QNAP2

Addendum V9.2-V9.3

July 1996

Trademarks

Unix is a registered trademark of AT&T.
X Window System is a trademark of the Massachusetts Institute of Technology.
SunView, OpenWindows, NFS are trademarks of Sun Microsystems, Inc.
TEX is a trademark of the American Mathematical Society.
POSTSCRIPT is a registered trademark of Adobe Systems Inc.

Abstract

The QUEUEING NETWORK ANALYSIS PACKAGE, version 9.3 (QNAP2) is a modeling and simulation software developed to facilitate the analysis of large and complex discrete event flow systems such as data communication networks, computer systems, manufacturing facilities, and logistics systems. QNAP2 uses an object oriented representation of model components. It is comprised of a user interface language with facilities for object oriented modeling and a collection of efficient resolution algorithms, including a discrete event simulator with run length control features. The QNAP2 documentation includes the User's Guide and the Reference Manual. The User's Guide presents the features of the QNAP2 language, including the mechanisms it provides to build and analyze models. The Reference Manual describes all the language commands.
This addendum presents the new functionalities in the QNAP2 9.3 version in comparison with the 9.2 version.

Contents

1 Introduction **1**

1.1 New features of QNAP2 V9.1 . . . 3
1.2 New features of QNAP2 V9.2 . . . 3
1.3 New features of QNAP2 V9.3 . . . 3

2 Addendum User's Guide **5**

2.1 Running QNAP2 . . . 7
2.2 Graphics and QNAP2 . . . 9
2.2.1 Procedure GPLOCUR . . . 10
2.2.2 Procedure GPLOBAR . . . 12
2.2.3 Example . . . 14
2.3 QNAP2 error messages . . . 19
2.3.1 Introduction . . . 19
2.3.2 List of messages . . . 20

3 Addendum Reference Manual **41**

3.1 New statistical functions. . . . 43
GETSTAT:SAMPSIZE . . . 44
GETSTAT:SAMPTIME . . . 45
GETSTAT:THRUPUT:MEAN . . . 46
SETSTAT:THRUPUT:MEAN . . . 47
3.2 New traced events. . . . 49
"PMULT" . . . 50
"VMULT" . . . 51
GETTRACE:CLLISTGET . . . 52
GETTRACE:LCLASSNB . . . 54
GETTRACE:LNUMNB . . . 55
GETTRACE:LPRIONB . . . 56
GETTRACE:NUMLISTGET . . . 57
GETTRACE:PRILISTGET . . . 59
3.3 Miscellaneous. . . . 61
HOSTSYS:SHELL . . . 62

Introduction 1

This chapter presents the new features of QNAP2 releases 9.1, 9.2 & 9.3.

1.1 New features of QNAP2 V9.1

These features are presented in the QNAP2 User's Guide and in the QNAP2 Reference Manual.

1. Multiple requests and-or releases of pass grants to semaphores or resource units.
   See `PMULT` and `VMULT` procedures.
2. The `CONVERT` function is now able to convert QNAP2 internal names into `STRING` variables.
   (see the reference manual)
3. The `HOSTSYS:GETENV` function can get the value of environment variables. Error codes are returned by the `HOSTSYS:GETERCOD` function. (see the reference manual)

1.2 New features of QNAP2 V9.2

These features are presented in the QNAP2 User's Guide and in the QNAP2 Reference Manual.

1. The simulation execution profile.
   This feature is available only on UNIX platforms. See the reference manual (`GETPROFILE` and `SETPROFILE`).
   Example :

```
/EXEC/
BEGIN
  SETPROFILE:CLEAR;                    & Clear profiling structures
  SETPROFILE:METERPROC(procedure);     & Ask for results on procedure
  SETPROFILE:STARTMETER;               & Start profiling the execution
  SIMUL;
  SETPROFILE:STOPMETER;                & Stop profiling
  IF GETPROFILE:ISMETERED(procedure)
    PRINT("Consumed time : ",          & Print results
          GETPROFILE:RESULTS:TOTALCPU(procedure));
  SETPROFILE:CLEAR;
END;
```

2. The parallelization of replications.
   See its description in chapter "Solvers" (Estimation of confidence intervals).
3. The SPLITMAT solver.
   See its description in chapter "Solvers" (The Split-Match approximation solver SPLITMAT).

