACY for an accuracy.
SETSTAT:SERVICE:PRECISION for a stop on a fixed precision.
SETSTAT:SERVICE:MARGINAL for marginal probabilities.
SETSTAT:SERVICE:CORRELATION for auto-correlation coefficients.

Periodical results may be computed by calling the SETSTAT:PARTIAL procedure.

EVALUATION

At the beginning of the resolution.

WARNING

No more complex statistical result can be requested on the service time if not preceded by SETSTAT:SERVICE:MEAN (*queue*) or SETSTAT:QUEUE (*queue*).

This procedure can only be called in an algorithmic sequence before the resolution (not possible in a service of a station).

This procedure has got a sens only for simulation.

SEE ALSO

SETSTAT:QUEUE - SETSTAT:CLASS - GETSTAT:SERVICE:MEAN - SETSTAT:PARTIAL - SETSTAT:ACCURACY - SETSTAT:MARGINAL - SETSTAT:CORRELAT - SETSTAT:PRECISION - SETSTAT:CANCEL

EXAMPLE

```
/DECLARE/ QUEUE q;
          CLASS c1, c2;
          .....

/EXEC/ BEGIN
         SETSTAT:SERVICE:MEAN (q);
         SETSTAT:SERVICE:MEAN (q, c1);
         SETSTAT:SERVICE:ACCURACY (q, c1);
         ....
         SIMUL;
         ....
       END;
```

SETSTAT:THRUPUT:MEAN

NAME

SETSTAT:THRUPUT:MEAN - Statistical results request on the throughput of a queue.

SYNTAX

SETSTAT:THRUPUT:MEAN (*list_of_queues*, *list_of_classes*) ;

DESCRIPTION

Constitutes an explicit request of the standard statistical result: mean (only) of the throughput for each (*queue*, *class*) specified couple.

This result is available by means of the following function:
GETSTAT:THRUPUT:MEAN

More complex results can be requested by means of the following procedures:
SETSTAT:THRUPUT:ACCURACY for an accuracy.
SETSTAT:THRUPUT:CORRELATION for auto-correlation coefficients.

Periodical results may be computed by calling the SETSTAT:PARTIAL procedure.

EVALUATION

At the beginning of the resolution.

WARNING

No more complex statistical result can be requested on the service time if not preceded by SETSTAT:THRUPUT:MEAN (*queue*) or SETSTAT:QUEUE (*queue*).

This procedure can only be called in an algorithmic sequence before the resolution (not possible in a service of a station).

This procedure has got a sens only for simulation.

SEE ALSO

SETSTAT:QUEUE - SETSTAT:CLASS - GETSTAT:THRUPUT:MEAN - SETSTAT:PARTIAL - SETSTAT:ACCURACY - SETSTAT:MARGINAL - SETSTAT:CORRELATION - SETSTAT:PRECISION - SETSTAT:CANCEL

EXAMPLE

```
/DECLARE/ QUEUE q;
          CLASS c1, c2;
          ...

/EXEC/ BEGIN
         SETSTAT:THRUPUT:MEAN (q);
         SETSTAT:THRUPUT:MEAN (q, c1);
         SETSTAT:THRUPUT:ACCURACY (q, c1);
         ...
         SIMUL;
         ...
       END;
```

SETTIMER

NAME

`SETTIMER` - `TIMER` objects handling.

SYNTAX

`SETTIMER:key-word`

DESCRIPTION

The avalaible key-words are :

SETTIMER:ABSOLUTE	Sets a timer for an absolute simulated time.
SETTIMER:CANCEL	Cancels a TIMER.
SETTIMER:CYCLIC	Sets a cyclical timer.
SETTIMER:RELATIVE	Sets a timer for a relative simulated time.
SETTIMER:SETPROC	Defines the procedure to be launched when the timer wakes up.
SETTIMER:TRACKTIME	Sets a cyclical and systematic timer after each simulation event.

NAME

SETTIMER:ABSOLUTE - Sets a timer for an absolute simulated time.

SYNTAX

SETTIMER:ABSOLUTE (*timer, real*) ;

DESCRIPTION

Programs the future activation of the specified timer for the absolute specified time. Then, the timer will be active except cancel using SETTIMER:CANCEL or a new activation undertaken before the provided time.

real is a real which represents the activation time (absolute time).

timer is a reference on a TIMER object.

EVALUATION

During the execution.

NOTES

A call to this procedure may be placed in the procedure associated to a timer and allows the timer to launch itself for a future time.

WARNING

This operation is forbidden on the TSYSSPCT and TSYSWDOG timers, but may be used on the TSYSTMAX timer. Moreover, if the user gives an activation time less than the current time, an error occurs.

SEE ALSO

SETTIMER:RELATIVE - SETTIMER:CYCLIC - SETTIMER:TRACKTIME - SETTIMER:SETPROC - SETTIMER:CANCEL - TIMER - STATE - ACTIARG - SETTIMER:SETPROC

EXAMPLE

```
 /DECLARE/ TIMER MYTIMER;
           PROCEDURE TIMHANDLE;
           BEGIN
             PRINT ("TIMER ACTIVE AT TIME", TIME);
           END;

 /STATION/ NAME = q;
           SERVICE = BEGIN
                       SETTIMER:ABSOLUTE (MYTIMER, 10.0);
&  The provided message will be edited at 10.0
                     END;

/EXEC/ BEGIN
         ...
         SETTIMER:SETPROC (MYTIMER, TIMHANDLE);
& connection of the timer to the treatment procedure
         ...
         SIMUL;
       END;
```

NAME

SETTIMER:CANCEL - Cancels a TIMER object.

SYNTAX

SETTIMER:CANCEL (*timer*) ;

DESCRIPTION

Cancels a TIMER object, previously activated or not.

timer is a reference on a TIMER object.

EVALUATION

During the execution.

NOTES

This operation has no effects if not executed during a simulation.

WARNING

This operation is forbidden on the TSYSTMAX, TSYSSPCT and TSYSWDOG timers. An error occurs if the user calls this procedure for such timers.

SEE ALSO

SETTIMER:ABSOLUTE - SETTIMER:RELATIVE - SETTIMER:CYCLIC - SETTIMER:TRACKTIME - TIMER - STATE - ACTIARG - SETTIMER:SETPROC

EXAMPLE

```
 /DECLARE/ TIMER MYTIMER;
           PROCEDURE TIMHANDLE;
           BEGIN
             PRINT ("TIMER ACTIVE AT TIME", TIME);
           END;

 /STATION/ NAME = q;
           SERVICE = BEGIN
                       SETTIMER:ABSOLUTE (MYTIMER, 10.0);
                       SETTIMER:CANCEL (MYTIMER);
& cancels the previous call to SETTIMER:ABSOLUTE
                       ...
                     END;

/EXEC/ BEGIN
         SETTIMER:SETPROC (MYTIMER, TIMHANDLE);
& connection of the timer to the treatment procedure
         SIMUL
       END;
```

NAME

SETTIMER:CYCLIC - Cyclical activation of a time.

SYNTAX

SETTIMER:CYCLIC (*timer, real*) ;

DESCRIPTION

Programs the future activation of the specified timer at the specified delay after the current time, as SETTIMER:RELATIVE, but asks again automatically an activation after the specified delay after each execution of the associated procedure.

This automatic activation will run until the end of the simulation or a call to a procedure concerning the timer which cancels this specification.

This procedure allows to emulated the PERIOD option of the /CONTROL/ command with a user timer. The real argument represents the delay between each activation.

timer is a reference on a TIMER object.

EVALUATION

During the execution.

WARNING

This operation is forbidden on the TSYSTMAX, TSYSSPCT and TSYSWDOG timers. Moreover, if the user gives a negative delay then an error occurs.

SEE ALSO

SETTIMER:ABSOLUTE - SETTIMER:RELATIVE - SETTIMER:TRACKTIME - SETTIMER:SETPROC - SETTIMER:CANCEL - TIMER - STATE - ACTIARG - SETTIMER:SETPROC

EXAMPLE

```
 /DECLARE/ TIMER MYTIMER;
           PROCEDURE TIMHANDLE;
           BEGIN
             PRINT ("TIMER ACTIVE AT TIME", TIME);
           END;

 /STATION/ NAME = q;
           SERVICE = BEGIN
                       SETTIMER:CYCLIC (MYTIMER, 10.0);
& The provided message will be displayed every 10 time units
                     END;

/EXEC/ BEGIN
         ...
         SETTIMER:SETPROC (MYTIMER, TIMHANDLE);
& connection of the timer to the treatment procedure
         ...
         SIMUL;
       END;
```

NAME

SETTIMER:RELATIVE - Sets a timer for a relative simulated time.

SYNTAX

SETTIMER:RELATIVE (*timer, real*) ;

DESCRIPTION

Programs the future activation of the specified timer at the specified delay after the current time. Then, the timer will be active except cancel using SETTIMER:CANCEL or a new activation undertaken before the provided time.

real is a real which represents the delay from the current time until the timer will be activated

timer is a reference on a TIMER object.

EVALUATION

During the execution.

NOTES

A call to this procedure may be placed in the procedure associated to a timer and allows the timer to launch itself for a future time.

WARNING

This operation is forbidden on the TSYSSPCT and TSYSWDOG timers, but may be used on the TSYSTMAX timer. Moreover, if the user gives a negative delay then an error occurs.

SEE ALSO

SETTIMER:ABSOLUTE - SETTIMER:CYCLIC - SETTIMER:TRACKTIME - SETTIMER:SETPROC - SETTIMER:CANCEL - TIMER - STATE - ACTIARG - SETTIMER:SETPROC

EXAMPLE

```
 /DECLARE/ TIMER MYTIMER;
           PROCEDURE TIMHANDLE;
           BEGIN
             PRINT ("TIMER ACTIVE AT TIME", TIME);
           END;

 /STATION/ NAME = q;
           SERVICE = BEGIN
                       SETTIMER:RELATIVE (MYTIMER, 10.0);
&  The provided message will be edited at 10.0 time units after the current time
                     END;

/EXEC/ BEGIN
         ...
         SETTIMER:SETPROC (MYTIMER, TIMHANDLE);
& connection of the timer to the treatment procedure
         ...
         SIMUL;
       END;
```

NAME

SETTIMER:SETPROC - Defines the procedure associated to the timer.

SYNTAX

SETTIMER:SETPROC (*timer, procedure*);

DESCRIPTION

Associates a procedure to the timer. This procedure will be run each time the timer is activated.

procedure is a reference on a procedure without argument

timer is a reference on a TIMER object.

EVALUATION

During the execution.

NOTES

The execution of this procedure on the TSYSPERI, TSYSTSTR, and TSYSTRACE timers cancels the automatic treatment associated to the PERIOD, TSTART, et TRACE options of the /CONTROL/ command.

Then, the user must define himself the corresponding actions.

WARNING

This operation is forbidden on the TSYSTMAX, TSYSSPCT, and TSYSWDOG timers.
The TIMER variable refers to the current timer when the procedure is running.

SEE ALSO

SETTIMER:ABSOLUTE - SETTIMER:RELATIVE - SETTIMER:CYCLIC - SETTIMER:TRACKTIME - HANDLER - TIMER

EXAMPLE

```
/DECLARE/ PROCEDURE TOPTIMER;
          BEGIN
            PRINT ("TIMER ACTIVE AT TIME", TIME);
          END;
          TIMER T1;

/EXEC/ BEGIN
         ...
         SETTIMER:SETPROC  (T1, TOPTIMER);
& Associates T1 timer to the TOPTIMER procedure .
         ...
         SIMUL;
         ...
       END;
```

NAME

SETTIMER:TRACKTIME - Sets a cyclical and systematic timer after each simulation event.

SYNTAX

SETTIMER:TRACKTIME (*timer*) ;

DESCRIPTION

Programs the cyclical activation of the specified timer just behind the next event provided in the scheduler, except the simultaneous events. The timer is called once at a given time with the smallest priority in comparison with the events provided at the same time. This procedure allows to emulate the PERIOD = 0; option of the /CONTROL/ command.

timer is a reference on a TIMER object.

EVALUATION

During the execution.

WARNING

This operation is forbidden on the TSYSTMAX, TSYSSPCT, and TSYSWDOG timers.

SEE ALSO

SETTIMER:ABSOLUTE - SETTIMER:RELATIVE - SETTIMER:CYCLIC - SETTIMER:SETPROC - SETTIMER:CANCEL - TIMER - STATE - ACTIARG - SETTIMER:SETPROC

EXAMPLE

```
 /DECLARE/ TIMER MYTIMER;
           PROCEDURE TIMHANDLE;
           BEGIN
             PRINT ("LAST EVENT ARRIVED AT TIME", TIME);
             PRINT ("THIS MECHANISM IS EXPENSIVE");
           END;

 /STATION/ NAME = q;
           SERVICE = BEGIN
                       SETTIMER:TRACKTIME (MYTIMER);
&  The provided message will be displayed after each even
                     END;

/EXEC/ BEGIN
         SETTIMER:SETPROC (MYTIMER, TIMHANDLE);
& connection of the timer to the treatment procedure
         SIMUL;
       END;
```

NAME

SETTRACE - Handling of the trace events in simulation and-or the associated generalized notions.

SYNTAX

```
SETTRACE:key-word
```

DESCRIPTION

The available key-words are :

SETTRACE:BOUNDS	Defines the bounds of the time interval when the trace is automatically activated.
SETTRACE:BRIEF	Selects the brief mode for the standard trace in simulation (default mode).
SETTRACE:DEFRESET	Cancels, if necessary by event type, the effect of a previous call to SETTRACE:DEFSET.
SETTRACE:DEFSET	Defines the trace treatment procedure to be applied by default on all the entities which do not have a specific one. By default, this treatment may depend on specific event types.
SETTRACE:DISPLAY	Displays the standard trace of an event.
SETTRACE:WIDTH	Selects the width of the standard trace.
SETTRACE:LONG	Selects the large mode for the standard trace (brief mode by default).
SETTRACE:OFF	Turn off the trace mechanisms.
SETTRACE:ON	Turn on the trace mechanisms.
SETTRACE:RESET	Cancels the effect of a previous call to SETTRACE:SET on an entity, if necessary by event type.
SETTRACE:SET	Defines a trace treatment procedure on a given entity and if necessary on precise event types.

These procedures allow to handle the trace concept in simulation. The trace is by default a display in a file, but may be replaced by a specific operation for each event or a part of them which is described in a specific procedure or a predefined one. This procedure may read all the data displayed on a standard trace and all the data known in the model without limitation. In such a procedure, the standard display may be called by SETTRACE:DISPLAY, or using specific user displays. These facilities may be used for instance for the graphical animation of a model.

Globally, these procedures allow :

- Select among the information usually displayed. The entities which may be individually traced are queues, (queue, class) couples, classes, customers, FLAG objects, TIMER objects. A specific trace treatment procedure may be defined for each of these entities.
- Select the event types to be traced, for all the entities, or for one given entity or mixed. A specific trace treatment procedure may be attached to each event type.

The control of the trace is handled by the SETTRACE:SET, SETTRACE:RESET, SETTRACE:DEFSET, and SETTRACE:DEFRESET procedures.

EVALUATION

During the execution.

NOTES

The previous SETTRACE procedure implemented in the QNAP2 versions before the V 9.0 is replaced by the calls to SETTRACE:BOUNDS in order to define the start time and the end time, and SETTRACE:WIDTH to define the width of the trace.

NAME

SETTRACE:BOUNDS - Defines the bounds of the time interval when the trace is automatically activated.

SYNTAX

SETTRACE:BOUNDS (*real1, real2*) ;

DESCRIPTION

The two real arguments respectively represent the starting and ending time of the period when the trace is active.

The trace will be automatically run when the start time is reached.

The trace will be stopped as soon as the end time is reached.

This mechanism is equivalent to the use of the TRACE parameter of the /CONTROL/ command. The control of the trace (turn on/off) is handled like the SETTRACE:ON and SETTRACE:OFF procedure, that means that the user may choose to manage the trace himself.

The trace is turned on/off using the predefined TSYSTRACE object (TIMER type). The user is allowed to handle this timer himself. If it is the case, a usual consequence is that the turn on/off operations do not run, so the use of SETTRACE:BOUNDS is not efficient.

EVALUATION

During the execution.

NOTES

If the TSYSTRACE timer is managed by the user, the execution of SETTRACE:BOUNDS during the simulation causes the activation of this timer for the specified start time or end time. The effect may be not desirable if the user prefers to control such an activation himself.

Otherwise, the specification of a user treatment procedure for this timer cancels the implicit treatment which consists of turning on/off the trace. Then, the user must manage himself calls to SETTRACE:ON or SETTRACE:OFF.

WARNING

An error occurs if the specified end time is less than the start time or if these values are not greater than zero or null.

SEE ALSO

SETTRACE:ON - SETTRACE:OFF

EXAMPLE

```
/DECLARE/ PROCEDURE TREATP;
          BEGIN
            ....
          END;

/CONTROL/ TMAX=500;

/EXEC/ BEGIN
        SETTRACE:BOUNDS (100,300);
        SETTRACE:SET(Q1,"V",TREATP);
        ....
        SIMUL;
       END;
```

NAME

SETTRACE:BRIEF - Selects the brief mode for the simulation standard trace edition.

SYNTAX

```
SETTRACE:BRIEF ;
```

DESCRIPTION

Selects the brief mode for editing the standard trace in a simulation (the brief mode is the default mode). In this mode, the number of the source code line which called the traced operation is not edited after the operation. This description concerns an automatical editing and also a call to SETTRACE:DISPLAY.

EVALUATION

During the execution.

NOTES

This procedure allows to cancel the effects of a previous call to SETTRACE:LONG. A call to this procedure is ignored if the brief mode is already set.

SEE ALSO

SETTRACE:DISPLAY - SETTRACE:LONG

EXAMPLE

```
/DECLARE/ PROCEDURE TREATP;
          BEGIN
           ....
          END;

/EXEC/ BEGIN
         SETTRACE:BRIEF;
         SETTRACE:WIDTH (80);
         SETTRACE:SET(Q1,"V",TREATP);
         SIMUL;
       END;
```

SETTRACE:DEFRESET

NAME

SETTRACE:DEFRESET - Cancels, possibly by event type, the specification of a default trace.

SYNTAX

SETTRACE:DEFRESET [(*event_type*)] ;

DESCRIPTION

If a treatment defined for an event type is cancelled then the default treatment for all the events prevails. If this global treatment is set to the default then a standard editing will be done.

event_type is a string (or an array or a list of strings) which defines, in the form of symbolic names, the event(s) to be handled. In a same string, several events may be defined using comma between each name. Blanks at the top and at the end of the string are ignored. "ALLEV" selects all the event types. If no event type is defined, then the action will be assigned to all the events which are not provided moreover for the considered entity.

EVALUATION

During the execution.

SEE ALSO

SETTRACE:SET - SETTRACE:RESET - SETTRACE:DEFSET - Event types

EXAMPLE

```
/DECLARE/ PROCEDURE TREATP;
          BEGIN
            .....
          END;


/CONTROL/ OPTTION=TRACE;

/EXEC/ BEGIN
        SETTRACE:DEFSET(NIL);
        SETTRACE:SET(Q1,"V",TREATP);
        ....
        SIMUL;
        SETTRACE:DEFRESET;
       END;
/END/
```

NAME

SETTRACE:DEFSET - Defines a default trace treatment procedure.

SYNTAX

SETTRACE:DEFSET ([*event_types*], *treatment_procedure*) ;

DESCRIPTION

The syntax of this call is the same as SETTRACE:SET, except that no entity is defined because this procedure concerns the default actions to be applied to the entities for whom nothing is defined.

- *event_type* is a string (or an array or a list of strings) which defines, in the form of symbolic names, the event(s) to be handled. In a same string, several events may be defined using comma between each name. Blanks at the top and at the end of the string are ignored. "ALLEV" selects all the event types. If no event type is defined, then the action will be assigned to all the events which are not provided moreover for the considered entity.

- *treatment_procedure* is a procedure (or a REF procedure) without argument which will be called in simulation when an event concerns the specified entity (if necessary with one of the specified event types).

EVALUATION

During the execution.

NOTES

The NIL procedure may be defined in order to avoid a non specific trace action for a trace entity.

The effects of a previous call to SETTRACE:SET may be cancelled using a call to SETTRACE:DEFRESET.

WARNING

The SETTRACE:DEFSET (NIL) call is not equivalent with SETTRACE:OFF which must be used if the user wants to stop all the trace actions. In the opposite case, the internal handling of the trace mechanism would be done, then stopped at each event (CPU cost increasing)

SEE ALSO

SETTRACE:SET - SETTRACE:RESET - SETTRACE:DEFRESET - SETTRACE:ON - SETTRACE:OFF - Event types.

EXAMPLE

```
/DECLARE/ PROCEDURE TREATP;
          BEGIN
            .....
          END;


/CONTROL/ OPTTION=TRACE;

/EXEC/ BEGIN
        SETTRACE:DEFSET(NIL);
        SETTRACE:SET(Q1,"V",TREATP);
        SETTRACE:SET(Q2,"P",TREATP);
        SIMUL;
       END;
/END/
```

NAME

SETTRACE:DISPLAY - Performs the standard display of a trace event during the simulation.

SYNTAX

SETTRACE:DISPLAY ;

DESCRIPTION

This procedure allows the user to display an event in the standard form and on the FSYSTRACE file if the current trace action is a call to a user procedure. Indeed, the execution of such a procedure would replace the standard editing that it is possible to continue forcing using the language.

EVALUATION

During the execution and only during the execution of a user trace procedure.

NOTES

The format and the nature of the information may be modified using calls to SETTRACE:WIDTH, SETTRACE:LONG, and SETTRACE:BRIEF.

WARNING

An error occurs if this procedure is called out of a trace treatment.

SEE ALSO

SETTRACE:WIDTH - SETTRACE:LONG - SETTRACE:BRIEF

EXAMPLE

```
/DECLARE/ PROCEDURE TREATP;
          BEGIN
           SETTRACE:DISPLAY;
          END;
....

/EXEC/ BEGIN
        SETTRACE:ON;
        SETTRACE:SET(Q1,"V",TREATP);
        SETTRACE:SET(Q2,"P",TREATP);
        SIMUL;
       END;
```

SETTRACE:LONG

NAME

SETTRACE:LONG - Selects the large mode for the simulation standard trace edition.

SYNTAX

SETTRACE:LONG ;

DESCRIPTION

Selects the large mode for the edition of the standard trace in simulation (the brief mode is the default mode). In this mode, the number of the source code line which called the traced operation is edited after the operation. This description concerns an automatical editing and also a call to SETTRACE:DISPLAY.

EVALUATION

During the execution.

NOTES

A call to this procedure is ignored if the large mode is already set.

SEE ALSO

SETTRACE:DISPLAY - SETTRACE:BRIEF

EXAMPLE

```
/DECLARE/ PROCEDURE TREATP;
          BEGIN
           ....
          END;
....

/EXEC/ BEGIN
        SETTRACE:ON;
        SETSTAT:LONG;
        SETTRACE:SET(Q1,"V",TREATP);
        ....
        SIMUL;
       END;
```

NAME

SETTRACE:OFF - Turns off the trace.

SYNTAX

```
SETTRACE:OFF ;
```

DESCRIPTION

Turns off the trace. The cancelled operation was an editing or a call to a user procedure.

EVALUATION

During the execution.

NOTES

This procedure may be called several times. If the trace is already turned off, a call to this procedure will be ignored.

SEE ALSO

SETTRACE:ON

EXAMPLE

```
/DECLARE/ PROCEDURE TREATP;
          BEGIN
           ...
          END;


/CONTROL/ TMAX= 50;
          TRACE=0.0,40.0;
           ....

/EXEC/ BEGIN
        SETTRACE:OFF;
        SIMUL;
       END;
```

SETTRACE:ON

NAME

SETTRACE:ON - Turns on the trace.

SYNTAX

SETTRACE:ON ;

DESCRIPTION

Turns on the trace mechanism. The action may be an edition or a user procedure execution.

EVALUATION

During the execution.

NOTES

This procedure may be called several times. If the trace is already turned on, a call to this procedure will be ignored.

SEE ALSO

SETTRACE:OFF

EXAMPLE

```
/DECLARE/ PROCEDURE TREATP;
          BEGIN
           ...
          END;


/CONTROL/ TMAX= 50;

/EXEC/ BEGIN
        SETTRACE:ON;
        SETTRACE:SET(Q1,"V",TREATP);
        SETTRACE:SET(Q2,"P",TREATP);
        SIMUL;
       END;
```

NAME

SETTRACE:RESET - Cancels a previous request of specific trace.

SYNTAX

SETTRACE:RESET (*entity* [, *event_type*]) ;

DESCRIPTION

Cancels the effects of a previous call to SETTRACE:SET on an entity or an event type.

When the effects of a previous call to SETTRACE:SET have been cancelled, the default treatments are restored. The rules are :

- When a treatment dedicated to an event type has been reset (default treatment restored), the default action specification for all the events assigned to the current entity is applied. The priority for the choice between several possible trace actions keeps assigned by this entity.
- If no treatment specification (except the default one) exists for an entity then a specification existing for another current entity may be applied, else the default actions defined by SETTRACE:DEFSET and/or SETTRACE:DEFRESET are undertaken.

- *entity* defines either:

- a queue (or a set of queues)
- a class (or a set of classes)
- a (queue, class) couple (or a set of couples)
- a customer (or a list of customers)
- a FLAG object (or a set of such objects)
- a TIMER object (or a set of such objects)

Object sets are allowed everywhere here (using only one object type, except for the (queue,class) couples) and must be defined in the form of lists.

- *event_type* is a string (or an array or a list of strings) which defines in the form of symbolic names the event(s) to be treated. In a same string, several events may be defined using comma between each name. Blanks at the top and at the end of the string are ignored. "ALLEV" selects all the event types. If no event type is defined, then the action will be assigned to all the events which are not provided moreover for the considered entity.

EVALUATION

During the execution.

NOTES

Restoring the default trace operation is not the same than assigning NIL to the treatment procedure. Indeed, the NIL specification is explicit and asks that an event must be ignored, even when restoring the default restores a standard edition.

WARNING

This procedure is not provided to cancel the effects of a previous call to SETTRACE:DEFSET. In this case the SETTRACE:DEFRESET procedure must be called.

SEE ALSO

SETTRACE:SET - SETTRACE:DEFSET - SETTRACE:DEFRESET - eventtypes

EXAMPLE

```
/DECLARE/ PROCEDURE TREATP;
          BEGIN
           .....
          END;

/EXEC/ BEGIN
         SETTRACE:SET(Q1,"V",TREATP);
         SETTRACE:SET(Q2,"P",TREATP);
         SIMUL;
         SETTRACE:RESET(Q1,"V");
         SIMUL;
       END;
```

NAME

SETTRACE:SET - Defines a trace treatment procedure on a specific entity and if necessary on precise event types.

SYNTAX

SETTRACE:SET (*entity* [, *event_type*], *treatment_procedure* [,*priority 1*, *priority 2*, *priority 3*]) ;

DESCRIPTION

This procedure allows to defines a trace treatment for entities or precise events.

- *entity* represents either:
  - a queue (or a set of queues)
  - a class (or a set of classes)
  - a (queue, class) couple (or a set of couples)
  - a customer (or a set of customers)
  - a FLAG object (or a set of such objects)
  - a TIMER object (or a set of such objects)

Object sets are allowed everywhere here (using only one object type, except for the (queue,class) couples) and must be defined in the form of lists.

- *event_type* is a string (or an array or a list of strings) which defines in the form of symbolic names the event(s) to be treated. In a same string, several events may be defined using comma between each name. Blanks at the top and at the end of the string are ignored. "ALLEV" selects all the event types. If no event type is defined, then the action will be assigned to all the events which are not provided moreover for the considered entity.

- *treatment_procedure* is a procedure (or a REF procedure) without argument which will be called in simulation when an event concerns the specified entity (if necessary with one of the specified event types), unless the treatment of the current entity stops for the advantage of entity with a higher priority (see hereafter).

- *priority 1*, *priority 2*, *priority 3* are integer. These priorities are used when several trace operations, perhaps contradictory, may be applied at a given time; It may be when a trace operation has been defined for the current customer at a given time, and for the current queue and perhaps for another customer or queue concerned by the operation.
The first priority applies when the examined entity is the entity which caused the traced operation (for instance : the current customer forces another customer waiting).
The second priority applies when the examined entity is the subject of the operation (in the previous example, it may be the customer who must wait or the FLAG object where the customer must wait).
The third priority applies when the examined entity is partner of the operation (for instance, customer in front or behind whom another customer is inserted by BEFCUST ou AFTCUST).

The state *causing*, *subject* or *partner* for an entity is mostly implicit but may result from specific rules dedicated to each event type. The priorities grow with the integer which represent them. If the entities which are subject of the operation have the same priority than the entities causing it then it is considered that the first ones have priority. In the same way the entities causing the operation have priority in comparison with partners entities. With a same priority level, it is considered that the decreasing entities order is :

- customers or TIMER objects
- FLAG objects
- (queue, class) couples
- queue
- class
- default action for every entity

If no specific trace action is selected for the entities on whom a trace action may be selected, or if the trace action is the default one, then the default trace is applied. This action is defined using a call to SETTRACE:DEFSET very similar to SETTRACE:SET. If nothing is defined by the user then the implicit action is the editing of the event trace on FSYSTRACE.

The effects of a previous call to SETTRACE:SET may be cancelled at any time calling SETTRACE:RESET.

EVALUATION

During the execution. The specifications concerning customers must be done during the simulation.

NOTES

The most usual trace action is an editing on file. In order to cancel some events in this editing, the method consists in specifying a default treatment using SETTRACE:DEFSET. The specification of NIL procedure means : ignore the considered events as if they did not occur. In order to specify the entities to trace, ignoring implicitly the other ones, the method consists in assigning by default the NIL procedure for the implicit trace action (using SETTRACE:DEFSET), and assigning to the entities to be traced a user procedure containing the single statement : SETTRACE:DISPLAY.

In a user trace treatment procedure, the characteristics of the traced event may be read using the GETTRACE:key-word function.

The trace mechanism which performs editings or calls to user procedures may be globally turned on/off using SETTRACE:ON and SETTRACE:OFF.

WARNING

To cancel all trace actions during a part or all the simulation, the user must turn off globally the trace instead of defining everywhere a NIL treatment procedure. This second method is equivalent to the first one in a logical point of view however will increase the CPU cost because the trace handling will be done at each event and will be aborted.

SEE ALSO

SETTRACE:RESET - SETTRACE:DEFSET - SETTRACE:DEFRESET -SETTRACE:DISPLAY - SETTRACE:ON - SETTRACE:OFF - GETTRACE:key-word - Event types.

EXAMPLE

```
/DECLARE/ QUEUE q1;
          FLAG f1;

          PROCEDURE proc1;
          BEGIN
            ...
          END;

          PROCEDURE proc2;
          BEGIN
            ...
          END;

/EXEC/ BEGIN
         SETTRACE:SET (q1, "P", proc1);
         SETTRACE:SET (f1, "WAIT", proc 2);
         ...
       END;
```

SETTRACE:WIDTH

NAME

SETTRACE:WIDTH - Selects the width of the editing line for the standard trace in simulation.

SYNTAX

SETTRACE:WIDTH (*integer*) ;

DESCRIPTION

This procedure performs the same operation than the LL80 or LL132 option of the TRACE parameter (/CONTROL/ command) : selects an editing line with 80 or 132 characters for the strandard trace in simulation on the FSYSTRACE file. The editing may occur either during the implicit treatment of the trace, either following calls to SETTRACE:DISPLAY.

The *integer* argument may take the values 80 or 132.

EVALUATION

During the execution.

NOTES

This procedure replaces the corresponding functionality of the SETTRACE procedure existing in the QNAP2 versions before the V9.0.

WARNING

The width must be 80 or 132.

SEE ALSO

SETTRACE:DISPLAY - SETTRACE:BRIEF - SETTRACE:LONG

EXAMPLE

```
/DECLARE/ PROCEDURE TREATP;
          BEGIN
           ....
          END;

/EXEC/ BEGIN
         SETTRACE:WIDTH (132);
         SETTRACE:SET(Q1,"V",TREATP);
         ....
         SIMUL;
       END;
```

7.3.2 Mathematic Tools

ABS	Integer absolute value.
ACOS	Arccosine trigonometric function.
ASIN	Arcsine trigonometric function.
ATAN	Arctangent trigonometric function.
COS	Cosine trigonometric function.
EXPO	Exponential function.
FIX	Integer part of a real number.
INTREAL	Integer value of a real number.
INTROUND	Nearest integer to a real number.
LOG	Natural logarithm function.
LOG10	Logarithm to base 10 function.
MAX	Maximum of several values.
MIN	Minimum of several values.
MOD	Modulo between two integers (remainder from their division).
REALINT	Real value of an integer.
SIN	Sine trigonometric function.
SQRT	Squared root function.
TAN	Tangent trignometric function.

ABS

NAME

ABS - Integer absolute value.

SYNTAX

ABS (*integer* | *real*)

DESCRIPTION

This function returns the absolute value of the number defined by argument. If the argument is an integer then the function returns an integer value and if the argument is a real number then the function returns a real value.

integer is an expression representing an INTEGER type entity.

real is an expression representing a REAL type entity.

EVALUATION

During the execution.

EXAMPLE

```
/DECLARE/ INTEGER i;
          REAL r = 2.1;

/EXEC/ BEGIN
         i := ABS (-1);              & result : 1
         PRINT (ABS (2* r-i));       & result : 3.2
       END;
```

NAME

ACOS - Arccosine trigonometric function.

SYNTAX

ACOS (*real*)

DESCRIPTION

This function returns a real number which is the trigonometric arccosine of the number defined by argument.

real is an expression representing a REAL type entity.

EVALUATION

During the execution.

NOTE

The returned value is given in radians.

WARNING

The definition domain of the argument is [-1, 1].

EXAMPLE

```
/DECLARE/ REAL r;

/EXEC/ BEGIN
         r := ACOS (1.0);            & result : 0.0
         PRINT (ACOS (r-1.0));       & result : 3.142
       END;
```

ASIN

NAME

ASIN - Arcsine trigonometric function.

SYNTAX

ASIN (*real*)

DESCRIPTION

This function returns a real number which is the trigonometric arcsine of the number defined by argument.

real is an expression representing a REAL type entity.

EVALUATION

During the execution.

NOTE

The returned value is given in radians.

WARNING

The definition domain of the argument is [-1, 1].

EXAMPLE

```
/DECLARE/ REAL r;

/EXEC/ BEGIN
         r := ASIN (0);              & result : 0.0
         r := 1.0;
         PRINT (ASIN (r-2.0));       & result :  -1.571
       END;
```

NAME

ATAN - Arctangent trigonometric function.

SYNTAX

ATAN (*real*)

DESCRIPTION

This function returns a real number which is the trigonometric arctangent of the number defined by argument.

real is an expression representing a REAL type entity.

EVALUATION

During the execution.

NOTE

The returned value is given in radians.

EXAMPLE

```
/DECLARE/ REAL r;

/EXEC/ BEGIN
         r := ATAN (0.0);            & result : 0.0
         r:=1.0;
         PRINT (ATAN (r-2.0));       & result :  -0.7854
       END;
```

COS

NAME

COS - Cosine trigonometric function.

SYNTAX

COS (*real*)

This function returns a real number which is the trigonometric cosine of the number defined by argument.

real is an expression representing a REAL type entity.

EVALUATION

During the execution.

NOTE

The argument must be given in radians.

EXAMPLE

```
/DECLARE/ REAL r;

/EXEC/ BEGIN
         r := COS (0);           & result : 1.0
         r:= 3.1415/2;
         PRINT (COS (2*r));      & result : -1.0
       END;
```

NAME

EXPO - Exponential function.

SYNTAX

EXPO (*real*)

DESCRIPTION

This function returns a real value representing the exponential of the real defined by argument.

real is an expression representing a REAL type entity.

EVALUATION

During the execution.

SEE ALSO

LOG

EXAMPLE

```
/DECLARE/ REAL r;

/EXEC/ BEGIN
         r := EXPO (2.3);              & result : 09.9742
         PRINT (EXPO (r/2));           & result : 146.51
       END;
```

FIX

NAME

FIX - Integer part of a real number.

SYNTAX

FIX (*real*)

DESCRIPTION

This function returns a real value which is the integer part of the real defined by argument.

real is an expression representing a REAL type entity.

EVALUATION

During the execution.

SEE ALSO

INTREAL

EXAMPLE

```
/DECLARE/ REAL r;

/EXEC/ BEGIN
         r := FIX(2.3);          & result : 2.0
         PRINT (FIX(2*r));       & result : 4.0
       END;
```

NAME

INTREAL - Integer value of a real number.

SYNTAX

INTREAL (*real*)

DESCRIPTION

This function returns an integer value which is the integer part of the real number defined by argument.

real is an expression representing a REAL type entity.

EVALUATION

During the execution.

SEE ALSO

FIX - INTROUND

EXAMPLE

```
/DECLARE/ REAL r=2.6;
          INTEGER i;

/EXEC/ BEGIN
         i :=  INTREAL (3.4);              & result : 3
         PRINT (INTREAL (r*2));            & result : 5
       END;
```

INTROUND

NAME

INTROUND - Nearest integer to a real number.

SYNTAX

INTROUND (*real*)

DESCRIPTION

This function returns an integer value which is the nearest integer to the real value defined by argument.

real is an expression representing a REAL type entity.

EVALUATION

During the execution.

SEE ALSO

INTREAL

EXAMPLE

```
/DECLARE/ REAL r=2.6;
          INTEGER i;

/EXEC/ BEGIN
         i :=  INTROUND (3.8);            & result : 4
         PRINT (INTREAL (r*2));           & result : 5
       END;
```

NAME

LOG - Natural logarithm function.

SYNTAX

LOG (*real*)

DESCRIPTION

This function returns a real value which is the natural logarithm of the real number defined by argument.

real is an expression representing a REAL type entity.

EVALUATION

During the execution.

SEE ALSO

LOG10 - EXPO

EXAMPLE

```
/DECLARE/ REAL r;

/EXEC/ BEGIN
         r := LOG (2.3);            & result : 0.8329
         PRINT (LOG (2 * r));       & result : 0.5103
       END;
```

LOG10

NAME

LOG10 - Logarithm to base 10 function.

SYNTAX

LOG10 (*real*)

DESCRIPTION

This function returns a real value which is the logarithm to base 10 of the real number defined by argument.

real is an expression representing a REAL type entity.

EVALUATION

During the execution.

SEE ALSO

LOG

EXAMPLE

```
/DECLARE/ REAL r;

/EXEC/ BEGIN
         r := LOG10 (2.3);            & result :  0.3617
         PRINT (LOG10 (2 * r));       & result :  -0.1406
       END;
```

NAME

MAX - Maximum of several values.

SYNTAX

MAX (*integer1* | *real1*, *integer2* | *real2*, ...)

DESCRIPTION

This function returns a value which is the maximum of all the values defined by the arguments.

The returned value is an integer if all the arguments are integers. It is a real number if one or more values are of real numbers.

Each argument may be an expression representing a REAL or INTEGER type entity.

EVALUATION

During the execution.

SEE ALSO

MIN

EXAMPLE

```
/DECLARE/ INTEGER i = 22;
          REAL r;

/EXEC/ BEGIN
         r := MAX (10.1, 2, 4, 7);            & result : 10.1
         PRINT (MAX (2* i, i + 10, 30));      & result : 44
       END;
```

MIN

NAME

MIN - Minimum of several values.

SYNTAX

MIN (*integer1* | *real1*, *integer2* | *real2*, ...)

DESCRIPTION

This function returns a value which is the minimum of all the values defined by the arguments.

The returned value is an integer if all the arguments are integers. It is a real number if one or more values are of real numbers.

Each argument may be an expression representing a REAL or INTEGER type entity.

EVALUATION

During the execution.

SEE ALSO

MAX

EXEMPLE

```
/DECLARE/ REAL r;
          INTEGER i = 22;

/EXEC/ BEGIN
         r := MIN (1.1, 2, 4, 7);               & resultat : 1.1
         PRINT ( MIN (2* i ,i + 10, 30));       & resultat : 30
       END;
```

NAME

MOD - Modulo between two integers (remainder from their division).

SYNTAX

MOD (*integer1, integer2*) ;

DESCRIPTION

This function returns an integer value which is the remainder from the division of the two integers defined by arguments.

EVALUATION

During the execution.

WARNING

An error is detected during execution if the divisor is equal to zero.

SEE ALSO

Operator of division (/).

EXAMPLE

```
/DECLARE/ INTEGER I, J, K;
/EXEC/ BEGIN
         I := 9;
         J := 4;
         K := MOD (I, J);
         IF (K<>1) THEN PRINT ("ANOMALY");
       END;
```

REALINT

NAME

REALINT - Real value of an integer.

SYNTAX

REALINT (*integer*)

DESCRIPTION

This function returns a real value which is the real value of the integer defined by argument.

integer is an expression representing an INTEGER type entity.

EVALUATION

During the execution.

SEE ALSO

INTREAL

EXAMPLE

```
/DECLARE/ REAL r;
          INTEGER i = 3;

/EXEC/ BEGIN
         r := REALINT (2);              & result : 2.0
         PRINT (REALINT (2* i));        & result : 6.0
       END;
```

NAME

SIN - Sine trigonometric function.

SYNTAX

SIN (*real*)

DESCRIPTION

This function returns a real number which is the trigonometric sine of the number defined by argument.

real is an expression representing a REAL type entity.

EVALUATION

During the execution.

NOTE

The argument must be given in radians.

EXAMPLE

```
/DECLARE/ REAL r;

/EXEC/ BEGIN
         r := SIN (0);              & result : 0.0
         r := 3.1415/2;
         PRINT (SIN (3* r));        & result : -1.0
       END;
```

SQRT

NAME

SQRT - Squared root function.

SYNTAX

SQRT (*real*)

DESCRIPTION

This function returns a real value which is the squared root of the real number defined by argument.

real is an expression representing a REAL type entity.

EVALUATION

During the execution.

WARNING

real should be positive or null.

EXAMPLE

```
/DECLARE/ REAL r;

/EXEC/ BEGIN
         r := SQRT (1.0);              & result : 1.0
         PRINT (SQRT (4*r));           & result : 2.0
       END;
```

NAME

TAN - Tangent trignometric function.

SYNTAX

TAN (*real*)

DESCRIPTION

This function returns a real number which is the trigonometric tangent of the real number defined by argument.

real is an expression representing a REAL type entity.

EVALUATION

During the execution.

NOTE

The argument must be given in radians.

EXAMPLE

```
/DECLARE/ REAL r;

/EXEC/ BEGIN
         r := TAN (0);                  & result : 0.0
         r := 3.1415/4;
         PRINT (TAN (3* r));            & result : -1.0
       END;
```

7.3.3 Random numbers

COX	`COX` - Work demand distributed according to a Cox law.
CST	Constant work demand.
DISCRETE	Drawing of a random number among a list of homogeneous elements according to a table of probabilities.
DRAW	Drawing of a boolean value according a probability.
ERLANG	Work demand distributed according to an Erlang law.
EXP	Work demand distributed according to an exponential law.
HEXP	Work demand distributed according to an hyper-exponential law.
HISTOGR	Random number generation according to an histogram.
NORMAL	Random number generation according to a Gaussian law.
RANDU	Random number uniformly distributed generation.
RINT	Integer random number uniformly distributed generation.
UNIFORM	Work demand uniformly distributed.

NAME

COX - Work demand distributed according to a Cox law.

SYNTAX

COX (*real-array* | *real-list*) ;

DESCRIPTION

This function, which can also be used as a procedure to request time consumption, computes a random value according to a Cox law. Characteristics of the Cox law are given by a list or array of real values giving the mean and probability of each exponential stage (list or array of (m_n, p_n) where m_n is the mean of the exponential stage and p_n is the probability to access to the n+1 stage).

EVALUATION

During execution.

NOTES

A N-stage Cox law will require a 2*N size array or list of reals.

WARNING

- The random number generator is reinitialised only at each /EXEC/ section.
- The procedure can only be used within a service description, whereas the function can be used in any algorithmic code.
- The last probability should be equal to zero (the probability to access to a stage greater than the number of stages of the law is zero).

EXAMPLE

```
/DECLARE/  REAL s,r(6) = (1.0, 0.3, 2.0, 0.2, 3.0, 0.0);
           QUEUE q ;

/STATION/ NAME = q;
          SERVICE = BEGIN
                      ...
                      COX ((1.0, 0.5, 2.0, 0.0));
                      ...
                      COX (r);
                      s := COX ((1.0, 0.5, 2.0, 0.0));
                    END;
```

CST

NAME

CST - Constant work demand.

SYNTAX

CST (*real*)

DESCRIPTION

This procedure defines a constant work demand whose duration is defined by the argument of the procedure.

real is an expression or constant evaluating to REAL.

EVALUATION

During execution.

NOTES

The random number generator is not called for this procedure.

WARNING

- This procedure can only be used within a service description.
- *real* must be positive.

EXAMPLE

```
/DECLARE/  REAL r = 3.4;
           QUEUE q;

/STATION/  NAME = q;
           SERVICE = BEGIN
                       ...
                       CST (5.4);
                       ...
                       CST (2 * r);
                       ...
                     END;
```

NAME

DISCRETE - Drawing of a random number among a list of homogeneous elements according to a table of probabilities.

SYNTAX

DISCRETE (*list* | *array, list* | *table-of-reals* [, *integer*])

DESCRIPTION

This function returns a value which is amongst a set of values with the same type according to a table of probabilities.

The first argument is a list (or an array) of elements of any type (scalar, QUEUE, CLASS, ... or user defined) in which a drawing will be done in order to select one of these elements.

The second argument is a list (or an array) of reals which represents the set of probabilities used for the random drawing. The i^{th} value is the probability of drawing the i^{th} element in the list (or array) defined by the first argument. If the sum of the probabilities is not equal to 1 then the values are normalized at 1.

The optional third argument represents the number of values to take into account during the drawing. By default, this number is the dimension of the first argument.

integer is an expression representing a INTEGER type entity.

EVALUATION

During the execution.

NOTE

The normalisation of the probabilities is based on the probabilities taken into account and defined by the optional third argument.

EXAMPLE

```
/DECLARE/ OBJECT gamme;
            INTEGER nbphase;
          END;

          REF gamme rg;
          gamme g1, g2, g3;

          INTEGER i = 2;
          REAL    r (3) = (0.5, 0.3, 0.2);
          STRING  s (3) = ("a", "b", "c");

/EXEC/ BEGIN
         rg := DISCRETE( (g1, g2, g3), (0.5, 0.3, 0.2) );
         & value g1 generated with probability 0.5
         & value g2 generated with probability 0.3
         & value g3 generated with probability 0.2
         PRINT (DISCRETE (s, r, i));
         & value "a" generated with probability 0.625
         & value "b" generated with probability 0.375
       END;
```

NAME

DRAW - Drawing of a boolean value according a probability.

SYNTAX

DRAW (*real*)

DESCRIPTION

This function returns a boolean value. This value is TRUE with a probability equal to the value defined by argument. This value is FALSE with a probability equal to the complement to 1 of the value defined by argument.

real is an expression representing a REAL type entity.

EVALUATION

During the execution.

WARNING

- The random numbers generator is reinitialized at the beginning of each execution time phase (/EXEC/ command).
- The definition domain for the argument is [0.0, 1.0]

SEE ALSO

RANDOM

EXAMPLE

```
/DECLARE/ REAL    r = 0.7;
          BOOLEAN b;

/EXEC/ BEGIN
         b := DRAW (0.3);
         PRINT(b);

         IF DRAW (r) THEN PRINT ("TRUE")
                     ELSE PRINT ("FALSE");
       END;
```

ERLANG

NAME

ERLANG - Work demand distributed according to an Erlang law.

SYNTAX

ERLANG (*real*, *integer*)

DESCRIPTION

This function, which can also be used as a procedure to request time consumption, computes a random value according to an Erlang law. Characteristics of the Erlang law are given by its mean, *real* argument, and its order, *integer* argument.

real is an expression or constant evaluating to REAL.

integer is an expression or constant evaluating to INTEGER.

EVALUATION

During execution.

NOTES

The order is the inverse of the square of the variance, it is the square of the mean divided by the variance.

WARNING

- The random number generator is reinitialised only at each /EXEC/ section.
- The procedure can only be used within a service description, whereas the function can be used in any algorithmic code.
- Both arguments must be positive.

EXAMPLE

```
/DECLARE/  REAL r,s = 3.2;
           QUEUE q;

/STATION/ NAME = q;
          SERVICE = BEGIN
                      ...
                      ERLANG (5.4,2);
                      ...
                      r:= ERLANG (s,2);
                    END;
```

NAME

EXP - Work demand distributed according to an exponential law.

SYNTAX

EXP (*real*)

DESCRIPTION

This function, which can also be used as a procedure to request time consumption, computes a random value according to an exponential law. The argument of the function or procedure is the mean value of the distribution.

real is an expression or constant evaluating to REAL.

EVALUATION

During execution.

WARNING

- The random number generator is reinitialised only at each /EXEC/ section.
- The procedure can only be used within a service description, whereas the function can be used in any algorithmic code.
- The argument must be positive.

EXAMPLE

```
/DECLARE/  REAL s,r = 2.3;
           QUEUE q ;

/STATION/ NAME = q;
          SERVICE = BEGIN
                      ...
                      EXP (0.5);
                      ...
                      EXP (5*r);
                      s := EXP (2.3);
                    END;
```

HEXP

NAME

HEXP - Work demand distributed according to an hyper-exponential law.

SYNTAX

HEXP (*real1*, *real2*)

DESCRIPTION

This function, which can also be used as a procedure to request time consumption, computes a random value according to an hyper-exponential law. Characteristics of the hyper-exponential law are given by its mean, *real1*, and the square of the coefficient of variation, *real2*.

real1 and *real2* are expressions or constants evaluating to REAL.

EVALUATION

During execution.

NOTES

The square of the coefficient of variation is the variance divided by the square of the mean.

WARNING

- The random number generator is reinitialised only at each /EXEC/ section.
- The procedure can only be used within a service description, whereas the function can be used in any algorithmic code.
- Both arguments must be positive.

EXAMPLE

```
/DECLARE/ REAL r = 3.4, s = 2.5;
          QUEUE q;

/STATION/  NAME = q ;
           SERVICE = BEGIN
                       ...
                       HEXP (5.4, 4.5);
                       ...
                       HEXP (2*r , 2*s));
                       ...
                       s:= HEXP (5.3,2.0);
                     END ;
```

NAME

HISTOGR - Random number generation according to an histogram.

SYNTAX

HISTOGR (*real-list1* | *real-array1, real-list2* | *real-array2* [, *integer*])

DESCRIPTION

This function computes a random number according to a distribution given by an histogram.

The histogram has contiguous bars. Bars are characterised by a list or an array of real values giving the bounds of the intervals (*real-list1* or *real-array1*), and a list or array of probabilities for each interval (*real-list2* or *real-array2*). The size of *list1* or *real-array1* is the number of intervals plus 1. If the sum of each value of the second argument is not equal to 1, probalilities are calculated by normalising the values. The third argument represents the number of intervals which should be considered to compute the random value. The default value for this thrid argument is the size of the second argument (number of intervals in the histogram).

Computation of the random value is achieved like described above:

- a first step computes, according to the probabilities of each interval, the interval into which the result falls,
- then a uniform generation computes the value inside the selected interval.

real-list1, *real-list2*, *real-array1* and *real-array2* are expressions or constants evaluating to respectively a list or an array of REAL.

integer is an expression or a constant evaluating to INTEGER.

EVALUATION

During execution.

WARNING

Normalisation of the probabilities is achieved on the number of intervals specified by the third argument if present.

EXAMPLE

```
/DECLARE/  REAL r1 (4) = (0.0, 1.5, 3.0, 4.5) ,
                r2 (3) = (0.5, 0.3, 0.2), r ;
           INTEGER  i = 2 ;

/EXEC/ BEGIN
         r : =  HISTOGR (r1, (0.5, 0.3, 0.2)) ;
         & value falling into interval with probability
         &                      [0.0, 1.5]         0.5
         &                      [1.5, 3.0]         0.3
         &                      [3.0, 4.5]         0.2

         PRINT ( HISTOGR ((0.0, 1.5, 3.0, 4.5), r2, i)) ;
         & value falling into interval with probability
         &                      [0.0, 1.5]         0.625
         &                      [1.5, 3.0]         0.375
       END;
```

NAME

NORMAL - Random number generation according to a Gaussian law.

SYNTAX

NORMAL (*real1, real2*)

DESCRIPTION

This function computes a random number according to a Gaussian distribution. The mean of the distribution is given by *real1* and the standard deviation is given by *real2*.

real1 and *real2* are expressions or constants evaluating to REAL.

EVALUATION

During execution.

WARNING

The random number generator is reinitialised only at each /EXEC/ section.

SEE ALSO

RANDOM

EXAMPLE

```
/DECLARE/ REAL r, s = 5.09;

/EXEC/ BEGIN
       r := NORMAL (5.3, 2.0);
       PRINT ( NORMAL (2* r, s/2));
       END;
```

RANDU

NAME

RANDU - Random number uniformly distributed generation.

SYNTAX

RANDU

DESCRIPTION

This function computes a random number uniformly distributed between 0 and 1. The result is a REAL value.

EVALUATION

During execution.

WARNING

The random number generator is reinitialised only at each /EXEC/ section.

SEE ALSO

RANDOM

EXAMPLE

```
/DECLARE/ REAL r;

/EXEC/ BEGIN
         r :=  RANDU;
         PRINT (RANDU);
       END;
```

NAME

RINT - Integer random number uniformly distributed generation.

SYNTAX

RINT (*integer1, integer2*)

DESCRIPTION

This function computes a random number uniformly distributed between *integer1* and *integer2*.

integer1 and *integer2* are expressions or constants evaluating to INTEGER.

EVALUATION

During execution.

WARNING

The random number generator is reinitialised only at each /EXEC/ section.

SEE ALSO

RANDOM

EXAMPLE

```
/DECLARE/ INTEGER i, j = 20;

/EXEC/ BEGIN
         i :=  RINT (5, 13);
         PRINT (RINT (3* i, j* 2));
       END;
```

UNIFORM

NAME

UNIFORM - Work demand uniformly distributed.