1.3 New features of QNAP2 V9.3

These features are described in this addendum.

1. Running QNAP2 using the command line.

2. Suppression of QNAP2 graphic commands.
   The QNAP2 graphic commands are not supported any longer. See how to emulate them in section "Graphics and QNAP2".
3. Documentation on error messages.
4. New statistical functions.
   See their descriptions in the Reference Manual (`GETSTAT:SAMPSIZE`, `GETSTAT:SAMPTIME`, `GETSTAT:THRUPUT:MEAN`, `SETSTAT:THRUPUT:MEAN`).
5. New traced events.
   See their descriptions in the Reference Manual (`"PMULT"` and `"VMULT"` events).
6. The `HOSTSYS:SHELL` function can issue a shell command.

Addendum User's Guide 2

2.1 Running QNAP2

The user can run the QNAP2 executable in the following way:

```
QNAP2V9 input.qnp [-o output.lis] [-l library.lib]
```

where:

- `input.qnp` is the name of the file containing the model; it is assigned to the QNAP2 predefined file `FSYSINPU`,
- `output.lis` is the name of the file into which the execution results are to be written; it is assigned to the QNAP2 predefined file `FSYSOUTP`; if no name is given, the results are printed on to the screen,
- `library.lib` is the name of the default QNAP2 library file; it is assigned to the QNAP2 predefined file `FSYSLIB`; if no name is given, there is no default library file.

2

2.2 Graphics and QNAP2

Until the V9.2 release, QNAP2 had included a few graphic commands that allowed to draw curves, bar charts and pie charts. Since QNAP2 V9.3, these commands (that are listed below) are no more available.

The graphic commands that have been suppressed are:

- CLEARSCR - clear of graphical window
- MBEGIN - initialization of graphical parameters
- MENDGR - end of graphical operations
- MODIFY - modification of graphical set up
- PLOATT - graphical set up of curves, bar charts and pie charts
- PLOCUR - curve plotting
- PLOHIS - bar chart plotting
- PLOSEC - pie chart plotting

They were suppressed because QNAP2 main purpose is not to perform graphical operations and, above all, because these graphic mechanisms were too rigid.

In the following pages, you will find examples of generation of *GNUPLOT* files for curves and bar charts plotting. *GNUPLOT* is a powerful chart generation and display tool, which is included in the *Modline* product.

These examples are integrated to the QNAP2 examples files.

2.2.1 Procedure GPLOCUR

This procedure is an example of GNUPLOT files generation procedure that can replace the old PLOCUR QNAP2 procedure, for curves plotting.

Procedure code:

```
/DECLARE/
FILE pc_datafile;
FILE pc_gnuplotfile;

PROCEDURE GPLOCUR (filename, nb_curves, nb_points, x, y, title, legend);

& Arguments:
& ----------

STRING filename;                                    & GNUPLOT file name
INTEGER nb_curves,                                  & number of curves
        nb_points;                                  & number of points
REAL x (nb_curves, nb_points),                      & abscissae
     y (nb_curves, nb_points);                      & ordinates
STRING title,                                       & chart title
       legend (nb_curves);                          & legends

& Preconditions (must be checked by the caller):
& -------------
& filename must be a legal file name
& (nb_curves >= 1)
& (nb_points >= 1)

& Local variables:
& ----------------

INTEGER icurve, ipoint;
STRING pref_datafile, ext_datafile, name_datafile;

BEGIN

  pref_datafile := "pc_";                           & data file name prefix
  ext_datafile  := ".dat";                          & data file name extension

  FILASSIGN (pc_gnuplotfile, filename);
  OPEN (pc_gnuplotfile, 3);

  WRITELN (pc_gnuplotfile, "set title '", title, "'");
```

```
  WRITELN (pc_gnuplotfile, "set autoscale");
  WRITELN (pc_gnuplotfile, "plot \");

  FOR icurve := 1 STEP 1 UNTIL nb_curves DO BEGIN

      name_datafile := CONVERT(icurve, STRING) // ext_datafile;
      FILASSIGN (pc_datafile, name_datafile);
      OPEN (pc_datafile, 3);

      FOR ipoint := 1 STEP 1 UNTIL nb_points DO BEGIN
          WRITELN (pc_datafile, x (icurve, ipoint), " ", y (icurve, ipoint));
      END;  & for

      CLOSE (pc_datafile);

      WRITE (pc_gnuplotfile, "'", name_datafile, "'");
      WRITE (pc_gnuplotfile, " title '", legend (icurve), "'");
      WRITE (pc_gnuplotfile, " with linespoints");

      IF (icurve < nb_curves) THEN WRITE (pc_gnuplotfile, ", \");
      WRITELN (pc_gnuplotfile);

  END;  & for

  WRITELN (pc_gnuplotfile, "pause -1 'Press return to continue...'");
  CLOSE (pc_gnuplotfile);

END;  & proc
```