SYNTAX

UNIFORM (*real1*, *real2*) ;

DESCRIPTION

This function, which can also be used as a procedure to request time consumption, computes a random value uniformly distributed between *real1* and *real2*.

real1 and *real2* are expressions or constants evaluating to REAL.

EVALUATION

During execution.

WARNING

- The random number generator is reinitialised only at each /EXEC/ section.
- The procedure can only be used within a service description, whereas the function can be used in any algorithmic code.
- *real1* and *real2* must be positive and *real1* must be lower than *real2* .

EXAMPLE

```
/DECLARE/  REAL s,r = 2.3;
           QUEUE q ;

/STATION/ NAME = q;
          SERVICE = BEGIN
                      ...
                      UNIFORM (5.4, 7.5);
                      ...
                      s := UNIFORM (2* r, 2* r);
                    END;
```

7.3.4 Strings Management

CHARCODE	Access to the internal code of a character.
CHREXCLUDE	Search for the first character of a string excluded from another string.
CHRFIND	Search for the first character of a string included in another string.
INDEX	Search for a substring in a string.
LTRIM	Substring deletion ahead.
REVERSE	String inversion.
RTRIM	String deletion at the end.
STRLENGTH	Effective size of a string.
STRMAX	Maximum size of a string.
STRREPEAT	String repetition.
SUBSTR	Substring in a string.
TRANSLATE	String translation.
WRCHCODE	Writing a character defined by its internal code in a string.
WRSUBSTR	Substitution of a substring in a string.

CHARCODE

NAME

CHARCODE - Access to the internal code of a character.

SYNTAX

CHARCODE(*string, integer*)

DESCRIPTION

This function returns an integer which is the internal code (ASCII or EBCDIC depending on the computer) of the character in the string given by the first argument whose position is given by the second argument. The value of the second argument has to be less than or equal to the effective size of the string.

string is an expression representing a STRING type entity.

integer is an expression representing a INTEGER type entity.

EVALUATION

During the execution.

WARNING

The internal code depends on the computer, in general ASCII is used.

SEE ALSO

STRING

EXAMPLE

```
/DECLARE/ STRING S;
          INTEGER I;
/EXEC/ BEGIN
        PRINT("Execution of the program produces :");
        PRINT(" ");
        PRINT("R1 : ",CHARCODE("AB",1));
        S := "HELLO";
        I := 3;
        PRINT("R2 : ",CHARCODE(S,I));
        PRINT("R3 : ",CHARCODE("21"//"43",5-2));
       END;
```

```
Execution of the program produces:
 R1 :        65        (ASCII)        193        (EBCDIC)
 R2 :        76        (ASCII)        211        (EBCDIC)
 R3 :        52        (ASCII)        244        (EBCDIC)
```

NAME

CHREXCLUDE - Search for the first character of a string excluded from another string.

SYNTAX

CHREXCLUDE(*string1, string2*)

DESCRIPTION

This function returns an integer which is the position of the first character in the string defined by the first argument excluded from the second string. If a character is found, then the value returned is zero. This function may be used for instance to find the first character in a string which is not a number (or a letter).

string1 and *string2* are expression representing STRING type entity.

EVALUATION

During the execution.

NOTE

If the research has failed the function returns 0.

SEE ALSO

STRING - CHRFIND

EXAMPLE

```
/DECLARE/ STRING S1,S2;
/EXEC/ BEGIN
        PRINT("Execution of the program produces :");
        PRINT(" ");
        PRINT("R1 : ",CHREXCLUDE("1234A56","1234567890"));
        S1 := "123A4";
        S2 := "12345";
        PRINT("R2 : ",CHREXCLUDE(S1,S2));
        PRINT("R3 : ",CHREXCLUDE("12"//"34","13"//"24"));
       END;
```

Execution of the program produces :

```
R1 :         5
R2 :         4
R3 :         0
```

CHRFIND

NAME

CHRFIND - Search for the first character of a string included in another string.

SYNTAX

CHRFIND(*string1, string2*)

DESCRIPTION

The function returns an integer which is the position of the first character in the string defined by the first argument also contained in the second string. If a character is found, then the value returned is zero. For example, this function may be used to find in a string the first character which is a number (or a letter).

string1 and *string2* are expression representing STRING type entity.

EVALUATION

During the execution.

NOTE

If the research has failed the function returns 0.

SEE ALSO

STRING - CHREXCLUDE

EXAMPLE

```
/DECLARE/ STRING S1,S2;
/EXEC/ BEGIN
        PRINT("Execution of the program produce :");
        PRINT(" ");
        PRINT("R1 : ",CHRFIND("ABC2EF","0123456789"));
        S1 := "123A4";
        S2 := "ABCD";
        PRINT("R2 : ",CHRFIND(S1,S2));
        PRINT("R3 : ",CHRFIND(S2,S1));
        PRINT("R4 : ",CHRFIND("12"//"34","AB"//"CD"));
       END;
```

Execution of the program produces :

```
R1 :         4
R2 :         4
R3 :         1
R4 :         0
```

NAME

INDEX - Search for a substring in a string.

SYNTAX

INDEX(*string1, string2*)

DESCRIPTION

This function returns an integer which is the position of the first occurence of the substring (second argument) in the string given in first argument. The produced integer cannot exceed the effective size of the first string.

string1 and *string2* are expression representing STRING type entities.

EVALUATION

During the execution.

NOTES

- If the research has failed the function returns 0.
- The function cann't return a value superior to the length of the string given in argument.

SEE ALSO

STRING

EXAMPLE

```
/DECLARE/ STRING S1,S2;

/EXEC/ BEGIN
        PRINT("Execution of the program produces :");
        PRINT(" ");
        PRINT("R1 : ",INDEX("123","4"));
        S1 := "123423";
        S2 := "23";
        PRINT("R2 : ",INDEX(S1,S2));
        PRINT("R3 : ",INDEX(S2//"45","3"//"4"));
       END;
```

Execution of the program produces :

```
R1 :          0
R2 :          2
R3 :          2
```

LTRIM

NAME

LTRIM - Substring deletion ahead.

SYNTAX

LTRIM(*string1[, string2]*)

DESCRIPTION

This function returns a string which is the string defined by the first argument where the substring given by the second argument is deleted from the beginning of the first string, to the extent possible.

string1 and *string2* are expressions representing STRING type entities.

EVALUATION

During the execution.

NOTES

If the second argument is omitted, then the default is one blank character.

WARNING

The result of the function is cut if it is returned in to short string.

SEE ALSO

RTRIM - STRING

EXAMPLE

```
/DECLARE/ STRING S1,S2;
/EXEC/ BEGIN
        PRINT("Execution of the program produces :");
        PRINT(" ");
        PRINT("R1 : ",LTRIM("  12"));
        PRINT("R2 : ",LTRIM("121234","12"));
        S1 := "ABAAA";
        S2 := "A";
        PRINT("R3 : ",LTRIM(S1,S2));
        PRINT("R1 : ",LTRIM("12"//"34","1"//"23"));
       END;
```

```
Execution of the program produces:
 R1 : 12
 R2 : 34
 R3 : BAAA
 R1 : 4
```

NAME

REVERSE - String inversion.

SYNTAX

REVERSE(*string*)

DESCRIPTION

This function returns a string with the same size as the size of the string argument with the entries in the reversed string .

string is an expression representing a STRING type entity.

EVALUATION

During the execution.

NOTE

The length of the returned string is the same than the length of the argument string.

SEE ALSO

STRING

EXAMPLE

```
/DECLARE/ STRING S;

/EXEC/ BEGIN
        PRINT("Execution of the program produces :");
        PRINT(" ");
        PRINT("R1 : ",REVERSE("OK"));
        S := "AVE";
        PRINT("R2 : ",REVERSE(S));
        PRINT("R3 : ",REVERSE("CO"//"OL"));
       END;
```

Execution of the program produces :

```
R1 : KO
R2 : EVA
R3 : LOOC
```

RTRIM

NAME

RTRIM - String deletion at the end.

SYNTAX

RTRIM(*string1[, string2]*)

DESCRIPTION

This function returns a string which is the string defined by the first argument where the substring given by the second argument is deleted from the end of the first to the extent possible. If the second argument is omitted, then the default is one blank character.

string1 and *string2* are expressions representing STRING type entities.

EVALUATION

During the execution.

NOTES

If the second argument is omitted, then the default is one blank character.

WARNING

The result of the function is cut if it is returned in to a short string.

SEE ALSO

LTRIM - STRING

EXAMPLE

```
/DECLARE/ STRING S1,S2;

/EXEC/ BEGIN
        PRINT("Execution of the program produces :");
        PRINT(" ");
        PRINT("R1 : ",RTRIM("12  "));
        PRINT("R2 : ",RTRIM("341212","12"));
        S1 := "AAABA";
        S2 := "A";
        PRINT("R3 : ",RTRIM(S1,S2));
        PRINT("R1 : ",RTRIM("41"//"23","12"//"3"));
       END;
```

Execution of the program produces :

```
R1 : 12
R2 : 34
R3 : AAAB
R1 : 4
```

STRLENGTH

NAME

STRLENGTH - Effective size of a string.

SYNTAX

STRLENGTH(*string*)

DESCRIPTION

This function returns an integer which is the effective size of the string given the argument. The value returned is the size of the data stored at the last modification. All characters in the string are taken into account, including blanks at the end of the string.

string is an expression representing a STRING type entity.

EVALUATION

During the execution.

NOTE

Blanck characters are significant even at the end of the string.

SEE ALSO

STRMAXL - STRING

EXAMPLE

```
/DECLARE/ STRING(10) S1;
/EXEC/ BEGIN
        PRINT("Execution of the program produces :");
        PRINT(" ");
        PRINT("R1 : ",STRLENGTH(S1));
        S1 := "ABC";
        PRINT("R2 : ",STRLENGTH(S1));
        S1 := "ABC ";
        PRINT("R3 : ",STRLENGTH(S1));
        PRINT("R4 : ",STRLENGTH("ET"));
        PRINT("R5 : ",STRLENGTH("ET"//S1));
       END;

Execution of the program produces:
 R1 :         0
 R2 :         3
 R3 :         4
 R4 :         2
 R5 :         6
```

NAME

STRMAXL - Maximum size of a string.

SYNTAX

STRMAXL(*string*)

DESCRIPTION

This function returns an integer which is the maximum size of the string given in the argument.

string is an expression representing a STRING type entity.

EVALUATION

During the execution.

NOTES

- Blank characters are significant even at the end of the string.
- The returned value is defined by the declaration of the variable or of the string. For a variable declared without size, the value returned is 80. If a variable is modified after the declaration, its maximum size does not change.

SEE ALSO

STRLENGTH - STRING

EXAMPLE

```
/DECLARE/ STRING(10) S1;
          STRING S2;

/EXEC/ BEGIN
        PRINT("Execution of the program produces :");
        PRINT(" ");
        PRINT("R1 : ",STRMAXL("AB"));
        PRINT("R2 : ",STRMAXL(S1));
        PRINT("R3 : ",STRMAXL(S2));
        S1 := " AB";
        PRINT("R4 : ",STRMAXL(S1));  & the maximum size does not change
        PRINT("R5 : ",STRMAXL("1"//S1));
       END;
```

Execution of the program produces :

```
R1 :          2
R2 :         10
R3 :         80
R4 :         10
R5 :          4
```

NAME

STRREPEAT - String repetition.

SYNTAX

STRREPEAT(*string, integer*)

DESCRIPTION

This function returns the string given by the first argument repeated the number of times specified by the integer defined in the second argument. Use of this function makes printing of reports easier.

string is an expression representing a STRING type entity.

integer is an expression representing a INTEGER type entity.

EVALUATION

During the execution.

WARNING

- The number of repetitions has to be greater than zero.
- The result of the function is cut if it is returned into a too short string.

SEE ALSO

STRING

EXAMPLE

```
/DECLARE/ STRING S;
          INTEGER I;

/EXEC/ BEGIN
        PRINT("Execution of the program produces :");
        PRINT(" ");
        PRINT("R1 : ",STRREPEAT("*",10));
        S := "123";
        I := 2;
        PRINT("R2 : ",STRREPEAT(S,I));
        PRINT("R3 : ",STRREPEAT(S//"4",I+1));
       END;
```

Execution of the program produces :

```
R1 : **********
R2 : 123123
R3 : 123412341234
```

NAME

SUBSTR - Substring in a string.

SYNTAX

SUBSTR(*string, integer1, integer2*)

DESCRIPTION

This function returns the substring in a string (first argument) between the first position defined by the second argument and the last position given by the third argument. Each argument may be a constant, a variable, a function call or an expression.

string is an expression representing a STRING type entity.

integer1 and *integer2* are expressions representing INTEGER type entities.

EVALUATION

During the execution.

WARNING

- The first position has to be less than or equal to the last position and greater than zero.
- The last position has to be less than or equal to the effective size of the string and greater than zero.

SEE ALSO

STRING

EXAMPLE

```
/DECLARE/ STRING(20) S1,S2;
          INTEGER I1,I2;

/EXEC/ BEGIN
        PRINT("Execution of the program produces :");
        PRINT(" ");
        S1 := "1234567";
        PRINT("R1 : ",SUBSTR(S1,1,7));
        I1 := 2; I2 := 5;
        PRINT("R2 : ",SUBSTR(S1,I1,I2));
        PRINT("R3 : ",SUBSTR("1234",2,3));
        PRINT("R4 : ",SUBSTR(S1//"89",I1+4,I2+3));
       END;
```

Execution of the program produces :

```
R1 : 1234567
R2 : 2345
R3 : 23
R4 : 678
```

NAME

TRANSLATE - String translation.

SYNTAX

TRANSLATE(*string1, string2, string3*)

DESCRIPTION

This function returns a string built following the principle:

- each character of the first string excluded from the second string is stored in the new string at the same position
- each character of the first string included in the second string is replaced in the third string at the same position by the character held in the third string at the same position as the first occurence of the character sought in the second string.

Each argument may be a constant, a variable, a function call or an expression. The size of the third string has to be greater than or equal to the size of the second string which must be greater than zero (size = effective size).

EVALUATION

During the execution.

NOTES

- The returned string has the same size as the size of the first string.
- For example, this function may be used to transform uppercase characters to lowercase characters.

SEE ALSO

STRING

EXAMPLE

```
/DECLARE/ STRING S1,S2,S3;
/EXEC/ BEGIN
        PRINT("Execution of the program produces :");
        PRINT(" ");
        PRINT("R1 : ",TRANSLATE("AICDEFG","BERFQDOLAZC","TLMOBUZASDM"));
        S1 := "JULES CAESAR";
        S2 := "ABDEFGHIKLMNOPQRSTUVWXYZ";
        S3 := "abdefghiklmnopqrstuvwxyz";
        PRINT("R2 : ",TRANSLATE(S1,S2,S3));
        PRINT("R3 : ",TRANSLATE("AB"//"CD","ECB"//"FG","12"//"345"));
       END;
```

Execution of the program produces :

```
R1 : SIMULOG
R2 : Jules Caesar
R3 : A32D
```

NAME

WRCHCODE - Writing a character defined by its internal code in a string.

SYNTAX

WRCHCODE(*string, integer1, integer2*)

DESCRIPTION

This procedure changes the content of the string given by the first argument as follows: the character whose the position is defined by the second argument is replaced by the character whose the internal code is given by the third argument (code ASCII or EBCDIC depending on the computer).

The first argument has to be a variable. For the two last arguments, each may be a constant, a variable, a function call or an expression.

string is an expression of STRING type entity.

integer1 and *integer2* are expressions representing INTEGER type entities.

EVALUATION

During the execution.

WARNING

- The internal code depends on the computer, in general ASCII is used.
- The last position has to be less than the effective size of the string and greater than zero.

SEE ALSO

CHARCODE - STRING

EXAMPLE

```
/DECLARE/ STRING S;
          INTEGER POSI,CODE;

/EXEC/ BEGIN
        PRINT("Execution of the program produces :");
        PRINT(" ");
        S := "WHy";
        WRCHCODE(S,3,89);           & ASCII code for Y
        PRINT("R1 : ",S);
        S := "WHY !";
        POSI := 5;
        CODE := 63;                 & ASCII code for ?
        WRCHCODE(S,POSI,CODE);
        PRINT("R2 : ",S);
        S := "NUMBER :  ";
        WRCHCODE(S,5*2,48+6);       & ASCII code for 6
        PRINT("R3 : ",S);
       END;
```

Execution of the program produces :

```
R1 : WHY
R2 : WHY ?
R3 : NUMBER : 6
```

NAME

WRSUBSTR - Substitution of a substring in a string.

SYNTAX

WRSUBSTR(*string1, string2, integer1, integer2*)

DESCRIPTION

This procedure changes the contents of the string given by the first argument as follows: the substring defined by the first position (third argument) and the last position (fourth argument) in the first string is replaced by the second string.

The first argument has to be a variable. For the three last arguments, each may be a constant, a variable, a function call or an expression. The last position has to be greater than or equal to the first position. It should be noted that the effective size of the first string may be changed if the effective size of the second string is different from the value "last position-first position+1."

string1 and *string2* are expressions representing STRING type entites.

integer1 and *integer2* are expressions representing INTEGER type entities.

EVALUATION

During the execution.

WARNING

- The first argument has to be a STRING typed variable.
- The first position has to be less than to the first position and greater than zero.
- The last position has to be greater than or equal to the first position and greater than zero.

SEE ALSO

STRING

EXAMPLE

```
/DECLARE/ STRING S1,S2;
          INTEGER PREM,DERN;

/EXEC/ BEGIN
        PRINT("Execution of the program produces :");
        PRINT(" ");
        S1 := "OK !";
        WRSUBSTR(S1," JUNIOR",3,3);
        PRINT("R1 : ",S1);
        S1 := "TEX HARODY";
        S2 := "AVER";
        PREM := 5;
        DERN := 9;
        WRSUBSTR(S1,S2,PREM,DERN);
        PRINT("R2 : ",S1);
        S1 := "NUMBER 1";
        WRSUBSTR(S1,"ON"//"E",4*2,4+4);
        PRINT("R3 : ",S1);
       END;
```

Execution of the program produces :

```
R1 : OK JUNIOR!
R2 : TEX AVERY
R3 : NUMBER ONE
```

7.3.5 Files Management

AUDIT	Creation of an audition of a file on one or several other files.
CLOSE	Close a file.
FAUDITED	Access to an audited file from a given auditor file.
FAUDITOR	Access to an auditor file from a given audited file.
FILASSIGN	File allocation.
FILSETTER	Recovery level definition.
GET	Input handling functions.
ISAUDTED	Tests for the existence of an auditor-audited link between two files
NBAUDITED	Access to the number of audited files linked to a
NBAUDITOR	Access to the number of audited files linked to a given auditor file.
OPEN	File opening.
RESTORE	Restoring processing.
SAVERUN	Library saving.
SETBUF	Buffer length definition.
SETRETRY	Definition of the number of trials.
SETSYN	defining synonyms.
UNAUDIT	End of audition of a file on one or several other files.
WRITE	Write operations.

AUDIT

NAME

AUDIT - Creation of an audition of a file on one or several other files.

SYNTAX

AUDIT(*file1, file2, fileN [,string] [,integer]*)

DESCRIPTION

This procedure defines an audition operation. All the input/output orders concerning the file given as first argument (audited) will be copied to other files given in argument (auditors)

The *string* parameter represents the prefix for all recordings from the audited file placed in the auditor files. By default, the present prefix is the name of the audited file. The parameter integer gives the format length used by the record number included in the auditors files. This number represents the record number in the audited file. By default (zero value assumed), this number is not copied.

The parameter file1 is an audited FILE type entity. The parameters *file2* ...*fileN* are auditors FILE type entities.

string is an expression representing a STRING type entity.

integer is an expression representing a INTEGER type entity.

EVALUATION

During the execution.

NOTE

- An auditor file can be closed before this procedure is called.
- The closure of an audited file destroys the audited-auditor links. A file can be assigned to several different audited files.
- A file can audit more than one file.

WARNINGS

- All auditor files have to be assigned before calling this procedure.
- Any auditor file opened before the procedure is called must be open in write mode.
- The construction of audition loops is forbidden

SEE ALSO

UNAUDIT - ISAUDTED - NBAUDTED - NBAUDTOR - FAUDITED - FAUDITOR

EXAMPLE

```
/DECLARE/FILE f1, f2, f3;
         INTEGER i=2;

/EXEC/BEGIN
        FILASSIGN (f1,"f1.lis");
        FILASSIGN (f2,"f2.lis");
        FILASSIGN (f3,"f3.lis");
        OPEN (f1,2);
        OPEN (f3,2);
        AUDIT (f1,f2);          & all writes to f1 will
                                & be repeated in f2
                                & with the write head: "f1:"
        AUDIT (f1,f3,"f1",2*i); & all writes to f1 will be
                                & repeated in f3 with the
                                & write head: "f1:write N"
        WRITELN (f3, "Beginning of f3");
        WRITELN(f1,"I am a copy of f1");
        ...
      END;

File contents f2
f1 : I am a copy of f1
File contents f3
Beginning of f3
f1 : I am a copy of f1
```

CLOSE

NAME

CLOSE - Close a file.

SYNTAX

CLOSE(*file[, integer]*)

DESCRIPTION

This procedure closes the file given by the first argument. The closing mode is defined by the second argument. The possible values are:

- < 0: closing and deleting the file
- >= 0: closing and saving the file

file is an expression representing a FILE type entities.

integer is an expression representing a INTEGER type entity.

EVALUATION

During the execution.

NOTE

- It modifies the file attribute OPENMODE.
- By default, a file is closed in saving mode (>= 0).

WARNING

- This procedure has to be called only on an opened file.
- It should be noted that the predeclared file FSYSINPU can not be closed; nor can implicit files be closed.

SEE ALSO

FILE - OPEN - OPENMODE

EXAMPLE

```
/DECLARE/ FILE F,G;
          REF FILE RG;
          INTEGER I;

/EXEC/ BEGIN
         ...             & file opening
         CLOSE(F,-1);    & closing with deletion for the file F
         RG := G;
         I := 0;
         OPEN(RG,I);     & closing with safe keeping for the file F
                         & (equivalent to CLOSE(RG)
       END;
```

FAUDITED

NAME

FAUDITED - Access to an audited file from a given auditor file.

SYNTAX

FAUDITED(*file,integer*)

DESCRIPTION

This function returns a reference to an audited file from an auditor file given as first argument. The second argument gives the rank of the sought audited file.

file is an expression representing a FILE type entities.

integer is an expression representing a INTEGER type entity.

EVALUATION

During the execution.

NOTE

- If the rank exceeds the number of audited files (NBAUDTED) the function returns a NIL value.
- If the file expressed in argument is not an auditor file then a NIL value is returned.

SEE ALSO

AUDIT - UNAUDIT - ISAUDTED - NBAUDTED - NBAUDTOR - FAUDITOR

EXAMPLE

```
/DECLARE/FILE f1, f2, f3;
        REF FILE rf;
        INTEGER i=1;

/EXEC/BEGIN
        FILASSIGN (f1,"f1.lis");
        FILASSIGN (f2,"f2.lis");
        OPEN (f1,2);
        AUDIT (f1,f2);
        rf := FAUDITED (f2,i); & the return function here : f1
        ...
        rf := FAUDITED (f2,2); & the return function here : NIL
        ...
        rf := FAUDITED (f1,1); & the return function here : NIL
        ...
      END;
```

NAME

FAUDITOR - Access to an auditor file from a given audited file.

SYNTAX

FAUDITOR(*file, integer*)

DESCRIPTION

This function returns a reference to an auditor file from an audited file given as first argument. The second argument gives the rank of the sought auditor file.

file is an expression representing a FILE type entities.

integer is an expression representing a INTEGER type entity.

EVALUATION

During the execution.

NOTES

- If the rank exceeds the number of audited files (NBAUDTED) the function gives a NIL return.
- If the file expressed in argument is not an auditor file a NIL value is returned.

SEE ALSO

AUDIT - UNAUDIT - ISAUDTED - NBAUDTED - NBAUDTOR - FAUDITED

EXAMPLE

```
/DECLARE/FILE f1, f2, f3;
        REF FILE rf;
        INTEGER i=1;

/EXEC/BEGIN
        FILASSIGN (f1,"f1.lis");
        FILASSIGN (f2,"f2.lis");
        OPEN (f1,2);
        AUDIT (f1,f2);
        rf := FAUDITOR (f1,i); & the return function here : f2
        ...
        rf := FAUDITOR (f1,2); & the return function here : NIL
        ...
        rf := FAUDITED (f2,1); & the return function here : NIL
        ...
      END;
```

FILASSIGN

NAME

FILASSIGN - File allocation.

SYNTAX

FILASSIGN(*file, string*)

DESCRIPTION

This procedure assigns a physical file to the file object given by the first argument. The name of the physical file is defined by the second argument.

file is an expression representing a FILE type entities.

string is an expression representing a STRING type entity.

EVALUATION

During the execution.

NOTE

- The specification of the access to the file depends on the machine.
- This procedure modifies the file attribute FILASSGN.

WARNING

- The file can not be opened before allocation.
- The predeclared files FSYSINPU, FSYSOUTP and FSYSTERM can not be assigned.

SEE ALSO

CLOSE - FILSETTER - OPEN - SETBUF - SETRETRY - SETSYN

EXAMPLE

```
/DECLARE/ FILE F;
          REF FILE RF;
          STRING S;

/EXEC/ BEGIN
        FILASSIGN(F,"file_name");
        S := "file_name";
        RF := F;
        FILASSIGN(RF,S);
       END;
```

NAME

FILSETERR - Recovery level definition.

SYNTAX

FILSETERR(*file, integer*)

DESCRIPTION

This procedure defines the recovery level for the file given by the first argument. The value assigned to the recovery level is defined by the second argument. The possible values are:

- <= 0: no recovery
- = 1: end of file recovery in read
- = 2: recovery for conversion error or error detected by QNAP2
- = 3: recovery for i/o error detected by the operating system or the FORTRAN library

An error is recovered if QNAP2 does not display an error message and continues running.

The user can test an error occurence by the ERRSTATUS attribute of the specified file. This attribute is update at each operation on the file.

file is an expression representing a FILE type entities.

integer is an expression representing a INTEGER type entity.

EVALUATION

During the execution.

NOTE

This procedure modifies the file attribute ERRHANDLE. This procedure does not respect synonyms.

SEE ALSO

FILE - ERRHANDLE - ERRSTATUS - SETSYN

EXAMPLE

```
/DECLARE/ FILE F;
          REF FILE RF;
          INTEGER I;

/EXEC/ BEGIN
        FILSETERR(F,1);   & end of file recovery for the file F
        RF := F;
        I := 0;
        FILSETERR(RF,I);   & no recovery for the file F
       END;
```

GET

NAME

GET , GETLN - Input handling functions.

SYNTAX

GETLN(*[file,] type[, integer]*)
GET(*[file,] type[, integer]*)

DESCRIPTION

These functions return a value which has the same type as the type given by the second argument. The type may be INTEGER, REAL, BOOLEAN, STRING, CLASS, QUEUE, FLAG or a type created by the user.

The read of the value is made from the file given by the first argument.

A format may be used to define the data length to read. It is given by the third argument. It is the size of the data to read; blanks preceding and following the data are not considered.

file is an expression representing a FILE type entity.

integer is an expression representing a INTEGER type entity.

EVALUATION

During the execution.

NOTES

- If the first argument does not appear, then the read is made from the implicit file (defined in the command /CONTROL/UNIT=GET(*file*)).
- The character "&" is considered as a comment limiter, excepted in the cases mentioned in WARNINGS section.
- The difference between GET and GETLN is that the current record is left after the read for the GETLN function.

WARNINGS

- For a string read : if the data length in known, then the format may be used and then the user can enter his string without the quotation marks ("). In this case the character "&" is not considered as a comment limiter.
- If the value to read has not the specified type, a new read can be performed depending on the value of ERRRRETRY file attribut, and the number of tried already performed.

SEE ALSO

FILE

EXAMPLE

```
/DECLARE/ INTEGER I;
          REF FILE RF;
          CLASS CL;
          FLAG FL;
          QUEUE Q;
          OBJECT OB;
              INTEGER IOB;
          END;
          OB OBJ;

/CONTROL/ UNIT=GET(FSYSINPU);

/EXEC/ BEGIN
        RF := FSYSINPU;
        PRINT("Execution of the program produces :");
        PRINT(" ");
        PRINT("R1 : ",GETLN(INTEGER));
        PRINT("R2 : ",GETLN(RF,REAL));
        PRINT("R3 : ",GET(RF,BOOLEAN,5));
        I := 3;
        PRINT("R4 : ",GETLN(RF,STRING,I+2));
        PRINT("R5 : ",GETLN(STRING));
        PRINT("R6 : ",GETLN(CLASS));
        PRINT("R7 : ",GETLN(RF,QUEUE));
        PRINT("R6 : ",GET(FLAG));
        PRINT("R6 : ",GETLN(RF,OB));
       END;
12
12.1
FALSE TRUE
"TRUE"
CL
Q
FL   OBJ
```

Execution of the program produces :

```
R1 :        12
R2 :    12.10
R3 : FALSE
R4 :  TRUE
R5 : TRUE
```

```
R6  : CL
R7  : Q
R6  : FL
R6  : OBJ
```

NAME

ISAUDTED - Tests for the existence of an auditor-audited link between two files

SYNTAX

ISAUDTED(*file1,file2*)

DESCRIPTION

This function returns a value TRUE if there is an auditor-audited or audited-auditor link between files *file1* and *file2*.

The two arguments are expressions representing FILE type entities.

EVALUATION

During the execution.

SEE ALSO

AUDIT - UNAUDIT - NBAUDTED - NBAUDTOR - FAUDITED - FAUDITOR

EXAMPLE

```
/DECLARE/FILE f1, f2, f3;

/EXEC/BEGIN
        FILASSIGN (f1,"f1.lis");
        FILASSIGN (f2,"f2.lis");
        OPEN (f1,2);
        AUDIT (f1,f2);
        IF ISAUDTED(f1,f2) THEN & returns TRUE
        ...
        IF ISAUDTED(f2,f1) THEN & returns TRUE
        ...
        IF ISAUDTED(f3,f1) THEN & returns FALSE
      END;
```

NBAUDITED

NAME

NBAUDTED - Access to the number of audited files linked to a given auditor file (NumBer AUDITED).

SYNTAX

NBAUDTED(*file*)

DESCRIPTION

This function returns the number of audited files associated with the auditor file given in first argument. This corresponds to the number of files from which the file is raised.

file is an expression representing a FILE type entities.

EVALUATION

During the execution.

SEE ALSO

AUDIT - UNAUDIT - ISAUDTED - NBAUDTOR - FAUDITED - FAUDITOR

EXAMPLE

```
/DECLARE/FILE f1, f2, f3;

/EXEC/BEGIN
        FILASSIGN (f1,"f1.lis");
        FILASSIGN (f2,"f2.lis");
        OPEN (f1,2);
        AUDIT (f1,f2);
        FOR i := 1 STEP 1 UNTIL NBAUDTED(f2) DO
                                  & the function returns 1 here
           BEGIN
             ...
             i : = NBAUDTED (f1); & function returns 0
             ...
           END;
      END;
```

NAME

NBAUDTOR - Access to the number of audited files linked to a given auditor file. (NumBer AUDITOR).

SYNTAX

NBAUDTOR(*file*)

DESCRIPTION

This function returns the number of audited files associated with the auditor file given in the argument. This corresponds to the number of files from which the file is raised.

file is an expression representing a FILE type entities.

EVALUATION

During the execution.

SEE ALSO

AUDIT - UNAUDIT - ISAUDTED - NBAUDTED - FAUDITED - FAUDITOR

EXAMPLE

```
/DECLARE/FILE f1, f2, f3;
         INTEGER i;

/EXEC/BEGIN
        FILASSIGN (f1,"f1.lis");
        FILASSIGN (f2,"f2.lis");
        OPEN (f1,2);
        AUDIT (f1,f2);
        FOR i := 1 STEP 1 UNTIL NBAUDTOR (f1) DO
                                & the function returns here : 1
            BEGIN
              ...
             i : = NBAUDTOR (f2); & function returns 0
             ...
            END;
      END;
```

OPEN

NAME

OPEN - File opening.

SYNTAX

OPEN(*file[, integer]*)

DESCRIPTION

This procedure opens the file given by the first argument. The opening mode is defined by the second argument. The possible values are:

= 1: opening in read

= 2: opening in write

= 3: opening in read/write

file is an expression representing a FILE type entities.

integer is an expression representing a INTEGER type entity.

EVALUATION

During the execution.

NOTE

- If the second argument is omitted, then the default value is 3.
- This procedure modifies the file attribute OPENMODE.
- This procedure respects synonyms.

WARNING

- A file opened in read mode corresponds to an existing physical file.
- A file opened in write mode does not correspond to any existing physical file.

SEE ALSO

CLOSE - FILASSIGN - FILE - OPENMODE

EXAMPLE

```
/DECLARE/ FILE F,G;
          REF FILE RG;
          INTEGER I;

/EXEC/ BEGIN
        OPEN(F,1);   & opening in read for the file F
        RG := G;
        I := 3;
        OPEN(RG,I);   & opening in read/write for the file F (equivalent
                      & to OPEN(RG)
       END;
```

RESTORE

NAME

RESTORE - Restoring processing.

SYNTAX

RESTORE (*[file,] "model_name".[,"*CONTINUE"] |"DEFERRED");

DESCRIPTION

This procedure restores the context of a QNAP2 execution saved by SAVE or SAVERUN procedure.

If the first argument does not appear, then the context restore is made from the implicit file (defined in the command:
/CONTROL/UNIT=LIBR(*file*). Otherwise, the save is made from the file given by the first argument.

The second argument defines the name of the model to be restored. By default the model contained in the file is restored. The first argument may be a file object or a reference to a file object. The second argument may be a constant, a variable, a function call or an expression.

The facilities for RESTORE include the possibility to define new statements to be executed before restarting the simulation (or the algorithmic block) only if a SAVERUN procedure has been used.

- If the key word CONTINUE is used, the simulation or the algorithmic sequence continues. This is the default behavior.
- If the key word DEFERRED appears, then the simulation or the algorithmic sequence will continue after the treatment of statements which must be defined in a new /REBOOT/ command to be placed just after the RESTORE statement. The statements used in the /REBOOT/ command can be any statement of the language, any procedure manipulating the customers and any procedure among the new ones introduced to permit the handling of several data defined in /CONTROL/ blocks.

file is an expression representing a FILE type entity.

string is an expression representing a STRING type entity.

EVALUATION

During the execution.

NOTES

- The default option is CONTINUE.
- The implicit file may be assigned at the beginning of a QNAP2 execution.
- The context may be automatically restored when an error occurs, in this case *"model_name"* is equal to QNAPSERR.

WARNING

- This procedure must be used in an EXEC command.
- If a file is specified, it must be assigned before (FILASSIGN procedure).
- If a file is specified, then it must be closed. The procedure handles the opening and the closing in this case.
- A RESTORE procedure destroys the previous context.

SEE ALSO

SAVE - SAVERUN - FILASSIGN - FILE

EXAMPLE

```
/DECLARE/ FILE F;
          STRING S;
          REF FILE RF;

/EXEC/ BEGIN
        RF := F;
        RESTORE("model_name");  & model restored form the implicit file
                                & FSYSLIB
        S := "model_name");
        RESTORE(F,S);           & model restored from the file F
        RESTORE;                & model restored from FSYSLIB without
                                & test on the model name
       END;
```

SAVERUN

NAME

SAVERUN , SAVE - Library saving.

SYNTAX

SAVE ([*file,*],"*model_name*")
SAVERUN ([*file,*],"*model_name*")

DESCRIPTION

This procedure saves the context in a file.
The procedure SAVE can be used only in an /EXEC/ sequence.
The procedure SAVERUN permits to execute statements placed after the SAVERUN operation either in a /EXEC/ block or in a service of a station (immediately or after a RESTORE).

If the first argument does not appear, then the context save is made on the implicit file (defined in the command /CONTROL/UNIT=LIBR(*file*)). Otherwise, the save is made to the file given by the first argument.

The second argument defines the name of the model to be saved. The first argument may be a file object or a reference to a file object. The second argument may be a constant, a variable, a function call or an expression.

file is an expression representing a FILE type entity.

string is an expression representing a STRING type entity.

EVALUATION

During the execution.

NOTE

The implicit file may be assigned at the beginning of a QNAP2 execution.

WARNINGS

- If a file is specified, it must be assigned befor (FILASSIGN procedure).
- If a file is specified, then it must be closed. The procedure handles the opening and the closing in this case.
- A RESTORE procedure destroys the previous context.

EXAMPLE

```
/DECLARE/ FILE F;
          STRING S;
          REF FILE RF;

/EXEC/ BEGIN
        RF := F;
        SAVE("model_name");  & model saved on the implicit file FSYSLIB
        S := "model_name");
        SAVE(F,S);           & model saved on the file F
        ...                  *
       END;

/EXEC/RESTORE ("model_name")
```

The operations labeled (*) are not processed after RESTORE.

This second example shows how a SAVERUN operation can be executed during a simulation.

```
/DECLARE/QUEUE Q; REAL DATE; FILE F;

/STATION/NAME=Q;
         SERVICE=BEGIN
                 .
                 .
                 IF ABS (TIME-DATE)>0.01 THEN
                    SAVERUN ("SAVE_SIMUL");
                 .
                 .
                 END;

/EXEC/BEGIN
       FILASSIGN (F, "SAVE.LIB");
       SIMUL;
       SAVERUN (F, "END_SIMUL");
       PRINT ("SAVE/RESTORE OPERATION DONE");
      END;
```

SETBUF

NAME

SETBUF - Buffer length definition.

SYNTAX

SETBUF(*file, integer*)

DESCRIPTION

This procedure defines the buffer length for the file given by the first argument. This number is defined by the second argument. The first argument may be a file object or a reference to a file object. The second argument may be a constant, a variable, a function call or an expression.

file is an expression representing a FILE type entities.

integer is an expression representing a INTEGER type entity.

EVALUATION

During the execution.

NOTES

- This procedure has to be called on a closed file.
- Its modifies the file RECLENGTH attribute.
- When a file is created, the length is initialized to 120.

WARNING

- This procedure does not respect synonyms.
- The buffer length may vary with the machines.

SEE ALSO

FILE - RECLENGTH - SETSYN

EXAMPLE

```
/DECLARE/ FILE F;
          REF FILE RF;
          INTEGER I;

/EXEC/ BEGIN
        SETBUF(F,80);   & 80 characters in a buffer for the file F
        RF := F;
        I := 120;
        SETBUF(RF,I);   & 120 characters for the file F (default value)
       END;
```

SETRETRY

NAME

SETRETRY - Definition of the number of trials.

SYNTAX

SETRETRY(*file, integer*)

DESCRIPTION

This procedure defines the number of read trials for the file given by the first argument. The number of trials is defined by the second argument.

file is an expression representing a FILE type entities.

integer is an expression representing a INTEGER type entity.

EVALUATION

During the execution.

NOTE

When a file is created, the number of trials is initialized to 5. Its modifies the file attribute ERRRETRY.

WARNING

This procedure does not respect synonimys.

SEE ALSO

FILE - ERRRETRY - SETSYN

EXAMPLE

```
/DECLARE/ FILE F;
          REF FILE RF;
          INTEGER I;

/EXEC/ BEGIN
        SETRETRY(F,1);   & only one trial the file F
        RF := F;
        I := 5;
        SETRETRY(RF,I);  & 5 trials allowed for the file F (default value)
       END;
```

NAME

SETSYN - defining synonyms.

SYNTAX

SETSYN(*file1, file2*)

DESCRIPTION

This procedure causes the file given by the first argument to be synonymous to the file defined by the second argument; that is to say, that all the procedures working for the first file will also work for the second file.

file1, file2 are expressions representing FILE type entities.

EVALUATION

During the execution.

WARNINGS

- The first file cannot be opened before the synonym is defined.
- All the procedures respect the synonym definition except FILASSIGN, FILSETERR, SETRETRY and SETSYN which work for the file given as argument.
- To find the characteristics of the first file, one must access the file attributes of this file except BUFPOSPT, FILPOS and RECLENGTH. For these attributes, the second file has to be used.

SEE ALSO

FILE - FILASSIGN

EXAMPLE

```
/DECLARE/ FILE F,G;
          REF FILE RF,RG;

/EXEC/ BEGIN
        SETSYN(F,G);  & all the orders on G are made in fact on F
        RF := F;
        RG := G;
        SETSYN(RF,RG);
       END;
```

UNAUDIT

NAME

UNAUDIT - End of audition of a file on one or several other files.

SYNTAX

UNAUDIT(*file1,file2,...fileN*)

DESCRIPTION

End of an audition operation, as a consequence, all input/output orders for the file presented as first argument (audited) will no longer be duplicated to the other files given in the argument (auditors).

file1, ...fileN are expressions representing FILE type entities.

EVALUATION

During the execution.

WARNING

An auditor file can only be closed if all links to the audited files have been destroyed by using UNAUDIT (or by closure of the audited files).

SEE ALSO

AUDIT - ISAUDTED - NBAUDTED - NBAUDTOR - FAUDITED - FAUDITOR

EXAMPLE

```
/DECLARE/FILE f1,f2,f3;
         INTEGER i=2;

/EXEC/BEGIN
        FILASSIGN (f1,"f1.lis")
        FILASSIGN (f2,"f2.lis")
        FILASSIGN (f3,"f3.lis")
        OPEN (f1,2)
        OPEN (f3,2)
        AUDIT (f1,f2)
        AUDIT (f1,f3,f1",2*i) & all entries in f1 will
                              & also be made in f3 with heading
                              & recording : "f1 no recording"
        WRITELN(f3," beginning of f3")
        WRITELN(f1," I am a copy of f1")
        UNAUDIT (f1,f3); & entries in f1 will no longer be
                         & duplicated to f3
        CLOSE (f3)
      END;
```

NAME

PRINT , WRITE , WRITELN - Write operations.

SYNTAX

PRINT(*integer1—real1—...[: integer2[: integer3]],...*)

WRITE(*[file,] integer1—real1—...[: integer2[: integer3]],...*)

WRITELN(*[file,] integer1—real1—...[: integer2[: integer3]],...*)

DESCRIPTION

These procedures are used to write results in the file specified by argument.

The difference between WRITE and WRITELN is that the current record is printed left after the write only for the WRITELN procedure.

The write is made on the file given by the first argument.

The result(s) to be written may be a value of a predeclared type or of a type created by the user.

A format may be used to describe how to write the result. It is given by the third argument and the fourth argument. The third argument defines the size of the data to write. The fourth argument is taken in account for real values. It gives the number of digits after the decimal point.

file is an expression representing a FILE type entity.

real is an expression representing a REAL type entity.

integer1, integer2, integer3 are expression representing INTEGER type entities.

EVALUATION

During the execution.

NOTES

- If the first argument does not appear in a WRITE or WRITELN operation, then the write is made on the implicit file (defined in the command /CONTROL/UNIT=PRINT(*file*)), in this case PRINT and WRITELN are equivalent.
- PRINT without argument corresponds to a page jump.
- PRINT (" ") corresponds to a lign jump.
- The write of a customer reference consists in writting the number associated, as in the standard trace file.

WARNINGS

- When a write is impossible in the specified format, then a level 2 error is generated (i.e. FILSTERR).
- For procedures WRITE, WRITELN the first character, which is the control character for the operation system, is not generated automatically as for the procedure PRINT. The user must insert this character in front of the buffer.

EXAMPLE

```
/DECLARE/ INTEGER I,LENG,AFTDECPT;
          REAL R;
          BOOLEAN B;
          STRING(1) BL=" ";   & control character in front of the buffer
          STRING S;
          REF FILE RF;
          CLASS CL;REF CLASS RCL=CL;
          FLAG FL;REF FLAG RFL=FL;
          QUEUE Q;REF QUEUE RQ=Q;
          OBJECT OB;
              INTEGER IOB;
          END;
          OB OBJ;REF OB ROBJ=OBJ;

/EXEC/ BEGIN
        RF := FSYSPRINT;
        PRINT("Execution of the program produces :");
        PRINT(" ");
        WRITELN(BL,"R1 : ",3);
        WRITELN(BL,"R2 : ",3.0);
        B := FALSE;
        WRITELN(BL,"R3 : ",B);
        S := "OK";
        WRITELN(RF,BL,"R4 : ",S);
        I := 123;
        WRITELN(BL,"R5 : ",I:4);
        R := 123.1;
        LENG := 6;
        AFTDECPT := 2;
        WRITELN(BL,"R6 : ",R:LENG:AFTDECPT);  & equivalent to the format
        WRITELN(BL,"R7 : ",RQ);               & F6.2 in FORTRAN
        WRITELN(BL,"R8 : ",RCL);
        WRITELN(BL,"R9 : ",RFL);
        WRITELN(BL,"R10: ",ROBJ);
       END;
```

Execution of the program produces :

```
R1 : 3
R2 : 3.000
R3 : FALSE
R4 : OK
R5 :  123
R6 : 123.10
R7 : Q
R8 : CL
R9 : FL
R10: OBJ
```

7.3.6 Object Management

CARD	Size of a list or an array.
DELETED	Object deletion test.
DISPOSE	Object deletion.
INCLUDIN	Access to the containing object.
NEW	Dynamic creation of a type instance.
REFSON	Access to a son of a customer.
TYPNAME	Access to the type of an object.

NAME

CARD - Size of a list or an array.

SYNTAX

CARD (*expression*)

DESCRIPTION

This function returns an integer representing the size of the referenced array or list.

expression is an array or a list.

EVALUATION

During the execution.

NOTE

SEE ALSO

EXAMPLE

```
/DECLARE/ INTEGER N(10);

/EXEC/    BEGIN
          PRINT(CARD(N));

          & Result is 10

  PRINT(CARD(1 STEP 1 UNTIL 6));

          & Result is 6

          PRINT(CARD(ALL QUEUE WITH NB > 5));

          & Prints the number of queues with more than 5 customers

          END;
```

DELETED

NAME

DELETED - Object deletion test.

SYNTAX

DELETED (*ref_obj*)

DESCRIPTION

The function DELETED returns TRUE if the object is effectively destroyed. This function also applies to all user-defined types and subtypes.

This function provides an help to test the object deletion.

ref_obj is an expression representing a REF type entity.

EVALUATION

During the execution.

NOTE

DELETED(NIL) returns FALSE.

SEE ALSO

NEW - DISPOSE

EXAMPLE

```
/DECLARE/ OBJECT TOOL;
             INTEGER LIFE;
             REAL REPAIR;
          END;

          REF FLAG RF;
          REF TOOL RT;
          REF CUSTOMER RC;
          QUEUE Q;

/STATION/NAME = Q;
         SERVICE=BEGIN
                   RT:=NEW(TOOL);
                   RF:=NEW(FLAG);
                   .
                   .
                   IF NOT (DELETED(RT))THEN
                     DISPOSE (RT);
                   IF NOT (DELETED (RF))THEN
                     DISPOSE (RF);
                   IF NOT (DELETED(RC))THEN
                     TRANSIT(RC, OUT);
                   .
                   .
                 END;
```

DISPOSE

NAME

DISPOSE - Object deletion.

SYNTAX

DISPOSE (*ref_object*)

DESCRIPTION

DISPOSE permits the deletion of objects created dynamically with the function NEW.

The deletion is consided by QNAP2 as a recuperation of memory space.

ref_obj is an expression representing a REF type entity.

EVALUATION

During the execution.

WARNINGS

- An object created in a /DECLARE/ command cannot be deleted.
- For scalar type objects cannot be deleted: INTEGER, REAL, BOOLEAN or STRING.
- For the types QUEUE, CLASS and FILE, an instance of the type cannot be deleted. The rule is the same for the subtypes of QUEUE, CLASS and FILE and for the types having an attribute of the type QUEUE, CLASS or FILE.
- An object of the type FLAG can be deleted only if no customer is waiting on the flag.
- Customers are deleted by transiting OUT.

SEE ALSO

NEW - DELETED

EXAMPLE

```
/DECLARE/ OBJECT gamme;
            INTEGER nbphas;
          END;

          REF gamme rg;

/EXEC/ BEGIN
         rg := NEW (gamme);
         ...
         DISPOSE (rg);
         ...
       END;
```

INCLUDIN

NAME

INCLUDIN - Access to the containing object.

SYNTAX

INCLUDIN (*ref_obj*)

DESCRIPTION

The function INCLUDIN returns a reference on the object which contains the specified object.

ref_obj is an expression representing a REF type entity.

EVALUATION

During the execution.

NOTE

If the object is not contained in another one, the function returns NIL.

SEE ALSO

OBJECT

EXAMPLE

```
/DECLARE/REF QUEUE RQ;
         OBJECT WORKTOP
            QUEUE Q1;
            .
            .
         END;

         OBJECT MACHINE;
         WORKTOP MACWORK;
         END;
         .
         .
         MACHINE MAC1;
         REF WORKTOP RW;
         REF ANY RANY;

/EXEC/BEGIN
       FOR RQ::ALL QUEUE DO
        BEGIN
          RANY:=INCLUDIN(RQ);
          PRINT(RANY); & the result is MACWORK
        END;
       FOR RW::ALL WORKTOP DO
        BEGIN
          RANY:=INCLUDIN(RW);
          PRINT(RANY); & the result is MAC1
        END;
       END;
```

NEW

NAME

NEW - Dynamic creation of a type instance.

SYNTAX

NEW (*type* [, *parameter-list*])

DESCRIPTION

This function returns a reference on the new created object.

The first argument represents the new object type.

The following arguments are the new object parameters, if some have been defined.

A CUSTOMER is created by the NEW(CUSTOMER) function. When a son customer is created, links between father and son are created.

A son customer has the same class and priority level sa its father.

EVALUATION

During the execution.

NOTES

- The FATHER attribute of a CUSTOMER object returns the father customer address.
- The SON attribute of a CUSTOMER object returns the son customer address.
- REFSON function accesses to every sons of a CUSTOMER.

WARNING

- A new customer can be created, in a service sequence, during a simulation resolution.
- A son customer is no original queue: (CQUEUE = NIL) so it must be transited to a station.
- If new attributes have been added to CUSTOMER type, the value of the son new attributs are not initialized when it is created.

SEE ALSO

:: - OBJECT - REF - SON - FATHER - REFSON - SONNB - DISPOSE - DELETED

EXAMPLE

```
/DECLARE/ QUEUE OBJECT machine (durusin, local);
            REAL durusin, local;
          END;
          CUSTOMER INTEGER val;
          REF CUSTOMER rc;
          QUEUE q;

/STATION/ NAME = *machine;
          SERVICE =
BEGIN
  ...
  rc := NEW (CUSTOMER);
  rc.val := val;     & affectation of the user attribute
  TRANSIT (rcq);
  ...
  TRANSIT (NEW (CUSTOMER), q);
  ...
END;

/DECLARE/ REF machine rm (2);

/EXEC/ BEGIN
         rm (1) := NEW (machine, 2.4, 6.0);
         rm (2) := NEW (machine, 5.8, 1.0);
       END;
```

REFSON

NAME

REFSON - Access to a son of a customer.

SYNTAX

REFSON(*customer, integer*)

DESCRIPTION

This function returns a reference to the customer which is a son of the customer defined by the first argument; the number of the son sought is given by the second argument.

The alive sons are numbered from one to the number of sons; the oldest son has the number one and the youngest has the highest number.

integer is an expression representing a INTEGER type entity.

EVALUATION

During the execution.

NOTES

- If the second argument is greater than the effective number of sons, then the value NIL is returned.
- The last created customer can be accessed by the SON attribute of the father, the associated number can be obtained by SONNB function.

WARNING

This function may be used only in simulation, in a service sequence or a TEST sequence.

EXAMPLE

```
/DECLARE/ QUEUE Q1,Q2;
          INTEGER I;
          REF CUSTOMER C;

/STATION/NAME=Q1;
         INIT=1;
         SERVICE=BEGIN
                  FOR I:=1 STEP 1 UNTIL 5 DO
                   BEGIN
                    C:=NEW(CUSTOMER);
                    PRINT("number : ",I,"  customer : ",C);
                    TRANSIT(C,Q2);
                   END;
                  PRINT("access to the sons");
                  FOR I:=1 STEP 1 UNTIL 5 DO
                   PRINT("number : ",I,"  refson : ",REFSON(CUSTOMER,I));
                  TRANSIT(Q2.FIRST,OUT);
                  PRINT("sons after exit");
                  FOR I:=1 STEP 1 UNTIL 4 DO
                   PRINT("number : ",I,"  refson : ",REFSON(CUSTOMER,I));
                  C:=CUSTOMER;
                  I:=6;
                  PRINT("number : ",I-1,"  refson : ",REFSON(C,I-1));
                 END;
         TRANSIT=OUT;

/CONTROL/TMAX=1;
         OPTION=NRESULT;

/EXEC/ BEGIN
        PRINT("Execution of the program produces :");
        SIMUL;
       END;
```

Execution of the program produces :

```
number :         1   customer :    999306
number :         2   customer :    999258
number :         3   customer :    999218
number :         4   customer :    999178
number :         5   customer :    999138
access to the sons
number :         1   refson :    999306
number :         2   refson :    999258
number :         3   refson :    999218
number :         4   refson :    999178
number :         5   refson :    999138
sons after exit
number :         1   refson :    999258
number :         2   refson :    999218
number :         3   refson :    999178
number :         4   refson :    999138
number :         5   refson : NIL
```

NAME

TYPNAME - Access to the type of an object.

SYNTAX

TYPNAME (*ref_object[,integer]*)

DESCRIPTION

The function TYPNAME returns a string which is the name of the type of the specified object.

Argument 1: object or reference on an object

Argument 2: overlapping level; the default value is 0

integer is an expression representing a INTEGER type entity.

EVALUATION

During the execution.

NOTES

- If the first argument is NIL, then TYPNAME returns ''NIL''.
- If the first argument is a deleted object, then TYPNAME returns ''DELETED''.
- If the second argument is higher than the real overlapping level, the TYPNAME returns ''ANY''.