2.2.2 Procedure GPLOBAR

This procedure is an example of GNUPLOT files generation procedure that can replace the old PLOHIS QNAP2 procedure, for bar charts plotting.

Procedure code:

```
/DECLARE/
FILE ph_datafile;
FILE ph_gnuplotfile;

PROCEDURE GPLOBAR (filename, nb_bars, nb_subparts, x, title, legend);

& Arguments:
& ----------

STRING filename;                                    & GNUPLOT file name
INTEGER nb_bars,                                    & number of bars
        nb_subparts;                                & number of bar parts
REAL x (nb_bars, nb_subparts);                      & values
STRING title,                                       & chart title
       legend (nb_bars);                            & legends

& Preconditions (must be checked by the caller):
& -------------
& filename must be a legal file name
& (nb_bars >= 1)
& (nb_subparts >= 1)

& Local variables:
& ----------------

INTEGER ibar, isubpart;
REAL sum;
STRING pref_datafile, ext_datafile, name_datafile;

BEGIN

  pref_datafile := "ph_";                           & data file name prefix
  ext_datafile  := ".dat";                          & data file name extension

  FILASSIGN (ph_gnuplotfile, filename);
  OPEN (ph_gnuplotfile, 3);

  WRITELN (ph_gnuplotfile, "set title '", title, "'");
```

```
  WRITELN (ph_gnuplotfile, "set boxwidth 0.5");
  WRITELN (ph_gnuplotfile, "set autoscale y");
  WRITELN (ph_gnuplotfile, "set xrange [0.5:", nb_bars + 0.5, "]");
  WRITELN (ph_gnuplotfile, "plot \");

  FOR ibar := 1 STEP 1 UNTIL nb_bars DO BEGIN

      name_datafile := pref_datafile //
                       CONVERT(ibar, STRING) // ext_datafile;
      FILASSIGN (ph_datafile, name_datafile);
      OPEN (ph_datafile, 3);

      sum := 0.;
      WRITELN (ph_datafile, ibar, " ", 0.0);

      FOR isubpart := 1 STEP 1 UNTIL nb_subparts DO BEGIN
          sum := sum + x (ibar, isubpart);
          WRITELN (ph_datafile, ibar, " ", sum);
      END;  & for

      CLOSE (ph_datafile);

      WRITE (ph_gnuplotfile, "'", name_datafile, "'");
      WRITE (ph_gnuplotfile, " title '", legend (ibar), "'");
      WRITE (ph_gnuplotfile, " with boxes");

      IF (ibar < nb_bars) THEN WRITE (ph_gnuplotfile, ", \");
      WRITELN (ph_gnuplotfile);

  END;  & for

  WRITELN (ph_gnuplotfile, "pause -1 'Press return to continue...'");
  CLOSE (ph_gnuplotfile);

END;  & proc
```

2.2.3 Example

Here is an example using the two preceding procedures with the corresponding GNUPLOT displays.