SEE ALSO

OBJECT

EXAMPLE

```
/DECLARE/
OBJECT WORKTOP;
      QUEUE Q1;
END;
WORKTOP OBJECT MACHINE;
.
.
END;

REF WORKTOP RW;
MACHINE MAC1;

/EXEC/BEGIN
       FOR RW:=ALL WORKTOP DO
        BEGIN
          PRINT(TYPNAME(RW));   & the result is "WORKTOP"
          PRINT(TYPNAME(RW,1)); & the result is "MACHINE"
          PRINT(TYPNAME(RW,2)); & the result is "ANY"
        END;
       END;
```

7.4 Debugging Tool

BREAK	Break point definition.
CANCELBR	Cancellation of model break points.
GO	Model execution continuation while debugging.
HALT	Creation of the first break point of the debugger.
INSERT	Insertion of algorithmic code in the model while debugging.
REMOVE	Removal of inserted code.
SHBREAK	Display of line numbers containing break points.
SHINSERT	Display of line numbers where code was inserted.
STP	Step by step execution of the model.

BREAK

NAME

BREAK - Break point definition.

SYNTAX

BREAK (*integer1* [, *integer2*,...]) ;

DESCRIPTION

This procedure puts one or many break points at one or many statements of the model.

The concerned statement or statements are referenced by their line number (*integer1* for the first one, *integer2* for the second one, and so on). These line numbers can be found in the FSYSOUTP file.

A call to this procedure is generally interactively performed by the user from the keyboard.

The lines at *integer1, integer2,...* should have been compiled with the DEBUG option. Otherwise, an error message is issued.

integer1, integer2,... are expressions returning or referencing an INTEGER result.

EVALUATION

During execution.

NOTES

Two break points cannot be put at the same statement.

WARNING

- The model should be executed interactively.
- Do not forget the ";" character closing the command.

SEE ALSO

HALT - GO -SHBREAK - CANCELBR -STP - OPTION

EXAMPLE

```
15 /STATION/ NAME = B;
16           SERVICE = BEGIN
17                       EXP (2);
18                       I := I + 1;
19                     END ;

BREAK( 17 );  & Break point
HALT ON BREAK POINT BEFORE LINE : 17 & Break point definition feedback
```

NAME

CANCELBR - Cancellation of model break points.

SYNTAX

CANCELBR (*integer1* [, *integer2*,...]) ;

DESCRIPTION

This procedure cancels the break points put by a BREAK command in a model.

integer1, integer2,... refer to the statements line numbers, as specified to the BREAK command, for which cancellation is requested. These line numbers can be found in the FSYSOUTP file.

integer1, integer2,... are expressions returning or referencing an INTEGER result.

EVALUATION

During execution.

NOTES

When no break point was previously put at the requested statement, a message is issued.

WARNING

- The model should be executed interactively.
- Do not forget the ";" character closing the command.
- Break point put using the HALT command cannot be cancelled.

SEE ALSO

BREAK - SHBREAK

EXAMPLE

```
SHBREAK; HALT ON BREAK POINT BEFORE LINE : 17  (Break point definition feedback)
```

This break point is cancelled by the following command: `CANCELBR (17) ;`

NAME

GO - Model execution continuation while debugging.

SYNTAX

```
GO ;
```

DESCRIPTION

This procedure resumes the model execution until next break point when debugging was activated.

A call to this procedure is generally interactively performed by the user from the keyboard.

EVALUATION

During execution.

WARNING

- The model should be executed interactively.
- Do not forget the ";" character closing the command.

SEE ALSO

HALT - BREAK - STP - OPTION

EXAMPLE

```
HALT ON BREAK POINT BEFORE LINE : 33
```

GO ; Resumes execution.

NAME

HALT - Creation of the first break point of the debugger.

SYNTAX

```
HALT ;
```

DESCRIPTION

This procedure should be used within an algorithmic block of the model.

A break point, which the user is not allowed to cancel, is generated at the current line each time the code is executed.

EVALUATION

During execution.

NOTES

This procedure is the only way to generate a first break point and enter debugging mode.

Only one call to the HALT procedure is allowed inside a model.

WARNING

- The model should be executed interactively.
- The debugging tool must not be used in MACRO.

SEE ALSO

GO - BREAK - STP -OPTION

EXAMPLE

```
/EXEC/ BEGIN
         HALT;
         ...
         SIMUL;
          ...
       END ;
```

INSERT

NAME

INSERT - Insertion of algorithmic code in the model while debugging.

SYNTAX

INSERT ([*file*,] [, *integer*]) ;

DESCRIPTION

This procedure allows the user to insert temporary algorithmic code in a model while debugging it.

The inserted code is executed before the statements of the referenced line.

file is the file from which code should be read ; if no file is given, the code is read from the keyboard.

integer is the line number where to insert the code. It can be found in the FSYSOUTP file. Code is inserted right before the specified line or, if no line number is given, right before the current line.

The line at *integer* should have been compiled with the DEBUG option. Otherwise, an error message is issued.

If the line at *integer* does not contain any executable code (comment line, or BEGIN,...), the code will be inserted right before the next line containing executable code.

When read from keyboard, insertion ends at the next continuation statement: STP or GO.

EVALUATION

During execution.

NOTES

Insertion of code on a line where a break point was put is not allowed. Also, insertion of code twice on the same line is not allowed.

WARNING

- The model should be executed interactively.
- Do not forget the ";" character closing the command.

SEE ALSO

OPTION - REMOVE - SHINSERT

EXAMPLE

```
15 /STATION/ NAME = B;
16           SERVICE = BEGIN
17                       EXP (2);
18                       I := I + 1;
19                     END ;

BREAK( 17 );  & Break point
HALT ON BREAK POINT BEFORE LINE : 17 & Break point definition feedback

INSERT (18);
BEGIN
  PRINT (I);
END ;
GO;           & Insertion ends here
```

REMOVE

NAME

REMOVE - Removal of inserted code.

SYNTAX

REMOVE (*integer1* [, *integer2*,...]) ;

DESCRIPTION

This procedure allows to remove previously inserted, using INSERT, algorithmic blocks.

integer1, integer2, ... refer to the line numbers where inserted blocks should be removed. This line numbers can be found in the FSYSOUTP file.

integer1, integer2,... are expressions returning or referencing an INTEGER result.

EVALUATION

During execution.

NOTES

When no code was inserted at the line numbers specified, a message is issued.

WARNING

- The model should be executed interactively.
- Do not forget the ";" character closing the command.

SEE ALSO

INSERT - SHINSERT - OPTION

EXAMPLE

```
SHINSERT ;
INSERT ON LINE : 11  Display of line numbers where code was inserted
REMOVE (11) ; Removes the code inserted at line 11
```

NAME

SHBREAK - Display of line numbers containing break points.

SYNTAX

```
SHBREAK ;
```

DESCRIPTION

Line numbers printed are those of the FSYSOUTP file.

This procedure prints out the list of break points actually put on the model, represented by their line number.

EVALUATION

During execution.

NOTES

The only break points listed are those put using the BREAK procedure, those specified by the HALT procedure are ignored.

WARNING

- The model should be executed interactively.
- Do not forget the ";" character closing the command.

SEE ALSO

HALT - GO - BREAK - CANCELBR - OPTION

EXAMPLE

```
SHBREAK ;

EXISTING BREAK POINT BEFORE LINE : 9
```

SHINSERT

NAME

SHINSERT - Display of line numbers where code was inserted.

SYNTAX

```
SHINSERT ;
```

DESCRIPTION

Line numbers printed are those of the FSYSOUTP file.

This procedure prints out a list of line numbers where code was inserted using the INSERT procedure.

EVALUATION

During execution.

WARNING

- The model should be executed interactively.
- Do not forget the ";" character closing the command.

SEE ALSO

INSERT - REMOVE - OPTION

EXAMPLE

```
SHINSERT ;

INSERT ON LINE : 11
```

NAME

STP - Step by step execution of the model.

SYNTAX

```
STP ;
```

DESCRIPTION

This procedure generates a break point (though not listed by SHBREAK) right before the next statement compiled with the DEBUG option.

The line number of new break point is issued.

EVALUATION

During execution.

NOTES

In case of a transit, STP steps to the statements executed by the next customer whose service was compiled with the DEBUG option.

WARNING

- The model should be executed interactively.
- Do not forget the ";" character closing the command.

SEE ALSO

HALT - GO - OPTION

EXAMPLE

```
15 /STATION/ NAME = B;
16           SERVICE = BEGIN
17                       EXP (2);
18                       I := I + 1;
19                       J := 3;
20                     END ;

BREAK( 17 );  & Break point at line 17
HALT ON BREAK POINT BEFORE LINE : 17 & Break point definition feedback
STP;                                & Request for step
STEP BEFORE LINE :  18              & Step feedback
```

7.4.1 General Tools

ABORT	Stops immediately the QNAP2 execution.
CONVERT,CVNOERR	Scalar type to scalar type conversion.
GETCPUT	CPU time elapsed
GETDATE	Current date.
GETDTIME	Current time.
GETMEM	Memory occupation.
HOSTSYS:GETENV	Get value of environment variables.
HOSTSYS:GETERCOD	Get the last system error code.
HOSTSYS:SHELL	Issue a shell command.
SETTMAX	Dynamic specification of the maximum simulation run length.
STOP	Stops the resolution execution.
UTILITY	Communication between QNAP2 and FORTRAN.

NAME

ABORT - Stops immediately the QNAP2 execution.

SYNTAX

```
ABORT
```

DESCRIPTION

This procedure stops immediately QNAP2 software. An error message is edited.

EVALUATION

During the execution.

NOTES

These procedure doesn't need any argument.

SEE ALSO

STOP

EXAMPLE

```
/DECLARE/ BOOLEAN berror=FALSE;

/EXEC/ BEGIN
         SIMUL;
         ...
         IF (berror) THEN ABORT;
         ...
         SIMUL;
       END;

/END/
```

CONVERT,CVNOERR

NAME

CONVERT , CVNOERR - Scalar type to scalar type conversion.

SYNTAX

CONVERT(*integer* | *real* | *boolean* | *string, type[, integer[, integer]]*)

CONVERT(*object* | *string [, integer [, integer]]*)

CVNOERR(*integer* | *real* | *boolean* | *string, type[, integer[, integer]]*)

CVNOERR(*object* | *string [, integer [, integer]]*)

DESCRIPTION

These functions convert each type (integer, real, boolean and string) in another among the four types defined here. They give the same result but react differently when the change requested is impossible or invalid :

- CONVERT halts the execution
- CVNOERR continues the execution and stores the value TRUE in the new predeclared boolean variable ERRCONV which contains the value FALSE when the conversion is correct.

Each argument except the second may be a constant, a variable, a function call or an expression. The second argument may be INTEGER, REAL, BOOLEAN or STRING. The result of the functions has the same type as the type defined by the second argument. The third argument and the fourth argument are optional and define the format for the conversion.

Conversion of a type to the same type

This operation is allowed and the first argument is copied as is. If arguments appear defining the format, they are ignored.

Conversion integer → real and real → integer

These conversions operate like the functions REALINT and INTREAL or the implicit conversion caused by ":=". The format is also ignored in this case.

Conversion integer → boolean and real → boolean

The result is TRUE if the value is not equal to zero; otherwise it is FALSE. The format is ignored.

Conversion boolean → integer and boolean → real

The result is zero if the value is FALSE, otherwise the result is 1 for an integer and 1.0 for a real number. The format is ignored.

Conversion string → integer

The result is the integer corresponding to the value stored in the string. The string may contain the characters "0", "1", "2", "3", "4", "5", "6", "7", "8", "9", "+" and "-". If another character

appears in the string, then the conversion is invalid. The format is ignored.

Conversion string → real
The result is the real number corresponding to the value stored in the string. The string may contain all the characters defined previously plus "E" and ".". Another character appearing in the string invalidates the conversion. The format is also ignored.

Conversion string → boolean
The result is the boolean value corresponding to the value stored in the string. The strings permitted for the conversion are "TRUE", "FALSE", "YES", "NO", "T", "F", "Y" and "N". In the other cases the conversion is invalid. The format is ignored.

Conversion integer → string
The result is a string which contains the value stored in the integer. The third argument may be used to define the format; it has to be greater than the number of decimal digits with the eventual sign, otherwise the conversion is invalid. The result is positioned on the right side in the string. The fourth argument is ignored if present.

Conversion real → string The result is a string which contains the value stored in the real number. The third argument and the fourth argument may be used to define the format. They represent the number of positions in which to store the value and the number of positions after the "." (similar to the format F in FORTRAN). If the fourth argument does not appear, then the value is printed with the format G as defined in FORTRAN. The result is positioned on the right side in the string. The third argument has to be greater than zero, otherwise the conversion is invalid.

Conversion object → string The result is a string which contains the QNAP2 internal name of the object. The third argument is the length of the resulting string. If the length is too short, a convert error occurs; if the third argument is omitted, the result is a string with the exact length. The optional fourth argument does not have any effect.

EVALUATION

During the execution.

WARNING

If an error occurs, a message is edited and the `QNAP2` session is stopped.
The conversion *object → string* does not exist in the `QNAP2` versions older than the 9.1 version. The greatest care must be taken in using the conversion *object → string*. It is advised to avoid using it in a comparison test with a user defined string, it is preferable to compare it with a string resulting from another similar `CONVERT` operation because the `QNAP2` internal nomenclature may change, particularly the arrays' one.

EXAMPLE

```
/DECLARE/ INTEGER I;
          REAL R;
          BOOLEAN B;
          STRING S;
          OBJECT FOO;
            END;
          FOO BAR;
          REF FOO GEE;
          QUEUE Q;
          REF QUEUE RQ;

/STATION/ NAME = Q;
INIT = 1;
SERVICE = BEGIN
          CST(1);
          S := CONVERT( CUSTOMER,STRING,8);  & result : 1
          END;
TRANSIT = OUT;

/EXEC/ BEGIN
        PRINT("Execution of the program produces :");
        PRINT(" ");
        PRINT("R1 : ",CONVERT(12,INTEGER));
        PRINT("R2 : ",CONVERT(12.0,REAL));
        PRINT("R3 : ",CONVERT(FALSE,BOOLEAN));
        PRINT("R4 : ",CONVERT("OK",STRING));
        I := CONVERT(12.5,INTEGER);
        PRINT("R5 : ",I);
        R := CONVERT(I,REAL);
        PRINT("R6 : ",R);
        B := CONVERT(0,BOOLEAN);
        PRINT("R7 : ",B);
        PRINT("R8 : ",CONVERT(1.5,BOOLEAN));
        PRINT("R9 : ",CONVERT(FALSE,INTEGER));
        PRINT("R10: ",CONVERT(TRUE,REAL));
        PRINT("R11: ",CONVERT("12",INTEGER));
        R := CVNOERR("12.A5",REAL);
        PRINT("R12: ",ERRCONV);
        R := CVNOERR("12.5",REAL);
        PRINT("R13: ",ERRCONV,R);
        PRINT("R14: ",CONVERT("F",BOOLEAN));
        PRINT("R15: ",CONVERT(FALSE,STRING,2));
```

```
 PRINT("R16: ",CONVERT(TRUE,STRING));
 PRINT("R17: ",CONVERT(12,STRING,3));
 PRINT("R18: ",CONVERT(12,STRING));
 PRINT("R19: ",CONVERT(12.5,STRING,5,1));
 PRINT("R20: ",CONVERT(12.5,STRING,5));
 PRINT("R21: ",CONVERT(12.5,STRING));

 PRINT("R22: ",CONVERT(BAR,STRING,8));
 PRINT("R23: ",CONVERT(GEE,STRING,8));
 GEE := BAR;
 PRINT("R24: ",CONVERT(BAR,STRING,8));
 RQ := Q;
 PRINT("R25: ",CONVERT(RQ,STRING,8));
END;
```

Execution of the program produces :

```
R1 :        12
R2 :    12.00
R3 : FALSE
R4 : OK
R5 :        12
R6 :    12.00
R7 : FALSE
R8 : TRUE
R9 :         0
R10:    1.000
R11:        12
R12: TRUE
R13: FALSE   12.50
R14: FALSE
R15:  F
R16: TRUE
R17:  12
R18: 12
R19:  12.5
R20: 12.500
R21: 12.50
R22: BAR
R23: NIL
R24: BAR
R25: Q
```

GETCPUT

NAME

GETCPUT - CPU time elapsed

SYNTAX

```
GETCPUT
```

DESCRIPTION

The function GETCPUT returns the elapsed time since the QNAP2 launch moment.

EVALUATION

During the execution.

NOTE

This function requires no argument.

WARNINGS

- The value returned corresponds to the effective CPU time consumed and not to the elapsed time.
- This function is not available on some sites depending on hardware and operating system considerations. It is mainly provided on UNIX computers or VMS ones. When not implemented, any call to this function returns a null value.

EXAMPLE

```
/EXEC/ PRINT("Cpu Time :",GETCPUT);
```

NAME

GETDATE - Current date.

SYNTAX

GETDATE

DESCRIPTION

The function GETDATE returns the current date on the computer.

The returned information is a string which has the form SS-MMM-AA where SS represents the day in 2 digits, MMM represents the first 3 characters of the month, and AA represents the last 2 digits of the year.

EVALUATION

During the execution.

NOTE

This function requires no argument.

SEE ALSO

GETDTIME

EXAMPLE

```
/EXEC/ PRINT("Current Date :",GETDATE);
```

GETDTIME

NAME

GETDTIME - Current time.

SYNTAX

GETDTIME

DESCRIPTION

The function GETDTIME returns the current time on the computer.

The returned information is a string which has the form HH-MM-SS where HH represents the hour ($0 \rightarrow 23$), MM represents the minutes, and SS represents the seconds

EVALUATION

During the execution.

NOTE

This function requires no argument.

WARNINGS

This function is different of TIME which represents the hour in the simulated system units.

SEE ALSO

GETDATE

EXAMPLE

```
/EXEC/ PRINT("Current Time :",GETDTIME);
```

NAME

GETMEM - Memory occupation.

SYNTAX

GETMEM

DESCRIPTION

The function GETMEM returns the occupation percentage of the internal memory of QNAP2. The returned information is a real value in the interval [0, 100].

EVALUATION

During the execution.

NOTE

This function requires no argument.

EXAMPLE

```
/EXEC/ PRINT("MEMORY.USED:",GETMEM);
```

HOSTSYS:GETENV

NAME

HOSTSYS:GETENV - Get value of environment variables.

SYNTAX

string1:= HOSTSYS:GETENV(*string2*)

DESCRIPTION

string1 and *string2* are STRING objects. The argument of HOSTSYS:GETENV is a string corresponding to the name of the environment variable. HOSTSYS:GETENV returns the value of this environment variable.

EVALUATION

During the execution.

NOTES

It is possible to get the error code generated by HOSTSYS:GETENV by using the HOSTSYS:GETERCOD function :

0 : no error.

1 : truncature (the environment variable value is longer than the QNAP2 string's maximum length.

2 : the environment variable does not exist.

3 : the argument is a null string.

SEE ALSO

HOSTSYS:GETERCOD

EXAMPLE

```
/EXEC/
BEGIN
  string := HOSTSYS:GETENV("PATH");
  PRINT ("Path : ", string);
END;
& Result :  /usr/local/bin:/usr/bin:/bin
```

NAME

HOSTSYS:GETERCOD - Get the last system error code.

SYNTAX

integer:= HOSTSYS:GETERCOD

DESCRIPTION

integer is an INTEGER object. HOSTSYS:GETERCOD returns the error code corresponding to the last QNAP2 system call (e.g. HOSTSYS:*key-word*).

EVALUATION

During the execution.

SEE ALSO

HOSTSYS:GETENV - HOSTSYS:SHELL

EXAMPLE

```
/EXEC/
BEGIN
  string1 := HOSTSYS:GETENV("PATH");
  PRINT (HOSTSYS:GETERCOD);            & Result :0
  string1 := HOSTSYS:GETENV("UNKNOWN");
  PRINT (HOSTSYS:GETERCOD);            & Result : 2
END;
```

HOSTSYS:SHELL

NAME

HOSTSYS:SHELL - Issue a shell command.

SYNTAX

integer:= HOSTSYS:SHELL(*string*)

DESCRIPTION

integer is an INTEGER object.

string is a STRING object.

The argument of HOSTSYS:SHELL is a string describing the command to be executed by the shell. QNAP2 waits until the command has completed.

HOSTSYS:SHELL returns the system exit status.

EVALUATION

During the execution.

NOTES

It is possible to get the error code generated by HOSTSYS:SHELL by using the HOSTSYS:GETERCOD function :

0 : no error.

1 : the argument is a null string.

SEE ALSO

HOSTSYS:GETERCOD

EXAMPLE

```
/EXEC/
BEGIN
  int := HOSTSYS:SHELL("ls *.qnp");
  PRINT ("Exit status: ", int);
END;
& Result: Exit status: 0
```

NAME

SETTMAX - Dynamic specification of the maximum simulation run length.

SYNTAX

SETTMAX (*real*) ;

DESCRIPTION

This procedure modifies dynamically the maximum simulation run length. During the simulation execution, the new simulation run length is set equal to *real*.

real is an expression representing a REAL type entity.

EVALUATION

During the execution.

NOTES

The static definition of the simulation run length is performed by the parameter TMAX of /CONTROL/ command.

WARNING

This procedure can be used in any algorithmic code sequence if a simulation resolution is performed.
The specified simulation run length has to be strictly greater than the current simulation time.

SEE ALSO

TMAX - REBOOT

EXAMPLE

```
/EXEC/ RESTORE ("model", "DEFERRED") ;
/REBOOT/ SETTMAX (TIME + 1000);
```

STOP

NAME

STOP - Stops the resolution execution.

SYNTAX

STOP

DESCRIPTION

This procedure stops QNAP2 execution after computing standards results.

EVALUATION

During the execution.

NOTES

This procedure does not need any argument.

SEE ALSO

ABORT

EXAMPLE

```
/DECLARE/ INTEGER nbmax;
          QUEUE A;

          PROCEDURE arret;
          BEGIN
            ...
            IF (A.NBOUT > nbmax) THEN STOP;
            ...
          END;

/END/
```

NAME

UTILITY - Communication between QNAP2 and FORTRAN.

SYNTAX

UTILITY(*integer, array1, [,array2]*)

DESCRIPTION

This procedure provides facilities for communication between QNAP2 and others external tools.

The first argument represents a code sent to the subroutine UTILIT which may take a different decision following the value.

The second argument is a single type array where are stored the data to transfer to FORTRAN or/and to retrieve from FORTRAN.

The third argument has the same semantic than the second one.

WARNINGS

- The arrays may have the type INTEGER, REAL, BOOLEAN or STRING.
- The content of the arrays may be modified in the user FORTRAN subroutine.

SEE ALSO

UTILIT

EXAMPLE

```
/DECLARE/ INTEGER itab (3) = (1, 8, 5);
          INTEGER otab (3);
          REAL rtab (1);

/EXEC/ BEGIN
         ...
         UTILITY (1, itab, otab);    & ordering itab values
                                     & ordered values are in otab
         UTILITY (2, itab, rtab);    & computes the mean of the values
                                     & the result is rtab
       END;
```

7.4.2 Resolution Procedures

CONVOL	Convolution algorithm.
DIFFU	Diffusion algorithm.
HEURSNC	Heuristic algorithm.
ITERATIV	Iterative algorithm.
MARKOV	Markovian solver.
MVA	Mean Value Analysis algorithm.
MVANCA	Mean Value Analysis and Normalized Convolution algorithm.
NETWORK	Selects a sub-model which defines the sub-network on.
PRIORPR	Algorithm for preemptive priority scheduling.
SIMUL	Discrete event simulation.
SOLVE	Analytical solvers.
SPLITMAT	SPLIT-MATCH approximation solver.

NAME

"CONVOL" - Convolution algorithm.

SYNTAX

SOLVE ("CONVOL");

DESCRIPTION

This method is the direct application of the product-form theorem due to Basket, Chandy, Muntz and Palacios.

CONVOL is a fast, exact but numerically unsafe solver.

The theorems extend to open, closed and mixed multiclass queueing networks previous results obtained by T.R. Jackson. Finite capacity queues and concurrency sets are taken into account by this solver.

Application stipulations summary :

station type	scheduling	service distribution	service per class	global serv. rate	serv. rate per class
SINGLE	FIFO	exponential	no	yes	no
	LIFO,PREEMPT	general	yes	yes	yes
	PS	general	yes	yes	yes
MULTIPLE	FIFO	exponential	no	yes	no
	EXCLUDE-FEFS	exponential	no	yes	no
	PS	general	yes	yes	yes
INFINITE		general	yes	yes	yes
SOURCE		exponential	no	no	no

EVALUATION

During the execution.

NOTES

This method may be used only for steady-state analysis.

If the network contains a finite capacity queue then the reject policy must be SKIP (transit to the next station).

If concurrency sets are used :

- The FEFS (First Eligible - First Served) option must be defined
- The concurrency sets must be defined using classes (probabilities are not available with CONVOL)
- Each class must be contained in a concurrency set.

SEE ALSO

SOLVE - "MVA" - "MVANCA" - "HEURSNC" - "PRIORPR" - "ITERATIV" - "DIFFU" - "SPLITMAT" - MARKOV - SIMUL

EXAMPLE

```
/DECLARE/ QUEUE q;
          CLASS c1, c2;

/STATION/ NAME = q;
          SCHED = LIFO, PREEMPT;
          SERVICE ( c1 ) = CST ( 10);
          SERVICE = EXP ( 5 );
          RATE ( c1 ) = 0.9;
          RATE ( c2 ) = 0.8;
 ...

/EXEC/ SOLVE ("CONVOL");
```

NAME

"DIFFU" - Diffusion algorithm.

SYNTAX

```
SOLVE ( "DIFFU" );
```

DESCRIPTION

This method implements a heuristic developed by E. Gelenbe and G. Pujolle in order to solve open queueing networks with stations having non-exponential service times and FIFO scheduling.

DIFFU is a fast, approximate and numerically safe solver.

Only open networks may be analysed.

Application stipulations summary :

station type	scheduling	service distribution	service per class	global serv. rate	serv. rate per class
SINGLE	FIFO	general	no	no	no
SOURCE		general	no	no	no

EVALUATION

During the execution.

WARNING

This method may be used only for steady-state analysis.

SEE ALSO

SOLVE - "CONVOL" - "MVA" - "MVANCA" - "HEURSNC" - "PRIORPR" - "ITERATIV" - "SPLITMAT" - MARKOV - SIMUL

EXAMPLE

```
/DECLARE/ QUEUE q1, q2;
          CLASS c1, c2;

/STATION/ NAME = q1;
          TYPE = SOURCE;
          SERVICE = EXP (20);
          TRANSIT = q2, c1, 1, q2, c2, 1;

/STATION/ NAME = q2;
          SERVICE ( c1 ) = CST ( 10);
          SERVICE = EXP ( 5 );
          TRANSIT = OUT;

/EXEC/ SOLVE ("DIFFU") ;
```

NAME

"HEURSNC" - Heuristic algorithm.

SYNTAX

SOLVE ("HEURSNC");

DESCRIPTION

This method implements a heuristic proposed by Schweitzer, Neuse and Chandy in order to solve networks with a large number of sub-chains.

HEURSNC is a fast, approximate and numerically unsafe solver.

Only closed networks may be analysed.

Application stipulations summary :

station type	scheduling	service distribution	service per class	global serv. rate	serv. rate per class
SINGLE	FIFO	exponential	no	yes	no
	LIFO,PREEMPT	general	yes	yes	no
	PS	general	yes	yes	no
MULTIPLE	FIFO	exponential	no	yes	no
	PS	general	yes	yes	no
INFINITE		general	yes	yes	yes

EVALUATION

During the execution.

WARNING

This method may be used only for steady-state analysis.

SEE ALSO

SOLVE - "CONVOL" - "MVA" - "MVANCA" - "PRIORPR" - "ITERATIV" - "DIFFU" - "SPLITMAT" - MARKOV - SIMUL

EXAMPLE

```
/DECLARE/ QUEUE q;
          CLASS c1, c2;

/STATION/ NAME = q;
          SCHED = LIFO, PREEMPT;
          SERVICE ( c1 ) = CST ( 10);
          SERVICE = EXP ( 5 );
          RATE = 1, 3, 5;

/EXEC/ SOLVE ("HEURSNC");
```

NAME

"ITERATIV" - Iterative algorithm.

SYNTAX

```
SOLVE ( "ITERATIV" );
```

DESCRIPTION

This method implements a heuristic developed by R. Marie in order to solve closed queueing networks including stations having a FIFO discipline and a non-exponentially distributed service time.

ITERATIV is a fast, approximate and numerically unsafe solver.

Only closed networks may be analysed.

Application stipulations summary :

station type	scheduling	service distribution	service per class	global serv. rate	serv. rate per class
SINGLE	FIFO	exponential	no	yes	no
	FIFO	general	no	no	no
	LIFO,PREEMPT	general	no	yes	no
	PS	general	no	yes	no
MULTIPLE	FIFO	exponential	no	yes	no
	PS	general	no	yes	no
INFINITE		generale	no	yes	no

EVALUATION

During the execution.

WARNING

This method may be used only for steady-state analysis.

SEE ALSO

SOLVE - "CONVOL" - "MVA" - "MVANCA" - "PRIORPR" - "HEURSNC" - "DIFFU" - "SPLITMAT" - MARKOV - SIMUL

EXAMPLE

```
/DECLARE/ QUEUE q;

/STATION/ NAME = q;
          SERVICE = CST ( 5 );

/EXEC/ SOLVE ("ITERATIV");
```

NAME

MARKOV - Markovian solver.

SYNTAX

MARKOV;

DESCRIPTION

This procedure calls the markovian solver.

This solver is available for any model that may be mapped onto a first order markovian process with a finite number of states.

Only closed queueing networks may be analysed.

The produced results are exact.

EVALUATION

During the execution.

WARNING

- This solver may be used only for steady-state analysis.
- The use of MARKOV is limited by the number of states generated during the analysis because an important number of states means a very important memory consummation which may cause an expensive computation cost.

SEE ALSO

SOLVE - SIMUL

EXAMPLE

```
/DECLARE/ QUEUE q1, q2, q3, sem;

/STATION/ NAME = q1;
          SERVICE = BEGIN
                      CST( 10 );
                      IF CUSTNB ( q2 ) > CUSTNB ( q3 ) THEN
                         TRANSIT ( q3 )
                      ELSE
                         TRANSIT( q2 );
                    END;

/EXEC/ MARKOV;
```

NAME

"MVA" - Mean Value Analysis algorithm.

SYNTAX

SOLVE ("MVA");

DESCRIPTION

This method implements the mean value analysis approach developed by M.Reiser.

MVA is a fast, exact and numerically safe solver.

Only closed networks may be analysed.

Application stipulations summary :

station type	scheduling	service distribution	service per class	global serv. rate	serv. rate per class
SINGLE	FIFO	exponential	no	no	no
	LIFO,PREEMPT	general	yes	no	no
	PS	general	yes	yes	no
INFINITE		general	yes	yes	no

EVALUATION

During the execution.

NOTES

This method avoids the computation of the normalization constants which may lead to numerical unstabilities.

WARNING

This method may be used only for steady-state analysis.

SEE ALSO

SOLVE - "MVA" - "CONVOL" - "HEURSNC" - "PRIORPR" - "ITERATIV" - "DIFFU" - "SPLITMAT" - MARKOV - SIMUL

EXAMPLE

```
/DECLARE/ QUEUE q;
          CLASS c1, c2;

/STATION/ NAME = q;
          SCHED = LIFO, PREEMPT;
          SERVICE ( c1 ) = CST ( 10);
          SERVICE = EXP ( 5 );
          RATE = 1, 3, 5;
          ...
/EXEC/ SOLVE ("MVA");
```

MVANCA

NAME

"MVANCA" - Mean Value Analysis and Normalized Convolution algorithm.

SYNTAX

SOLVE ("MVANCA");

DESCRIPTION

This method implements an hybride method based on the mean value analysis and the normalized convolution algorithms proposed by M.Reiser.

MVANCA is a fast, exact and numerically safe solver.

Only closed networks may be analysed.

Application stipulations summary :

station type	scheduling	service distribution	service per class	global serv. rate	serv. rate per class
SINGLE	FIFO	exponential	no	yes	no
	LIFO,PREEMPT	general	yes	yes	no
	PS	general	yes	yes	no
MULTIPLE	FIFO	exponential	no	yes	no
	PS	general	yes	yes	no
INFINITE		general	yes	yes	no

EVALUATION

During the execution.

NOTES

This method avoids the computation of the normalization constants which may lead to numerical unstabilities.

WARNING

This method may be used only for steady-state analysis.

SEE ALSO

SOLVE - "MVA" - "CONVOL" - "HEURSNC" - "PRIORPR" - "ITERATIV" - "DIFFU" - "SPLITMAT" - MARKOV - SIMUL

EXAMPLE

```
/DECLARE/ QUEUE q;
          CLASS c1, c2;

/STATION/ NAME = q;
          SCHED = LIFO, PREEMPT;
          SERVICE ( c1 ) = CST ( 10);
          SERVICE = EXP ( 5 );
          RATE = 1, 3, 5;
          ...
/EXEC/ SOLVE ("MVANCA");
```

NETWORK

NAME

NETWORK - Selects a sub-model which defines the sub-network on which the solution method is applied.

SYNTAX

```
NETWORK(q1,q2,...,qn)
```

DESCRIPTION

q1, q2, ... are QUEUE type objects. The procedure restricts the resolution to the queues named in the parameter list.

EVALUATION

During the execution.

ATTENTION

The parameters of the stations belonging to the sub-network have to reference only stations within the same sub-network.

VOIR AUSSI

MARKOV - SIMUL - SOLVE

EXEMPLE

```
/DECLARE/ QUEUE A,B,C;
/STATION/ NAME=A;
            ...
          TRANSIT=C;
/EXEC/ BEGIN
       NETWORK(A,B);
& Incorrect statement because A references C
& which is outside the network (A,B)
       SOLVE;
       END;
```

NAME

"PRIORPR" - Algorithm for preemptive priority scheduling.

SYNTAX

```
SOLVE( "PRIORPR" );
```

DESCRIPTION

This method implements an algorithm developed by M. Veran for the solution of closed multi-class queueing networks with preemptive priority scheduling.

PRIORPR is a fast, approximate and numerically safe solver.

Only closed networks may be analysed.

Application stipulations summary :

station type	scheduling	service distribution	service per class	global serv. rate	serv. rate per class
SINGLE	FIFO	exponential	no	no	no
	LIFO,PREEMPT	general	yes	no	no
	PRIOR,PREEMPT	exponential	yes	yes	yes
	PS	general	yes	no	no
INFINITE		generale	yes	yes	no

EVALUATION

During the execution.

WARNING

- This method may be used only for steady-state analysis.
- Only one PRIOR, PREEMPT station is allowed in the network.

SEE ALSO

SOLVE - "CONVOL" - "MVA" - "MVANCA" - "HEURSNC" - "ITERATIV" - "DIFFU" - "SPLITMAT" - MARKOV - SIMUL

EXAMPLE

```
/DECLARE/ QUEUE q;
          CLASS c1, c2;

/STATION/ NAME = q;
          SCHED=PRIOR,PREEMPT;
          PRIOR ( c1 ) = 2;
          SERVICE ( c1 ) = EXP( 10);
          SERVICE = EXP ( 5 );
          ...

/EXEC/ SOLVE ("PRIORPR");
```

NAME

SIMUL - Discrete event simulation.

SYNTAX

SIMUL;

DESCRIPTION

This procedure calls the discrete event simulation.

The simulator is based on a scheduler and a random number generator used particularly for the service time distribution.

The results are displayed in a standard table. They represent the mean value of the different criteria. The user may request the computation of the confidence intervals (ACCURACY parameter).

EVALUATION

During the execution.

NOTES

- The simulation may be used for steady-state and transient analysis.
- No application stipulation exists.

WARNING

The simulation start requires that the simulation duration has been defined previously using TMAX.

SEE ALSO

SOLVE - MARKOV - TMAX - ACCURACY

EXAMPLE

```
/DECLARE/ QUEUE q1, q2, q3,sem;

/STATION/ NAME = q1;
          SERVICE = BEGIN
                      P ( sem );
                      EXP ( 10 );
                      V ( sem );
                      IF CUSTNB ( q2 ) > CUSTNB ( q3 ) THEN
                        TRANSIT ( q3 )
                      ELSE
                        TRANSIT( q2 );
                    END;

/CONTROL/ TMAX = 10000 ; & mandatory before the simulation call
/EXEC/ SIMUL;
```

NAME

SOLVE - Analytical solvers

SYNTAX

SOLVE [(" *key-word* ")];

DESCRIPTION

This procedure calls an analytical solver.

Without argument, this procedure provides an automatical dispatching process in order to select the *best* solver according to the characteristics of the queueing network.

key-word represents one method among the 8 ones available :

- "CONVOL" : convolution algorithms
- "MVA" : mean value analysis algorithm
- "MVANCA" : mean value analysis and normalized convolution algorithms
- "PRIORPR" : algorithm for preemptive priority scheduling
- "HEURSNC" : heuristic algorithms
- "ITERATIV" : iterative algorithms
- "DIFFU" : diffusion algorithms
- "SPLITMAT" : SPLIT-MATCH approximation solver

EVALUATION

During the execution.

NOTES

All the solvers are fast.

WARNING

These methods may be used only for steady-state analysis.

DIAGNOSIS

- If the requested method (using the key-word) can not be applied, then an error message displays that the analysis is impossible.
- If the procedure is called without argument and if no method is possible for the queueing network, then an error message displays that an analytical resolution is impossible.

SEE ALSO

"CONVOL" - "MVA" - "MVANCA" - "HEURSNC" - "ITERATIV" - "DIFFU" - "SPLITMAT" - MARKOV - SIMUL

EXAMPLE

```
/DECLARE/ QUEUE q;
          CLASS c1, c2;

/STATION/ NAME = q;
          SCHED = PRIOR, PREEMPT;
          PRIOR ( c1 ) = 2;
          SERVICE ( c1 ) = EXP ( 10);
          SERVICE = EXP ( 5 );
          ...

/EXEC/ SOLVE;             & selects automatically the most convenient method
   ...                    & and calls this method
/EXEC/ SOLVE ("PRIORPR"); & calls the PRIORPR method
```

NAME

"SPLITMAT" - Split-Match approximation solver.

SYNTAX

```
SOLVE ( "SPLITMAT" );
```

DESCRIPTION

This method implements a bounding algorithm developped by F. Baccelli (INRIA). It applies to open, series-parallel networks with exponential FIFO single server stations.

DIFFU is a fast, approximate and numerically safe solver.

Only open networks may be analyzed.

Application stipulations summary :

station type	scheduling	service distribution	service per class	global serv. rate	serv. rate per class
SINGLE	FIFO	exponential	no	no	no
SOURCE	not relevant	exponential	not relevant	no	not relevant

EVALUATION

During the execution.

WARNING

See User's Guide for the application stipulations of this solver.

SEE ALSO

SOLVE - "CONVOL" - "MVA" - "MVANCA" - "HEURSNC" - "PRIORPR" - "ITERATIV" - "DIFFU" - MARKOV - SIMUL

EXAMPLE

```
/DECLARE/ QUEUE F,P1,P2,J,Q;
          CLASS C0,C1,C2;

/STATION/ NAME=F; TYPE=SOURCE;
          SERVICE=EXP(1.5);
          SPLIT=(P1,C1,1,P2,C2,1);

/STATION/ NAME=P1;
          SERVICE=EXP(1.4);
          TRANSIT=J;

/STATION/ NAME=P2;
          SERVICE=EXP(0.01);
          TRANSIT=J;

/STATION/ NAME=J;
          SERVICE=EXP(1.0);
          MATCH=(F):(C1,1,C2,1) C0;
          TRANSIT=Q;

/STATION/ NAME=Q;
          SERVICE=EXP(1.2);
          TRANSIT=OUT;

/EXEC/ BEGIN
         SOLVE("SPLITMAT");
       END;
/END/
```

7.4.3 Synchronization Procedures

PMULT	Multiple requests of pass grants to one or several semaphores or resource units.
P	Request of a pass grant to a semaphore, or a resource unit.
VMULT	Multiple releases of resource units and-or multiple production of pass grants to one or several semaphores.
V(resource)	Release a resource unit.
V(semaphore)	Production of a pass grant for a semaphore.
AFTCUST	Queueing a customer after another in a station.
BEFCUST	Queueing a customer before another in a station.
BLOCK	Station locking or unlocking
FLINKCUS	Access the next customer waiting on a flag.
FLISTCUS	Access the first customer waiting on a flag.
FREE	Customer unblocking.
JOIN	Definition of rendezvous with son customers.
MOVE	Transition of the first customer of a queue to another queue.
PRIOR	Update of the customer priority level.
SET	Set a flag to "set" or "unset" state.
SKIP	Jump a limited capacity station to the following station.
TRANSIT	Transition of a customer to a station.
WAIT	Define waiting conditions on flags state.

PMULT

NAME

PMULT - Multiple requests of pass grants to one or several semaphores or resource units.

SYNTAX

PMULT ([*customer*,] [*list_of_integers*,] *list_of_queues* [,*list_of_classes*] [,*list_of_integers*]) ;

DESCRIPTION

This procedure does in an indivisible way a whole set of P requests to one or several semaphores and-or resources. It is possible to :

1. Request several pass grants or units to the same resource or the same semaphore. All these requests are done with the same class and the same priority level (for one semaphore or resource specification).
2. Make a set of requests, respectively of one or several units, to various semaphores and-or resources (these requests can concern semaphores and some others resources). Every request to a semaphore or resource can be done with different numbers, different classes and different priority levels.

Once a multiple request has been performed by PMULT, all the free semaphores and-or resource units that were requested are immediately granted (in case of a resource, it is reserved; in case of a semaphore, its counter is immediately decremented). If all the requests have been immediately satisfied, it is impossible that the customer ever wait. Otherwise, the customer will wait until all its requests are granted. Every blocked request is satisfied when a semaphore or resource unit is free and not allocated to another customer better placed in the queue. The customer is freed when all its requests are satisfied.

The procedure arguments are :

- The first argument, optional, is the customer doing the request. By default, it is the current customer.
- The second argument, optional, is a list of integers, whose size has to be equal to the size of the following list. Each integer represents the priority level respectively assigned to the set of requests to each semaphore or resource. Its use is analogous to the optional fourth argument of the P procedure (see this procedure for more details). By default, the priority level of the customer doing the request is applied.
- The third argument, mandatory, represents the list (possibly restricted to a single element) of requested semaphores and-or resources.
- The fourth argument, optional, is a list of classes respectively assigned to each set of requests of semaphore or resource. Its size has to be equal to the one of the previous list. By default, the class of the customer performing the request is applied.

- The fifth argument, optional, is a list of integers representing the number of semaphore pass grants or resource units respectively requested to each semaphore or resource. The size of this list has to be equal to the one of the list of semaphores or resources. By default, a list of 1 is applied.

EVALUATION

During the execution.

NOTES

- If a PMULT operation is requested, the customer's service can be interupted, but the server can not be allocated to another customer (except in case of preemption)
- A customer which is blocked on a PMULT operation can be freed either by a VMULT, a FREE or by a succession of V.
- The results obtained on the semaphores and-or resources are calculated according to the same rules as the P procedure case.
- The PMULT operation is strictly equivalent to an indivisible succession of P operations, that will never be interrupted and that will be totally fulfilled, even if some of these operations are blocking.

SEE ALSO

P - V - FREE - VMULT

EXAMPLE

```
/DECLARE/ QUEUE q1, q2, sem1, sem2, res;
          CLASS c1, c2;
          REF CUSTOMER rc;

/STATION/ NAME = sem1;
          TYPE = SEMAPHORE, MULTIPLE(2);
          SCHED = PRIOR;

/STATION/ NAME = sem2;
          TYPE = SEMAPHORE, MULTIPLE(4);

/STATION/ NAME = res;
          TYPE = RESOURCE;

/STATION/ NAME = q1;
          SERVICE =

/STATION/ NAME = q2;
```

```
          SERVICE =
          BEGIN

          rc:=q1.FIRST;
  PMULT(rc,(3,1),(sem1,sem1),(c2,c1),(1,1));
  PMULT((sem1,sem2,res),(1,2,1));

& the current customer in the q2 station forces the request by the
& first customer in the q1 station of 2 pass grants to the sem1
& semaphore, one with class c2 and priority level 3 and the other with
& class c1 and priority level 1, then it requests 1 pass grant to the
& sem1 semaphore, 2 pass grants to the sem2 semaphore and 1 res
& resource unit

END;
```

NAME

P - Request of a pass grant to a semaphore, or a resource unit.

SYNTAX

P ([*customer*,] *queue* [, *class*] [, *integer*]) ;

DESCRIPTION

This procedure forces a customer, specified by the first argument, to ask:
- for a pass grant to the semaphore queue
- for a resource unit of a resource queue

depending of the *queue* TYPE. The MULTIPLE parameter of the queue specifies the number of pass grants, for a semaphore, or resource units, for resource. If the customer is not specified, the procedure is applied to the current customer.

Description of the request process:

- *queue* TYPE is SEMAPHORE:
  - If the counter of the semaphore is positive, the customer goes on with its treatement and the semaphore counter is decremented by 1.
  - If the counter of the semaphore is null or negative, the customer is blocked, a son is created and sent in the semaphore queue with a default priority equal to the initial customer priority.
- *queue* TYPE is RESOURCE: A son customer is always created and sent in the resource queue with a default priority equal to the initial customer priority.
  - If a resource unit is available it is allocated to the customer which can going on with its service.
  - If no resource unit is available the customer is blocked.

integer represents the priority level affected to the customer.

queue is an expression representing a QUEUE type entity.

customer is an expression representing a CUSTOMER type entity.

class is an expression representing a CLASS type entity.

integer is an expression representing a INTEGER type entity.

EVALUATION

During the execution

NOTES

- If a P operation is requested on a customer during its service execution, the service is interrupted, but the server cannot be allocated to another customer (except if preemption has been defined for the station scheduling).
- A customer blocked on a P operation can be freed by operations such as V and FREE.
- The standard results obtained on the *queue* station, concern all customers which have performed a P operation, successfully or not.

WARNING

This procedure can be used only during a simulation execution, in a service, never in a TEST algorithmic sequence.

SEE ALSO

SCHED - TEST - V - FREE - TYPE

DIAGNOSTICS

An error message is edited and the simulation excecution is stopped if the reference to the customer is NIL or if the customer has been deleted. The treatement is the same if the specified queue is neither a resource and nor a semaphore, or if the queue is referenced by a NIL value.

EXAMPLE

```
/DECLARE/  QUEUE q, res1, res2;
           CLASS c1, c2;
           INTEGER i;
           REF CUSTOMER rc;

/STATION/ NAME = res1;
          TYPE = RESOURCE;

/STATION/ NAME = res2;
          TYPE = SEMAPHORE, MULTIPLE (3);
          SCHED = PRIOR;

/STATION/ NAME = q;
          SERVICE =
BEGIN
  ...
  P (res1);              & request of res1 resource unit request of a pass
  P (rc, res2, c2, 10);  & grant to res2 semaphore a son customer is created
                         & with a class c2 and a priority level equal to 10
                         & the son is sent in res2
END;
```

NAME

VMULT - Multiple releases of resource units and-or multiple production of pass grants to one or several semaphores.

SYNTAX

VMULT ([*customer*,] *list_of_queues* [,*list_of_integers*]) ;

DESCRIPTION

This procedure is equivalent to an indivisible set of V operations on one or several resources and-or semaphores.
The VMULT command causes the production of pass grants to semaphores and the release of requested units to the resources. These processes are analogous to those of the V command.
The procedure arguments are :

- The first argument, optional, is the customer executing the operation. By default, it is the current customer.
- The second argument, mandatory, represents the list (possibly restricted to a single element) of the semaphores and-or resources to which the productions of pass grants and resource units releases are done.
- The third argument, optional, is a list of integers representing the number of releases respectively done to each semaphore and-or resource (i.e. the second integer in the list corresponds to the number of releases of the second queue). Therefore, the size of this list has to be equal to the one of the previous list. By default, the argument is a list of 1.

EVALUATION

During the execution

NOTES

- A VMULT operation is equivalent to a succession of V operations that will never be interrupted by a higher priority customer.
- A VMULT can free every request of resource or semaphore previously performed by either a P or a PMULT operation.
- If a customer performs a VMULT operation on resources, he must have already performed a upper or equal number of P and-or PMULT operations on these resources (i.e. a customer must own a resource to release it). If this condition is not fulfilled, the simulation stops and displays an error message.

SEE ALSO

P - V - FREE - PMULT

EXAMPLE

```
/DECLARE/ QUEUE q1, q2, sem1, sem2, res;
          REF CUSTOMER rc;

/STATION/ NAME = sem1;
          TYPE = SEMAPHORE, MULTIPLE(2);

/STATION/ NAME = sem2;
          TYPE = SEMAPHORE, MULTIPLE(2);

/STATION/ NAME = res;
          TYPE = RESOURCE;

/STATION/ NAME = q1;
          SERVICE =
          BEGIN

          PMULT((sem1,sem2),(1,2));
          P(sem2);
          P(res);

          END;

/STATION/ NAME = q2;
          SERVICE =
          BEGIN

          rc:=q1.FIRST;
          VMULT(rc,(sem1,sem2,res),(1,3,1));

& The current customer on the q2 station forces the first customer of
& the q1 station to produce 1 pass grant for the sem1 semaphore, 3 pass
& grants for the sem2 semaphore and to release 1 unit of the res
& resource

END;
```

NAME

`V (resource)` - Release a resource unit.

SYNTAX

`V` ([*customer*,]*queue*) ;

DESCRIPTION

This procedure releases a resource unit requested before by a `P` operation.

The process to release a resource unit consists in:

- Destruction of the son customer waiting in the resource queue.
- If other customers are waiting for a resource unit in the queue, the free unit is allocated to the first one, and the corresponding father customer is freed.

`V` operation can be requested for a customer specified in first argument or, by default, for the current customer.

queue is an expression representing a `QUEUE` type entity.

customer is an expression representing a `CUSTOMER` type entity.

EVALUATION

During the execution

WARNING

- A customer must own a ressource to release it.
- This procedure can be used only during a simulation execution, in a service, but never in a `TEST` algorithmic sequence.
- A `V` operation has no consequence on the concerned customer, but if the freed unit resource is allocated to a customer with priority level greater then to the current one, the freed customer becomes the current customer.

SEE ALSO

`TEST - P - V (Semaphore)`

DIAGNOSIS

A `V` operation is available only if a `P` operation has been performed before, otherwise an error message is edited and the simulation is stopped.

EXAMPLE

```
/DECLARE/  QUEUE q, res1, res2 ;
           CLASS c1, c2 ;
           INTEGER i ;
           REF CUSTOMER rc ;

/STATION/  NAME = res1 ;
           TYPE = RESOURCE ;

/STATION/  NAME = res2 ;
           TYPE = RESOURCE, MULTIPLE (3) ;
           SCHED = PRIOR ;

/STATION/  NAME = q ;
           SERVICE =
BEGIN
  ...
  P (res1) ;              & request of res1 resource unit
  P (rc, res2, c2, 10) ; & request of res2 resource unit
                          & a son customer is created with a c2 class
                          & and a priority level equal to 10
  ...
  V (rc, res2) ;          & release of res2 resource unit for
                          & for rc customer
  V (res1) ;              & release of res1 resource unit for
                          & the current customer
END ;
```

NAME

`V (semaphore)` - Production of a pass grant for a semaphore.

SYNTAX

`V` (*queue*) ;

DESCRIPTION

This procedure produces of a pass grant for a semaphore.

The process is:

- If the counter is positive or null, it is incremented by 1.
- If the counter is negative, the first customer waiting in the semaphore queue is released, and the corresponding father customer goes on its service treatement. The counter is incremented by 1.

queue is an expression representing a type `QUEUE` entity.

EVALUATION

During the execution

NOTES

A `V` operation is available even if no `P` operation has been performed on the semaphore.

WARNING

- This procedure can be used only during a simulation execution, in a service, but never a `TEST` algorithmic sequence.
- A `V` operation has no consequence on the concerned customer, but if the freed unit resource is allocated to a customer with priority level greater than to the current one, the freed customer becomes the current customer.

SEE ALSO

`TEST - P - V (Resource)`

EXAMPLE

```
/DECLARE/  QUEUE q, res1, res2 ;
           CLASS c1, c2 ;
           INTEGER i ;
           REF CUSTOMER rc ;

/STATION/  NAME = res1 ;
           TYPE = SEMAPHORE;

/STATION/  NAME = res2 ;
           TYPE = SEMAPHORE, MULTIPLE (3) ;
           SCHED = PRIOR ;

/STATION/  NAME = q ;
           SERVICE =
BEGIN
  ...
  P (res1);              & request of a pass grant to res1 semaphore
  P (rc, res2, c2, 10);  & request of a pass grant to res2 semaphore
                         & a son customer is created with a class c2
                         & and a priorite level equal to 10, the son
                         & is sent in res2
  ...
  V (rc, res2) ;         & rc customer produces of a pass grant
                         & for res2 semaphore
  V (res1) ;             & production of a pass grant for res1 semaphore
END ;
```

NAME

AFTCUST - Queueing a customer after another in a station.

SYNTAX

AFTCUST (*customer1*,*customer2*, [,*class*] [,*integer*]) ;

DESCRIPTION

This procedure ordering the first specified customer after the second specified customer in a station queue.

A class specification assigns a new class to the first customer. This specification must be done if the customer comes from a SOURCE station.

integer representing the priority level assigned to the first customer.

customer1 et *customer2* are expressions representing a CUSTOMER type entities.

class is an expression representing a CLASS type entity.

integer is an expression representing a INTEGER type entity.

EVALUATION

During the execution

WARNING

This procedure can be used only in a simulation resolution in a service, but never in a TEST algorithmic sequence.

SEE ALSO

PRIOR (PARAMETRE) - SCHED - CPRIOR - BEFCUST -TRANSIT - MOVE -
TRANSIT (PARAMETRE) - TEST

DIAGNOSIS

An error message is edited and the simulation is stopped if:
- the scheduling of the station (SCHED parameter) is incompatible with the required order
- the second referenced customer is not contained in a station, or has been released or is referenced by NIL value.

EXAMPLE

```
/DECLARE/ QUEUE q1, q2, q3;
          CLASS c1, c2;
          INTEGER i;
          REF CUSTOMER rc;

/STATION/ NAME = q1;
          SERVICE =
 BEGIN
   ...
   IF q2. NB > 1 THEN
     AFTCUST (rc, q2.FIRST)
   ELSE
     AFTCUST (rc, q3.FIRST, c2, 4);
   ...
 END;
```

NAME

BEFCUST - Queueing a customer before another in a station.

SYNTAX

BEFCUST (*customer1*,*customer2*, [,*class*] [,*integer*]) ;

DESCRIPTION

This procedure orders the first specified customer before the second specified customer in a station queue.

A class specification assigns a new class to the first customer. This specification must be done if the customer comes from a SOURCE station.

integer representing the priority level assigned to the first customer.

customer1 et *customer2* are expressions representing a CUSTOMER type entities.

class is an expression representing a CLASS type entity.

integer is an expression representing a INTEGER type entity.

EVALUATION

During the execution

WARNING

This procedure can be used only in a simulation resolution in a service, but never in a TEST algorithmic sequence.

SEE ALSO

PRIOR (PARAMETRE) - SCHED - CPRIOR - AFTCUST -TRANSIT - MOVE -
TRANSIT (PARAMETRE) - TEST

DIAGNOSIS

An error message is edited and the simulation is stopped if:
- the scheduling of the station (SCHED parameter) is incompatible with the required order
- the second referenced customer is not contained in an station, or has been released or is referenced by NIL value.

EXAMPLE

```
/DECLARE/ QUEUE q1, q2, q3;
          CLASS c1, c2;
          INTEGER i;
          REF CUSTOMER rc;

/STATION/ NAME = q1;
          SERVICE =
BEGIN
  ...
  IF q2. NB > 1 THEN
    BEFCUST (rc, q2.LAST)
  ELSE
    BEFCUST (rc, q3.LAST, c2, 4);
END;
```

NAME

BLOCK , UNBLOCK , ISBLOCK - Station locking or unlocking

SYNTAX

BLOCK (*queue1, queue2, ..., queuen*)
UNBLOCK (*queue1, queue2, ..., queuen*)
ISBLOCK (*queue*)

DESCRIPTION

It is possible to block the activity of a station. If a station is blocked, all the customers in service are blocked. The waiting customers are also blocked. The procedure used to block one or several stations is BLOCK.

The procedure UNBLOCK unblocks one or several stations.

ISBLOCK permit to test the state of a station (blocked or not). This function returns a boolean which is TRUE if the station is blocked and FALSE in the other case.

queue1, queue2, ..., queuen are expressions representing QUEUE type entities.

EVALUATION

During the execution.

NOTES

- A customer in a blocked station can be freed by the procedures TRANSIT or MOVE.
- If a customer arrives in a blocked station with at least one free server, no server will be allocated before the end of the locking.
- The blocked time will appear only for the customers in service.

WARNING

These procedures can be used only during a simulation resolution.

EXAMPLE

```
/DECLARE/QUEUE MACHINE, FAILURE, GENPIECE;
         REAL T_MACH,INT_FAIL,T_FAIL,INT_PIEC;

/STATION/NAME=MACHINE;
         SERVICE=CST(T_MACH);
         TRANSIT=OUT;

/STATION/NAME=FAILURE;
         INIT=1;
         SERVICE=BEGIN
                  EXP(INT_FAIL);
                  BLOCK(MACHINE);
                  EXP(T_FAIL);
                  UNBLOCK(MACHINE);
                 END;
         TRANSIT=FAILURE;

/STATION/NAME=GENPIECE;
         TYPE=SOURCE;
         SERVICE=EXP(INT_PIEC);
         TRANSIT=MACHINE;
```

NAME

FLINKCUS - Access the next customer waiting on a flag.

SYNTAX

FLINKCUS (*customer[,flag]*)

DESCRIPTION

This function returns the reference to the customer following the customer defined by the first argument waiting on the flag.

If the second argument does not appear, the customer waits on a single condition (WAIT). If this condition does not hold (WAITOR or WAITAND), then the second argument has to be given. It defines the flag where the next customer must be sought. The first argument may be a variable or a function call; the second argument has to be a variable. It should be noted that for a single condition, the predeclared attribute LINK for customers is equivalent.

customer is an expression representing a CUSTOMER type entity.

flag is an expression representing a FLAG type entity.

EVALUATION

During the execution

NOTE

The function returns NIL if the customer is the last of the list.

WARNINGS

- This function can be used only during a simulation resolution.
- The waiting customer are waiting with a LIFO scheduling.

SEE ALSO

FLAG - FREE - SET - RESET -FLISTCUS - STATE - WAIT

EXAMPLE

```
/DECLARE/ FLAG F1,F2,F3;
          QUEUE Q1,Q2;
          CLASS X,Y,Z;
          REF CUSTOMER C;
          REF FLAG FL;

/STATION/NAME=Q1;
         TYPE=MULTIPLE(3);
         INIT=1;    & one customer per class
         SERVICE(X)=BEGIN
                  CST(1);
                  PRINT("customer : ",CUSTOMER);
                  WAITOR(F1,F3);
                 END;
         SERVICE(Y)=BEGIN
                  CST(1);
                  PRINT("customer : ",CUSTOMER);
                  WAITOR(F2,F3);
                 END;
         SERVICE(Z)=BEGIN
                  CST(1);
                  PRINT("customer : ",CUSTOMER);
                  WAIT(F3);
                 END;
         TRANSIT=OUT;

/STATION/NAME=Q2;
         INIT(X)=1;
         SERVICE=BEGIN
                  RESET(F1);RESET(F2);RESET(F3);
                  CST(2);
                  & printing of the content of the queues
                  & joined to the flags
                  FOR FL:=ALL FLAG DO
                   BEGIN
                    PRINT("flag : ",FL);
                    C:=FLISTCUS(FL);   & first customer on the flag
                    WHILE C<> ANIL DO
                     BEGIN
                      PRINT("customer : ",C,"  on flag : ",FL);
                      C:=FLINKCUS(C,FL);  & following customer
                     END;
```

```
                    END;
                  END;
          TRANSIT=OUT;

/CONTROL/TMAX=5;
         OPTION=NRESULT;

/EXEC/ BEGIN
        PRINT(" ");
        PRINT("Execution of the program produces :");
        PRINT(" ");
        SIMUL;
       END;
```

Execution of the program produces :

```
customer :    999346
customer :    999314
customer :    999282
flag : F1
customer :    999346   on flag : F1
flag : F2
customer :    999314   on flag : F2
flag : F3
customer :    999282   on flag : F3
customer :    999314   on flag : F3
customer :    999346   on flag : F3
```

FLISTCUS

NAME

FLISTCUS - Access the first customer waiting on a flag.

SYNTAX

FLISTCUS(*flag*)

DESCRIPTION

This function returns the reference to the first customer waiting on the flag given by the argument.

The customer may wait on a single condition (WAIT) or on multiple conditions (WAITOR or WAITAND).

flag is an expression representing a FLAG type entity.

EVALUATION

During the execution

NOTES

- The function returns NIL if no customer waits on the flag.
- The function returns a customer reference if the flag state is "reset".

WARNINGS

- This function can be used only during a simulation resolution.
- For the queue joined to a flag being scheduled in LIFO, the first customer is in fact the customer blocked most recently.

SEE ALSO

FLAG - FREE - SET - RESET -FLINKCUS - STATE - WAIT

EXAMPLE

```
/DECLARE/ FLAG F1,F2,F3;
          QUEUE Q1,Q2;

/STATION/NAME=Q1;
         TYPE=MULTIPLE(3);
         INIT=3;
         SERVICE=BEGIN
                  CST(1);
                  PRINT("customer : ",CUSTOMER);
                  WAITOR(F1,F2,F3);
                 END;
         TRANSIT=OUT;

/STATION/NAME=Q2;
         INIT=1;
         SERVICE=BEGIN
                  RESET(F1);RESET(F2);RESET(F3);
                  CST(2);
                  PRINT("FLISTCUS(F1) : ",FLISTCUS(F1));
                 END;
         TRANSIT=OUT;

/CONTROL/TMAX=5;
         OPTION=NRESULT;

/EXEC/ BEGIN
         PRINT(" ");
         PRINT("Execution of the program produces :");
         PRINT(" ");
         SIMUL;
        END;
```

Execution of the program produces :

```
customer :    999410
customer :    999378
customer :    999346
FLISTCUS(F1) :    999346
```

FREE

NAME

FREE - Customer unblocking.

SYNTAX

FREE(*customer[, queue | flag]*)

DESCRIPTION

This procedure cancels the specified waiting conditions concerning the customer referenced by the first argument.

If the optional parameter *queue | flag* is omitted, a call to the FREE procedure frees the customer referenced by *customer* from all its simple and compound waiting conditions due to flags, JOIN synchronizations, and from its current service time (if any in progress). The procedure is ineffective if customer is not currently waiting on any of those conditions nor executing a service time.

If the parameter queue is present, the customer is freed only from the specified semaphore or resource queue. The parameter flag is restricted to a flag appearing in a simple waiting condition (WAIT procedure). When present, it frees the customer from its simple waiting condition on the specified flag.

- If the second argument does not appear, then the customer given by the first argument is freed from the flag, the join synchronization or the current service.
- If the second argument is a queue, the customer is freed only from the specified semaphore or resource queue.
- If the second argument is a flag, the customer is freed from the flag if it was blocked by the WAIT procedure.

The extensions related to the use of the FREE procedure for the customer blocked by flags after a call to the WAITOR procedure or WAITAND procedure. In this case the flag given in the second argument is removed from the list of the flags defined for the waiting condition. If the flag was the last one in the list, or if the flag was the single one in the state RESET in the case of a WAITAND, then the customer is freed from the flags remaining in the list.

EVALUATION

During the execution.

WARNINGS

- These procedures can be used only during a simulation resolution (it is forbidden in a TEST algorithmic sequence).
- If the customer was blocked on semaphore request, the FREE procedure has no consequence on the semaphore counter.
- The current customer can be replaced by the releasing of a customer with a higher priority level.

SEE ALSO

FLAG - SEMAPHORE - RESOURCE - QUEUE - P - WAIT

EXAMPLES

Example 1 :

```
/DECLARE/ FLAG F1,F2,F3;
          QUEUE Q1,Q2;

/STATION/NAME=Q1;
         INIT=1;
         SERVICE=BEGIN
                   CST(1);
                   WAITOR(F1,F2,F3);
                   PRINT("waitor is passed at time : ",TIME);
                   WAITAND(F1,F2,F3);
                   PRINT("waitand is passed at time : ",TIME);
                 END;
         TRANSIT=OUT;

/STATION/NAME=Q2;
         INIT=1;
         SERVICE=BEGIN
                  RESET(F1);RESET(F2);RESET(F3);
                  CST(2);
                  PRINT("FREE on F2 at time : ",TIME);
                  FREE(Q1.FIRST,F2);
                  & the customer in the queue Q1 makes
                  & in fact now WAITOR(F1,F3)
                  CST(1);
                  PRINT("SET on F3 at time : ",TIME);
                  SET(F3);   & the flag F3 is in state "set", so the first
                             & customer in Q1 will be freed
                  CST(1);
                  PRINT("SET on F2 at time : ",TIME);
                  SET(F2);   & the flag is in state "set"
```

```
                    CST(1);
                    PRINT("FREE on F1 at time : ",TIME);
                    FREE(Q1.FIRST,F1);
                    & the customer in the queue Q1 makes in fact now
                    & WAITAND(F2,F3); F2 and F3 are in state "set", so the
                    & customer is freed
                   END;
          TRANSIT=OUT;

/CONTROL/TMAX=10;
         OPTION=NRESULT;

/EXEC/ BEGIN
        PRINT("Execution of the program produces :");
        PRINT(" ");
        SIMUL;
       END;
```

Execution of the program produces :

```
FREE on F2 at time :    2.000
SET on F3 at time :    3.000
waitor is passed at time :    3.000
SET on F2 at time :    4.000
FREE on F1 at time :    5.000
waitand is passed at time :    5.000
```

Example 2 :

```
& releasing all the customer
& of class X customers waiting
& on a flag.
&
/DECLARE/ QUEUE A, B;CLASS X;
          FLAG F; REF CUSTOMER C;

/STATION/ NAME = A;
          SERVICE = WAIT (F);
&           ...

/STATION/ NAME = B;
          SERVICE =
              BEGIN
              C := F.LIST;
              WHILE C <> NIL DO
                  BEGIN
                  IF C.CCLASS=X THEN
                     FREE(C,F);
                  C := C.LINK;
                  END;
              END;
```

The customers entering station A wait on flag F. The customer in B releases all the class X customers waiting.

JOIN

NAME

JOIN , JOINC - Definition of rendezvous with son customers.

SYNTAX

JOIN [(*integer* | *customer1*, *customer2*, ...)] ;
JOINC (*customer* [*integer* | *customer1*, *customer2*, ...]) ;

DESCRIPTION

These procedures cause the current customer (i.e. JOIN), or the customer specified in first argument (i.e. JOINC) to wait until all, or only the specified son customers: *customer1*, *customer2*, have been destroyed (transit OUT).

If an *integer* argument is used, the current , or the specified, customer will wait until the specified number of son customers have been destroyed (transited OUT).

customer, *customer1* et *customer2* are expressions representing CUSTOMER type entities.

integer is an expression representing INTEGER type entity.

EVALUATION

During the execution

NOTES

- A son customer is created by using the NEW function during a service execution.
- If *integer* is greater than the current number of son customers, JOIN or JOINC operation will concern only the existing son customers.

WARNING

This procedure can be used only in a simulation resolution in a service algorithmic sequence. It is forbidden in a TEST algorithmic sequence.

SEE ALSO

NEW - OUT

EXAMPLE

```
/DECLARE/ QUEUE q1, q2, q3;
          INTEGER i = 1;
          REF CUSTOMER rc, rc1, rc2;

/STATION/ NAME = q1;
          INIT = 1;
          SERVICE =
 BEGIN
   TRANSIT (NEW (CUSTOMER), q3);
   rc1 := NEW (CUSTOMER);
   TRANSIT (rc1, q3);
   rc2 := NEW (CUSTOMER);
   TRANSIT (rc2, q3);
   JOIN (rc1, rc2);              & current customer is blocked until
                                 & rc1 and rc2 be  OUT
   JOIN;                         & current customer is blocked until
                                 & all son customers be OUT
   ...
   rc := NEW (CUSTOMER);
   TRANSIT (rc, q);
   JOINC (rc,2);                 & rc customer is blocked until
                                 & two son customers be OUT
END;

/STATION/ NAME = q2;
          TRANSIT = OUT;
          SERVICE =
 BEGIN
   TRANSIT (NEW (CUSTOMER), q3);
   TRANSIT (NEW (CUSTOMER), q3);
   TRANSIT (NEW (CUSTOMER), q1);
END;
```

MOVE

NAME

MOVE - Transition of the first customer of a queue to another queue.

SYNTAX

MOVE (*queue1*, *queue2*) ;

DESCRIPTION

This procedure moves the first customer of queue specified in first argument to the specified queue specified in second argument.

queue1 et *queue2* are expressions representing QUEUE type entities.

EVALUATION

During the execution.

NOTES

- A MOVE operation on the first customer stops the current service treatement.
- The operation has no consequence if the original queue is empty.

WARNINGS

- The procedure can be used only during a simulation execution.
- It is not possible to send a blocked customer out of the network (OUT).

SEE ALSO

TRANSIT - TEST - TRANSIT (STATION)

EXAMPLE

```
 /DECLARE/ QUEUE q1, q2, q3;

/STATION/ NAME = q;
  SERVICE =
  BEGIN
    ...
    IF q2. NB > q3. NB THEN
      MOVE (q2, q3)
    ELSE
      MOVE (q3, q2);
    ...
  END;
```

NAME

PRIOR - Update of the customer priority level.

SYNTAX

PRIOR ([*customer*,] *integer*) ;

DESCRIPTION

This procedure modifies the current, or the specified, customer priority level or the timer priority level.

integer corresponding to the new priority level.

customer is an expression representing a CUSTOMER type or a TIMER type entity.

EVALUATION

During the execution.

NOTES

- The modification of a customer priority level can modify the scheduler organization, and a higher priority customer can become the current customer.
- If the specified customer is waiting in the queue with a priority management (SCHED = PRIOR) the order of the customers in the queue can be modified.

WARNING

This procedure can be used only during a simulation execution, in a service algorithmic sequence.

SEE ALSO

PRIOR (STATION) - SCHED - CPRIOR - TIMPRIOR

EXAMPLE

```
/DECLARE/ QUEUE q;
          INTEGER i;
          REF CUSTOMER rc;

/STATION/ NAME = q;
          SCHED = PRIOR;
          SERVICE =
  BEGIN
    i : = CPRIOR;
    PRIOR (9999);   & maximal priority level, the current customer
    ...             & cann't be interrupted by an instantaneous event

    PRIOR (i);      & initial priority level
    ...
    PRIOR (rc, 8);  & modification of an other customer priority level
    ...
 END;
```

NAME

SET , RESET - Set a flag to "set" or "unset" state.

SYNTAX

SET (*flag*) ;
RESET (*flag*) ;

DESCRIPTION

These procedures set a flag to "set" or "unset" state.

If the specified flag state is "unset", by RESET operation, all WAIT, WAITAND or WAITOR operations will block the concerned customer until a SET or a FREE procedure.

If the specified flag state is "set", by SET operation, all WAIT, WAITAND or WAITOR operations will have no consequence until a RESET procedure.

flag is an expression representing a FLAG type entity.

EVALUATION

During the execution.

NOTES

- These procedures are without consequences for the current customers.
- UNSET is equivalent to RESET.

WARNINGS

- These procedures can be used only during a simulation resolution (it is forbidden in a TEST algoritmic sequence).
- The customers are released with a LIFO scheduling.
- The current customer can be replaced by the released customer with a higher priority level.

SEE ALSO

FLAG - WAIT - WAITAND - WAITOR - FREE - STATE

EXAMPLE

```
/DECLARE/ QUEUE q1, q2;
          FLAG f;

/STATION/ NAME = q1;
          INIT = 1;
          TRANSIT = q1;
          SERVICE =
 BEGIN
   CST (10);
   SET (f);    & f is in ``set'' state, customer are not blocked
   CST (200);  & during 200 unities of time
   RESET (f);  & f is in ``reset'' state, customer can be blocked
 END;
```

NAME

SKIP - Jump a limited capacity station to the following station.

SYNTAX

```
REJECT  = SKIP
```

DESCRIPTION

This procedure specifies to a rejected customer that he must jump the limited capacity station, and must move to the following station, specified by parameter TRANSIT of the limited capacity station.

SKIP can be used only in the algorithmic sequence defining the treatement of customers rejected of a limited capacity station.

EVALUATION

During the resolution.

NOTES

SKIP treatement is the necessary and unique reject treatement accepted for a mathematical resolution.

SEE ALSO

CAPACITY - REJECT

EXAMPLE

```
/STATION/ NAME = q1;
          CAPACITY = 5;
          TRANSIT = q2;
          REJECT = SKIP;
```

TRANSIT

NAME

TRANSIT - Transition of a customer to a station.

SYNTAX

TRANSIT ([*customer*,] *queue* [, *class*] [, *integer*]) ;

DESCRIPTION

This procedure moves the current, or the specified, customer from the current station to a station specified by the second argument.

A class specification assigns a new class to the customer. This specification must be done if the customer comes from a SOURCE station.

integer representing the priority level assigned to the first customer.

queue is an expression representing a QUEUE type entity.

customer is an expression representing a CUSTOMER type entity.

class is an expression representing a CLASS type entity.

integer is an expression representing an INTEGER type entity.

EVALUATION

During the execution

NOTES

- The modification of a customer priority level can modify the scheduler organization, and the current customer can be replaced by an other higher priority customer.
- A TRANSIT operation on the current customer stops the current service treatement.

WARNINGS

- This procedure can be used only during a simulation resolution (it is forbidden in a TEST algoritmic sequence).
- It is not possible to send a blocked customer out of the network (OUT).

SEE ALSO

PRIOR (PARAMETRE) - SCHED - CPRIOR - MOVE - TRANSIT (PARAMETRE) - TEST

DIAGNOSIS

An error message is edited and the simulation is stopped if the referenced customer has been released or is referenced by NIL value.

EXAMPLE

```
/DECLARE/  QUEUE q1, q2, q3;
           CLASS c1, c2;
           INTEGER i;
           REF CUSTOMER rc;

/STATION/  NAME = q1;
           SERVICE =
BEGIN
  ...
  IF q2. NB > q3. NB THEN
    BEGIN
      TRANSIT (q2.LAST, q3);
      TRANSIT (q2);
    END;
  ELSE
    TRANSIT (q3, c2);
  ...
END;
```

WAIT

NAME

WAIT , WAITAND , WAITOR - Define waiting conditions on flags state.

SYNTAX

WAIT ([*client*,] *flag*) ;
WAITAND ([*client*,] *flag1*, *flag2*, ...) ;
WAITOR ([*client*,] *flag1*, *flag2*, ...) ;

DESCRIPTION

These procedures define waiting conditions on flags state. If only one flag state is "unset" the current, or specified, customer have to wait, excepted if WAITOR procedure is used; in this case the customer is blocked only if all specified flags are in "unset" state.

If all flags state is "set" the operation has no consequence on the customer.

If the customer is blocked, only SET and FREE procedures can release it.

client is an expression representing a CUSTOMER type entity.

flag, *flag1* and *flag2* are expressions representing FLAG type entities.

EVALUATION

During the execution

NOTE

FLISTCUS and FLINKCUS functions reference any customer wait on a FLAG entity.

WARNINGS

- These procedures can be used only during a simulation resolution (it is forbidden in a TEST algoritmic sequence).
- The waiting customer are waiting with a LIFO scheduling.
- A flag is in "unset" state at the beginning of a simulation.

SEE ALSO

FLAG - FREE - SET - RESET -FLISTCUS - FLINKCUS - STATE

EXAMPLE

```
/DECLARE/ QUEUE q1, q2;
          FLAG f1,f2,f3;
          CLASS c1,c2,c3;

/STATION/ NAME = q1;
          INIT = 1;
          TRANSIT = q1;
          SERVICE =
 BEGIN
   CST (10);
   SET (f1);           & f1 ''set''
   CST (20);
   SET (f2);           & f2 ''set''
   CST (10);
   SET (f3);           & f2 ''set''
   CST (200);
   RESET (f1,f2,f3);   & blocking flags
 END;

/STATION/ NAME = q2;
          TYPE = INFINITE;
          SERVICE (c1) =  BEGIN
                            ...
                            WAIT (f1);           & customer is blocked if
                                                 & f1 state is ''unset''
                            ...
                          END;
          SERVICE (c2) =  BEGIN
                            ...
                            WAITAND (f1,f2,f3);  & customer is blocked if
                                                 & f1 or f2 or f3 state
                                                 & is ''unset''
                            ...
                          END;
          SERVICE (c3) =  BEGIN
                            ...
                            WAITOR (f1,f2,f3);   & customer is blocked if
                                                 & f1, f2 and f3 state
                                                 & is ''unset''
                            ...
                          END;
```

7.4.4 Results

CBLOCKED	Returns the confidence interval values on blocking time.
CBUSYPCT	Returns the confidence interval values on busy percentage of the station servers.
CCUSTNB	Returns the confidence interval values on the mean number of customers in the station.
CRESPONSE	Returns the confidence interval values on response time.
CSERVICE	Returns the confidence interval values on service time.
CUSTNB	Returns the number of customers in the station.
MAXCUSTNB	Returns the maximum number of customers in a station.
MBLOCKED	Returns the mean value of blocking time in the station.
MBUSYPCT	Returns the mean busy percentage of the station servers.
MCUSTNB	Returns the mean number of customers in the station.
MRESPONSE	Returns the mean value of response time in the station.
MSERVICE	Returns the mean value of service time in the station.
MTHRUPUT	Returns the mean value of the station throughput.
OUTPUT	Requests the edition of QNAP2 standard results.
PCUSTNB	Access to marginal probabilities of a station.
PMXCUSTNB	Access to the maximum number of customers in a station during the last simulation period.
SERVNB	Access to the number of customers served during a simulation.
SONNB	Acces to the number of a customer's sons.
VCUSTNB	Access to the variance of the mean number of customers in a station.
VRESPONSE	Access to the variance on the mean response time of a station.

NAME

CBLOCKED - Returns the confidence interval values on blocking time.

SYNTAX

CBLOCKED (*queue* [, *class*])

DESCRIPTION

The function returns a real value corresponding to the confidence interval length obtained on the blocking time mean.

If a class is specified among the arguments, the returned value corresponds to the confidence interval length obtained on the blocking time mean for the class in the queue.

The blocking time is concerned by operations such as: P, WAIT, WAITAND, WAITOR, JOIN, JOINC and BLOCK which can block the service of the customer.

queue is an expression of QUEUE type.

class is an expression of CLASS type.

EVALUATION

During the execution.

WARNING

- This function can be used only during a simulation resolution.
- The values, used to compute statistical results, are collected between the end of the initialization period (or the beginning of a period if periodic results have been requested) and the last call to SAMPLE function.
- Confidence interval can be obtained for a class of customers only if the mean has been requested by operations such as: /CONTROL/ CLASS = ... or SETSTAT:CLASS.
- The result is available for customers which have left the station.

SEE ALSO

ACCURACY - STATISTICS - CLASS - SETSTAT:BLOCKED:ACCURACY - GETSTAT:BLOCKED:ACCURACY

EXAMPLE

```
 /DECLARE/ QUEUE q1, q2;
           CLASS c1, c2;
           REF CLASS rc;
           REAL r;

/CONTROL/ TMAX =1000;
          TSTART = 100;
          ESTIM = REGEN;
          PERIOD = 200;
          TEST =
  BEGIN
    SAMPLE;
    ...
    PRINT (CBLOCKED (q1));     & confidence interval on blocking time
                               & of q1
    ...
    r : =  CBLOCKED (q2, c1);  & confidence interval on blocking time
                               & of q2 for class c1 customers
    ...
  END;


/EXEC/ BEGIN
         SETSTAT:QUEUE (q1,q2);
         SETSTAT:CLASS (q1,c1);
         SESTAT:ACCURACY (q1,c1);
         SESTAT:ACCURACY (q1,q2);
         ...
         SIMUL;
         PRINT (CBLOCKED (q2));  & confidence interval on blocking time
                                 & for q2 station
      END;
```

NAME

CBUSYPCT - Returns the confidence interval values on busy percentage of the station servers.

SYNTAX

CBUSYPCT (*queue* [, *class*])

DESCRIPTION

The function returns a real value corresponding to the confidence interval length obtained on busy percentage of the station servers.

If a class is specified among the arguments, the returned value corresponds to the confidence interval length obtained on the busy percentage of the station servers for the specified class.

queue is an expression of QUEUE type.

class is an expression of CLASS type.

EVALUATION

During the execution.

WARNING

- This function can be used only during a simulation resolution.
- The values, used to compute statistical results, are collected between the end of the initialization period (or the beginning of a period if periodic results have been requested) and the last call to SAMPLE function.
- Confidence interval can be obtained for a class of customers only if the mean has been requested by operations such as: /CONTROL/ CLASS = ... or SETSTAT:CLASS.
- The result is available for customers which have left the station.

SEE ALSO

ACCURACY - STATISTICS - CLASS - SETSTAT:BUSYPCT:ACCURACY - GETSTAT:BUSYPCT:ACCURACY

EXAMPLE

```
 /DECLARE/ QUEUE q1, q2;
           CLASS c1, c2;
           REF CLASS rc;
           REAL r;

/CONTROL/ TMAX =1000;
          TSTART = 100;
          ESTIM = REGEN;
          PERIOD = 200;
          TEST =
  BEGIN
    SAMPLE;
    ...
    PRINT (CBUSYPCT (q1));     & confidence interval on busy percentage
                               & of q1
    ...
    r : =  CBUSYPCT (q2, c1);  & confidence interval on  busy percentage
                               & of q2 for class c1 customers
    ...
  END;


/EXEC/ BEGIN
         SETSTAT:QUEUE (q1,q2);
         SETSTAT:CLASS (q1,c1);
         SESTAT:ACCURACY (q1,c1);
         SESTAT:ACCURACY (q1,q2);
         ...
         SIMUL;
         PRINT (CBUSYPCT (q2));  & confidence interval on  busy percentage
                                 & for q2 station
      END;
```

NAME

CCUSTNB - Returns the confidence interval values on the mean number of customers in the station.

SYNTAX

CCUSTNB (*queue* [, *class*])

DESCRIPTION

The function returns a real value corresponding to the confidence interval length obtained on the mean number of customers in the station.

If a class is specified among the arguments, the returned value corresponds to the confidence interval length obtained on the mean number of customers of a class in the queue.

queue is an expression of QUEUE type.

class is an expression of CLASS type.

EVALUATION

During the execution.

WARNING

- This function can be used only during a simulation resolution.
- The values, used to compute statistical results, are collected between the end of the initialization period (or the beginning of a period if periodic results have been requested) and the last call to SAMPLE function.
- Confidence interval can be obtained for a class of customers only if the mean has been requested by operations such as: /CONTROL/ CLASS = ... or SETSTAT:CLASS.
- The result is available for customers which have left the station.

SEE ALSO

ACCURACY - STATISTICS - CLASS - SETSTAT:CUSTNB:ACCURACY - GETSTAT:CUSTNB:ACCURACY

EXAMPLE

```
 /DECLARE/ QUEUE q1, q2;
           CLASS c1, c2;
           REF CLASS rc;
           REAL r;

/CONTROL/ TMAX =1000;
          TSTART = 100;
          ESTIM = REGEN;
          PERIOD = 200;
          TEST =
  BEGIN
    SAMPLE;
    ...
    PRINT (CCUSTNB (q1));      & confidence interval on the mean number
                               & of customers in q1 station.
    ...
    r : =  CCUSTNB (q2, c1);  & confidence interval on the mean number
                               & of class c1 customers in q2 station
    ...
  END;


/EXEC/ BEGIN
         SETSTAT:QUEUE (q1,q2);
         SETSTAT:CLASS (q1,c1);
         SESTAT:ACCURACY (q1,c1);
         SESTAT:ACCURACY (q1,q2);
         ...
         SIMUL;
         PRINT (CCUSTNB (q2));  & confidence interval on the mean
                                & number of customers in q2 station
      END;
```

NAME

CRESPONSE - Returns the confidence interval values on response time.

SYNTAX

CRESPONSE (*queue* [, *class*])

DESCRIPTION

The function returns a real value corresponding to the confidence interval length obtained on the response time mean.

If a class is specified among the arguments, the returned value corresponds to the confidence interval length obtained on the response time mean for the class in the queue.

queue is an expression of QUEUE type.

class is an expression of CLASS type.

EVALUATION

During the execution.

WARNING

- This function can be used only during a simulation resolution.
- The values, used to compute statistical results, are collected between the end of the initialization period (or the beginning of a period if periodic results have been requested) and the last call to SAMPLE function.
- Confidence interval can be obtained for a class of customers only if the mean has been requested by operations such as: /CONTROL/ CLASS = ... or SETSTAT:CLASS.
- The result is available for customers which have left the station.

SEE ALSO

ACCURACY - STATISTICS - CLASS - SETSTAT:RESPONSE:ACCURACY - GETSTAT:RESPONSE:ACCURACY

EXAMPLE

```
 /DECLARE/ QUEUE q1, q2;
           CLASS c1, c2;
           REF CLASS rc;
           REAL r;

/CONTROL/ TMAX =1000;
          TSTART = 100;
          ESTIM = REGEN;
          PERIOD = 200;
          TEST =
  BEGIN
    SAMPLE;
    ...
    PRINT (CRESPONSE (q1));     & confidence interval on response time
                                & of q1
    ...
    r : =  CRESPONSE (q2, c1);  & confidence interval on response time
                                & of q2 for class c1 customers
    ...
  END;


/EXEC/ BEGIN
         SETSTAT:QUEUE (q1,q2);
         SETSTAT:CLASS (q1,c1);
         SESTAT:ACCURACY (q1,c1);
         SESTAT:ACCURACY (q1,q2);
         ...
         SIMUL;
         PRINT (CRESPONSE (q2));  & confidence interval on response time
                                  & for q2 station
      END;
```

NAME

CSERVICE - Returns the confidence interval values on service time.

SYNTAX

CSERVICE (*queue* [, *class*])

DESCRIPTION

The function returns a real value corresponding to the confidence interval length obtained on the service time mean.

If a class is specified among the arguments, the returned value corresponds to the confidence interval length obtained on the service time mean for the class in the queue.

queue is an expression of QUEUE type.

class is an expression of CLASS type.

EVALUATION

During the execution.

WARNING

- This function can be used only during a simulation resolution.
- The values, used to compute statistical results, are collected between the end of the initialization period (or the beginning of a period if periodic results have been requested) and the last call to SAMPLE function.
- Confidence interval can be obtained for a class of customers only if the mean has been requested by operations such as: /CONTROL/ CLASS = ... or SETSTAT:CLASS.
- The result is available for customers which have left the station.

SEE ALSO

ACCURACY - STATISTICS - CLASS - SETSTAT:SERVICE:ACCURACY - GETSTAT:SERVICE:ACCURACY

EXAMPLE

```
 /DECLARE/ QUEUE q1, q2;
           CLASS c1, c2;
           REF CLASS rc;
           REAL r;

/CONTROL/ TMAX =1000;
          TSTART = 100;
          ESTIM = REGEN;
          PERIOD = 200;
          TEST =
  BEGIN
    SAMPLE;
    ...
    PRINT (CSERVICE (q1));     & confidence interval on service time
                               & of q1
    ...
    r : =  CSERVICE (q2, c1);  & confidence interval on service time
                               & of q2 for class c1 customers
    ...
  END;


/EXEC/ BEGIN
         SETSTAT:QUEUE (q1,q2);
         SETSTAT:CLASS (q1,c1);
         SESTAT:ACCURACY (q1,c1);
         SESTAT:ACCURACY (q1,q2);
         ...
         SIMUL;
         PRINT (CSERVICE (q2));  & confidence interval on service time
                                 & for q2 station
      END;
```

NAME

CUSTNB - Returns the number of customers in the station.

SYNTAX

CUSTNB (*queue* [,*class*])

DESCRIPTION

This function returns an integer value, representing the current number of customers in a station. The station is defined by the first argument of the function.

If a class is specified, the function returns the current number of customers in the specified class.

queue is an expression of QUEUE type.

class is an expression CLASS type.

EVALUATION

During the execution.

WARNING

This function can be used only in simulation resolution.

SEE ALSO

NB

EXAMPLE

```
 /DECLARE/ QUEUE q1, q2;
           CLASS c1, c2;

 /STATION/ NAME =q1;
           SERVCIE =
  BEGIN
    ...
    PRINT (CUSTNB (q1));    & current number of customers in q1
    PRINT (CUSTNB (q1,c1)); & current number of class c1 customers in q1
    ...
  END;


/EXEC/ BEGIN
         SETSTAT:QUEUE (q1,q2);
         SETSTAT:CLASS (q1,c1);
         ...
         SIMUL;
       END;
```

NAME

MAXCUSTNB - Returns the maximum number of customers in a station.

SYNTAX

MAXCUSTNB (*queue* [, *class*])

DESCRIPTION

This function returns an integer value, representing the maximum number of customers in a station. The station is defined by the first argument of the function.

If a class is specified, the function returns the maximum number of customers in the specified class.

queue is an expression of QUEUE type.

class is an expression CLASS type.

EVALUATION

During the execution.

WARNING

- This function can be used only in simulation resolution.
- The values, used to compute statistical results, are collected between the end of the initialization period (or the beginning of a period if periodic results have been requested) and the last call to SAMPLE function.
- This result can be obtained for a class of customers only if results per class have been requested by operations such as: /CONTROL/ CLASS = ... or SETSTAT:CLASS.
- The result is available for customers which have left the station.

SEE ALSO

PMXCUSTNB - CLASS - GETSTAT:CUSTNB:MAXIMUM

EXAMPLE

```
 /DECLARE/ QUEUE q1, q2;
           CLASS c1, c2;

/EXEC/ BEGIN
         SETSTAT:QUEUE (q1,q2);
         SETSTAT:CLASS (q1,c1);
         ...
         SIMUL;
         ...
         PRINT (MAXCUSTNB (q1));    & maximum number of customers in q1
         PRINT (MAXCUSTNB (q1,c1)); & maximum number of class c1
                                    & customers in q1
       END;
```

NAME

MBLOCKED - Returns the mean value of blocking time in the station.

SYNTAX

MBLOCKED (*queue* [, *class*])

DESCRIPTION

The function returns a real value corresponding to the mean value of the blocking time of the station specified by the first argument.

If a class is specified among the arguments, the returned value corresponds to blocking time mean for the class in the queue.

The blocking time is concerned by operations such as: P, WAIT, WAITAND, WAITOR, JOIN, JOINC and BLOCK which can block the service of the customer.

queue is an expression of QUEUE type.

class is an expression of CLASS type.

EVALUATION

During the execution.

WARNING

- This function can be used only during a simulation resolution.
- The values, used to compute statistical results, are collected between the end of the initialization period (or the beginning of a period if periodic results have been requested) and the last call to SAMPLE function.
- This result can be obtained for a class of customers only if results per class have been requested by operations such as: /CONTROL/ CLASS = ... or SETSTAT:CLASS.
- The result is available for customers which have left the station.

SEE ALSO

CLASS - SETSTAT:BLOCKED:MEAN - GETSTAT:BLOCKED:MEAN

EXAMPLE

```
 /DECLARE/ QUEUE q1, q2;
           CLASS c1, c2;
           REAL r;

/CONTROL/ TMAX =1000;
          TSTART = 100;

/EXEC/ BEGIN
         SETSTAT:QUEUE (q1,q2);
         SETSTAT:CLASS (q1,c1);
         ...
         SIMUL;
         PRINT (MBLOCKED (q1));    & blocking time mean in q1
         ...
         r :=  MBLOCKED (q2, c1);  &  blocking time mean in
                                   & q2 for class c1 customers.
      END;
```

NAME

MBUSYPCT - Returns the mean busy percentage of the station servers.

SYNTAX

MBUSYPCT (*queue* [, *class*])

DESCRIPTION

The function returns a real value corresponding to the mean busy percentage of the specified station servers. The busy percentage corresponds to the proportion of time during a server has been requested by customers.

If a class is specified among the arguments, the returned value corresponds to mean busy percentage of the station servers for the specified class customers.

queue is an expression of QUEUE type.

class is an expression of CLASS type.

EVALUATION

During the execution.

WARNING

- This function can be used only during a simulation resolution.
- The values, used to compute statistical results, are collected between the end of the initialization period (or the beginning of a period if periodic results have been requested) and the last call to SAMPLE function.
- This result can be obtained for a class of customers only if the results per class have been requested by operations such as: /CONTROL/ CLASS = ... or SETSTAT:CLASS.
- The result is available for customers which have left the station.

SEE ALSO

CLASS - SETSTAT:BUSYPCT:MEAN - GETSTAT:BUSYPCT:MEAN

EXAMPLE

```
 /DECLARE/ QUEUE q1, q2;
           CLASS c1, c2;
           REAL r;

/CONTROL/ TMAX =1000;
          TSTART = 100;

/EXEC/ BEGIN
         SETSTAT:QUEUE (q1,q2);
         SETSTAT:CLASS (q1,c1);
         ...
         SIMUL;
         PRINT (MBUSYPCT (q1));    & busy percentage in q1
         ...
         r :=  MBUSYPCT (q2, c1);  &  busy percentage in
                                   & q2 for class c1 customers.
      END;
```

NAME

MCUSTNB - Returns the mean number of customers in the station.

SYNTAX

MCUSTNB (*queue* [, *class*])

DESCRIPTION

The function returns a real value corresponding to the mean number of customers in the station specified by the first argument.

If a class is specified among the arguments, the returned value corresponds to the mean number of the class customers in the queue.

The response time represents the elapsed timer between the entry in a queue and the exit from the queue (after a possible service).

queue is an expression of QUEUE type.

class is an expression of CLASS type.

EVALUATION

During the execution.

WARNING

- This function can be used only during a simulation resolution.
- The values, used to compute statistical results, are collected between the end of the initialization period (or the beginning of a period if periodic results have been requested) and the last call to SAMPLE function.
- This result can be obtained for a class of customers only if results per class have been requested by operations such as: /CONTROL/ CLASS = ... or SETSTAT:CLASS.
- The result is available for customers which have left the station.

SEE ALSO

CLASS - SETSTAT:CUSTNB:MEAN - GETSTAT:CUSTNB:MEAN

EXAMPLE

```
 /DECLARE/ QUEUE q1, q2;
           CLASS c1, c2;
           REAL r;

/CONTROL/ TMAX =1000;
          TSTART = 100;

/EXEC/ BEGIN
         SETSTAT:QUEUE (q1,q2);
         SETSTAT:CLASS (q1,c1);
         ...
         SIMUL;
         PRINT (MCUSTNB (q1));    & mean number of customers in q1
         ...
         r :=  MCUSTNB (q2, c1);  & mean number of class c1 customers
                                  & in q2.
      END;
```

NAME

MRESPONSE - Returns the mean value of response time in the station.

SYNTAX

MRESPONSE (*queue* [, *class*])

DESCRIPTION

The function returns a real value corresponding to the mean value of the response time of the station specified by the first argument.

If a class is specified among the arguments, the returned value corresponds to response time mean for the class in the queue.

queue is an expression of QUEUE type.

class is an expression of CLASS type.

EVALUATION

During the execution.

WARNING

- This function can be used only during a simulation resolution.
- The values, used to compute statistical results, are collected between the end of the initialization period (or the beginning of a period if periodic results have been requested) and the last call to SAMPLE function.
- This result can be obtained for a class of customers only if results per class have been requested by operations such as: /CONTROL/ CLASS = ... or SETSTAT:CLASS.
- The result is available for customers which have left the station.

SEE ALSO

CLASS - SETSTAT:RESPONSE:MEAN - GETSTAT:RESPONSE:MEAN

EXAMPLE

```
 /DECLARE/ QUEUE q1, q2;
           CLASS c1, c2;
           REAL r;

/CONTROL/ TMAX =1000;
          TSTART = 100;

/EXEC/ BEGIN
         SETSTAT:QUEUE (q1,q2);
         SETSTAT:CLASS (q1,c1);
         ...
         SIMUL;
         PRINT (MRESPONSE (q1));    & response time mean in q1
         ...
         r :=  MRESPONSE (q2, c1);  & response time mean in
                                    & q2 for class c1 customers.
      END;
```

NAME

MSERVICE - Returns the mean value of service time in the station.

SYNTAX

MSERVICE (*queue* [, *class*])

DESCRIPTION

The function returns a real value corresponding to the mean value of the service time of the station specified by the first argument.

If a class is specified among the arguments, the returned value corresponds to service time mean for the class in the queue.

queue is an expression of QUEUE type.

class is an expression of CLASS type.

EVALUATION

During the execution.

WARNING

- This function can be used only during a simulation resolution.
- The values, used to compute statistical results, are collected between the end of the initialization period (or the beginning of a period if periodic results have been requested) and the last call to SAMPLE function.
- This result can be obtained for a class of customers only if results per class have been requested by operations such as: /CONTROL/ CLASS = ... or SETSTAT:CLASS.
- The result is available for customers which have left the station.

SEE ALSO

CLASS - SETSTAT:SERVICE:MEAN - GETSTAT:SERVICE:MEAN

EXAMPLE

```
 /DECLARE/ QUEUE q1, q2;
           CLASS c1, c2;
           REAL r;

/CONTROL/ TMAX =1000;
          TSTART = 100;

/EXEC/ BEGIN
         SETSTAT:QUEUE (q1,q2);
         SETSTAT:CLASS (q1,c1);
         ...
         SIMUL;
         PRINT (MSERVICE (q1));    & service time mean in q1
         ...
         r :=  MSERVICE (q2, c1);  & service time mean in
                                   & q2 for class c1 customers.
      END;
```

NAME

MTHRUPUT - Returns the mean value of the station throughput.

SYNTAX

MTHRUPUT (*queue* [, *class*])

DESCRIPTION

The function returns a real value corresponding to the mean value of the throughput of the station specified by the first argument.

If a class is specified among the arguments, the returned value corresponds to the throughput of the class in the queue.

queue is an expression of QUEUE type.

class is an expression of CLASS type.

EVALUATION

During the execution.

WARNING

- This function can be used only during a simulation resolution.
- The values, used to compute statistical results, are collected between the end of the initialization period (or the beginning of a period if periodic results have been requested) and the last call to SAMPLE function.
- This result can be obtained for a class of customers only if results per class have been requested by operations such as: /CONTROL/ CLASS = ... or SETSTAT:CLASS.
- The result is available for customers which have left the station.

SEE ALSO

CLASS - SETSTAT:THRUPUT:MEAN - GETSTAT:THRUPUT:MEAN

EXAMPLE

```
 /DECLARE/ QUEUE q1, q2;
           CLASS c1, c2;
           REAL r;

/CONTROL/ TMAX =1000;
          TSTART = 100;

/EXEC/ BEGIN
         SETSTAT:QUEUE (q1,q2);
         SETSTAT:CLASS (q1,c1);
         ...
         SIMUL;
         PRINT (MTHRUPUT (q1));    & q1 throughput
         ...
         r :=  MTHRUPUT (q2, c1);  & throughput of class c1 customers
                                   & in q2.
      END;
```

NAME

OUTPUT - Requests the edition of QNAP2 standard results.

SYNTAX

```
OUTPUT ;
```

DESCRIPTION

This procedure provokes the edition of the resolution standard results.

EVALUATION

During the execution.

NOTE

The standard results are edited even if the command `/CONTROL/ OPTION = NRESULT` has been used.

SEE ALSO

OPTION

EXAMPLE

```
/DECLARE/ QUEUE q1, q2;
          CLASS c1, c2;

/STATION/ NAME = q1;
          SERVICE =
BEGIN
  ...
  OUTPUT;
  ...
END;
             ...
/CONTROL/ TEST = BEGIN
                   ...
                   OUTPUT;
                   ...
                 END;
            PERIOD = 100;
            TMAX = 1000;
            OPTION = NRESULT;

/EXEC/ BEGIN
         SIMUL;   & the standard results are not printed at the end of
                  & the resolution
         OUTPUT;
       END;
```

NAME

PCUSTNB - Access to marginal probabilities of a station.

SYNTAX

PCUSTNB (*integer*, *queue* [, *class*])

DESCRIPTION

The function returns a real value wich represents the probability for the station to contain a number of customers equal to the first argument of the function.

If a class is specified, in thirth argument, the returned value is the probability for the station to contain a number of the specified class customers equal to the first argument of the function.

integer is an expression representing an entity of INTEGER type. *integer* must be positive.

queue is an expression representing an entity of QUEUE type.

class is an expression representing an entity of CLASS type.

EVALUATION

During the execution.

NOTE

Requested marginal probabilities are edited after the standard results.

WARNING

- The computation of marginal probabilities must be requested by using the parameter MARGINAL of the /CONTROL/ command or by the procedure SETSTAT:MARGINAL.
- The values, used to compute statistical results, are collected between the end of the initialization period (or the beginning of a period if periodic results have been requested) and the last call to SAMPLE function.
- This result can be obtained for a class of customers only if results per class have been requested by operations such as: /CONTROL/ CLASS = ... or SETSTAT:CLASS.
- The result is available for customers which have left the station.

SEE ALSO

MARGINAL - CLASS - SETSTAT:CUSTNB:MARGINAL - GETSTAT:CUSTNB:MARGINAL

EXAMPLE

```
 /DECLARE/ QUEUE q1, q2;
           CLASS c1, c2;
           REAL r;

/CONTROL/ MARGINAL= q1,q2;

/EXEC/ BEGIN
         ...
         SOLVE;
         PRINT (PCUSTNB (2,q1)); & Probability for q1 to contain
                                 & 2 customers
         ...
       END;
```

NAME

PMXCUSTNB - Access to the maximum number of customers in a station during the last simulation period.

SYNTAX

PMXCUSTNB (*queue* [, *class*])

DESCRIPTION

This function returns an integer value representing the maximum number of customers in a station, specified by the first argument, during the last simulation period.

If a class is specified, the function returns the maximum number of customers in the specified class, during the last period

queue is an expression of QUEUE type.

class is an expression CLASS type.

EVALUATION

During the execution.

WARNING

- This function can be used only in simulation resolution.
- The values, used to compute statistical results, are collected between the end of the initialization period (or the beginning of a period if periodic results have been requested) and the last call to SAMPLE function.
- This result can be obtained for a class of customers only if results per class have been requested by operations such as: /CONTROL/ CLASS = ... or SETSTAT:CLASS.
- The result is available for customers which have left the station.

SEE ALSO

MAXCUSTNB - CLASS

EXAMPLE

```
 /DECLARE/ QUEUE q1, q2;
           CLASS c1, c2;
           REF CLASS rc;
           REAL r;

/CONTROL/ TMAX =1000;
          TSTART = 100;
          ESTIM = REGEN;
          PERIOD = 200;
          TEST =
  BEGIN
    SAMPLE;
    ...
    PRINT (PMXCUSTNB (q1));      & maximum number of customers in q1
                                 & since the beginning of the last period
    ...
    r : =  (PMXCUSTNB (q2,c1)); & maximum number of class c1 customers in
                                & q2 since the beginning of the last period
    ...
  END;


/EXEC/ BEGIN
         SETSTAT:QUEUE (q1,q2);
         SETSTAT:CLASS (q1,c1);
         SESTAT:PARTIAL (q1,c1);
         SESTAT:PARTIAL (q1,q2);
         ...
         SIMUL;
      END;
```

NAME

SERVNB - Access to the number of customers served during a simulation.

SYNTAX

SERVNB (*queue* [, *class*])

DESCRIPTION

This function returns an integer value representing the number of customers served, in the specified station, during a simulation.

If a class is specified, the function returns the number of customers of the specified class served in the station.

queue is an expression of QUEUE type.

class is an expression CLASS type.

EVALUATION

During the execution.

WARNING

- This function can be used only in simulation resolution.
- The values, used to compute statistical results, are collected between the end of the initialization period (or the beginning of a period if periodic results have been requested) and the last call to SAMPLE function.
- This result can be obtained for a class of customers only if results per class have been requested by operations such as: /CONTROL/ CLASS = ... or SETSTAT:CLASS.
- The result is available for customers which have left the station.

SEE ALSO

CLASS - NBIN - NBOUT - NBSERV

EXAMPLE

```
 /DECLARE/ QUEUE q1, q2;
           CLASS c1, c2;
           REAL r;

/CONTROL/ TMAX =1000;
          TSTART = 100;

/EXEC/ BEGIN
         SETSTAT:QUEUE (q1,q2);
         SETSTAT:CLASS (q1,c1);
         ...
         SIMUL;
         PRINT (SERVNB (q1));    & number of customers served in q1
         ...
         r :=  SERVNB (q2, c1);  & number of class c1 customers served
                                 & in q2
       END;
```

NAME

SONNB - Acces to the number of a customer's sons.

SYNTAX

SONNB (*customer*) ;

DESCRIPTION

This function returns an integer value representing the number of sons of a specified customer.

EVALUATION

During the execution.

WARNING

This function can be used only in simulation, in an algorithmic sequence, defining a service or a test procedure.

SEE ALSO

NEW - REFSON

EXAMPLE

```
/DECLARE / QUEUE q;

/STATION / NAME = q;
           SERVICE = BEGIN
                       ...
                       IF SONNB (CUSTOMER) < 7 THEN
                         TRANSIT (NEW (CUSTOMER), q);
                       ...
                     END;
```

VCUSTNB

NAME

VCUSTNB - Access to the variance of the mean number of customers in a station.

SYNTAX

VCUSTNB (*queue* [, *class*])

DESCRIPTION

The function returns a real value corresponding to the variance of the mean number of customers in the station.

If a class is specified among the arguments, the returned value corresponds to the variance on the mean number of customers of a class in the queue.

queue is an expression of QUEUE type.

class is an expression of CLASS type.

EVALUATION

During the execution.

WARNING

- This function can be used only during a simulation resolution.
- The values, used to compute statistical results, are collected between the end of the initialization period (or the beginning of a period if periodic results have been requested) and the last call to SAMPLE function.
- This result can be obtained for a class of customers only if results per class have been requested by operations such as: /CONTROL/ CLASS = ... or SETSTAT:CLASS.
- The result is available for customers which have left the station.

SEE ALSO

CLASS - SETSTAT:CUSTNB:MEAN - GETSTAT:CUSTNB:VARIANCE

EXAMPLE

```
 /DECLARE/ QUEUE q1, q2;
           CLASS c1, c2;
           REAL r;

/CONTROL/ TMAX =1000;
          TSTART = 100;

/EXEC/ BEGIN
         SETSTAT:QUEUE (q1,q2);
         SETSTAT:CLASS (q1,c1);
         ...
         SIMUL;
         PRINT (VCUSTNB (q1));      & variance on the mean number of
                                    & customers in  q1
         PRINT (VCUSTNB (q2, c1);   & variance on the mean number of
                                    & class c1 customers in q2
         ...
       END;
```

VRESPONSE

NAME

VRESPONSE - Access to the variance on the mean response time of a station.

SYNTAX

VRESPONSE (*queue* [, *class*])

DESCRIPTION

The function returns a real value corresponding to the variance of the mean response time of a station.

If a class is specified among the arguments, the returned value corresponds to the variance on the mean response time of customers of a class in the station.

queue is an expression of QUEUE type.

class is an expression of CLASS type.

EVALUATION

During the execution.

WARNING

- This function can be used only during a simulation resolution.
- The values, used to compute statistical results, are collected between the end of the initialization period (or the beginning of a period if periodic results have been requested) and the last call to SAMPLE function.
- This result can be obtained for a class of customers only if results per class have been requested by operations such as: /CONTROL/ CLASS = ... or SETSTAT:CLASS.
- The result is available for customers which have left the station.

SEE ALSO

CLASS - SETSTAT:RESPONSE:MEAN - GETSTAT:RESPONSE:VARIANCE

EXAMPLE

```
 /DECLARE/ QUEUE q1, q2;
           CLASS c1, c2;
           REAL r;

/CONTROL/ TMAX =1000;
          TSTART = 100;

/EXEC/ BEGIN
         SETSTAT:QUEUE (q1,q2);
         SETSTAT:CLASS (q1,c1);
         ...
         SIMUL;
         PRINT (VRESPONSE (q1));    & variance on mean response time
                                    & for q1
         ...
         r :=  VRESPONSE (q2, c1);  & variance on mean response time
                                    & for class c1 customers in q2.
      END;
```