Example code:

```
/DECLARE/

INTEGER nb_curve = 4,
        nb_point = 10;

REAL x (nb_curve, nb_point), y (nb_curve, nb_point);

STRING legend (nb_curve);

INTEGER icurve, ipoint;

/EXEC/
BEGIN

   FOR icurve := 1 STEP 1 UNTIL nb_curve DO BEGIN
       FOR ipoint := 1 STEP 1 UNTIL nb_point DO BEGIN
           x (icurve, ipoint) := ipoint + RANDU;
           y (icurve, ipoint) := icurve + RANDU;
       END;
       legend (icurve) := "Stream " // CONVERT (icurve, STRING);
   END;

   GPLOCUR ("curve.gnuplot", nb_curve, nb_point,
            x, y, "Random curves", legend);
   PRINT ("Please run gnuplot on curve.gnuplot to display curves.");

END;  & exec

/DECLARE/

INTEGER nb_bar = 6,
        nb_subpart = 5;

REAL z (nb_bar, nb_subpart);

STRING legend2 (nb_bar);

INTEGER ibar, isubpart;
```

```
REAL barmax;

/EXEC/
BEGIN

   barmax:= nb_subpart * ( 1 + RANDU);
   FOR ibar := 1 STEP 1 UNTIL nb_bar DO BEGIN
       FOR isubpart := 1 STEP 1 UNTIL nb_subpart DO BEGIN
           z (ibar, isubpart) := barmax - (isubpart + RANDU);
       END;
       legend2 (ibar) := "Bar chart " // CONVERT (ibar, STRING);
   END;

   GPLOBAR ("bar.gnuplot", nb_bar, nb_subpart,
            z, "Random bar charts", legend2);
   PRINT ("Please run gnuplot on bar.gnuplot to display bar charts.");

END;  & exec

/END/
```

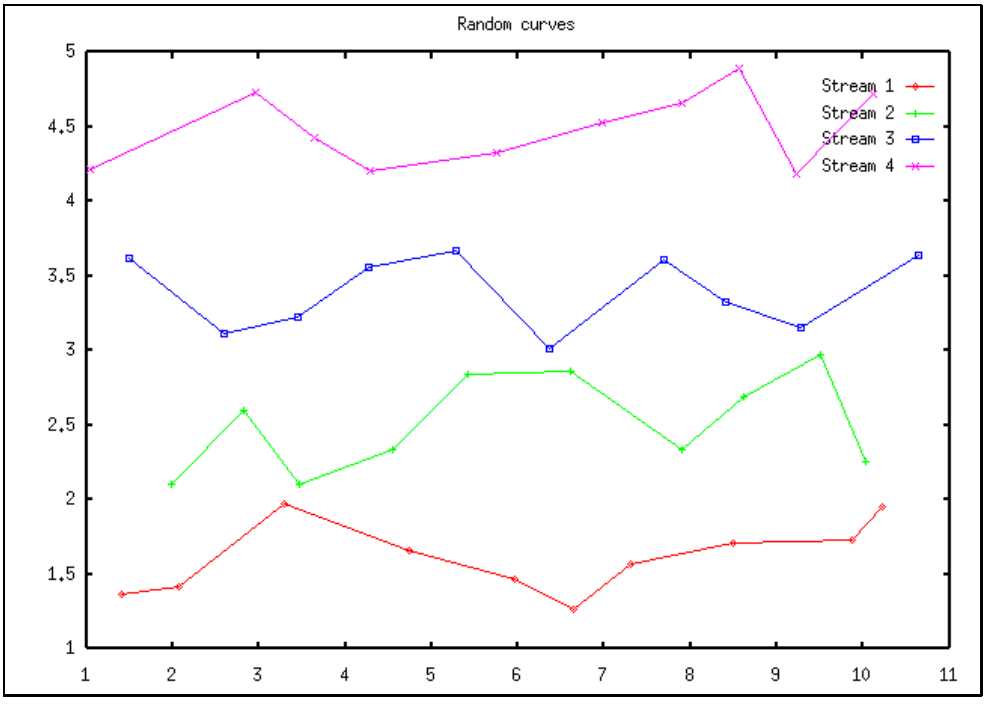

Figure 2.1: GNUPLOT curves plotting

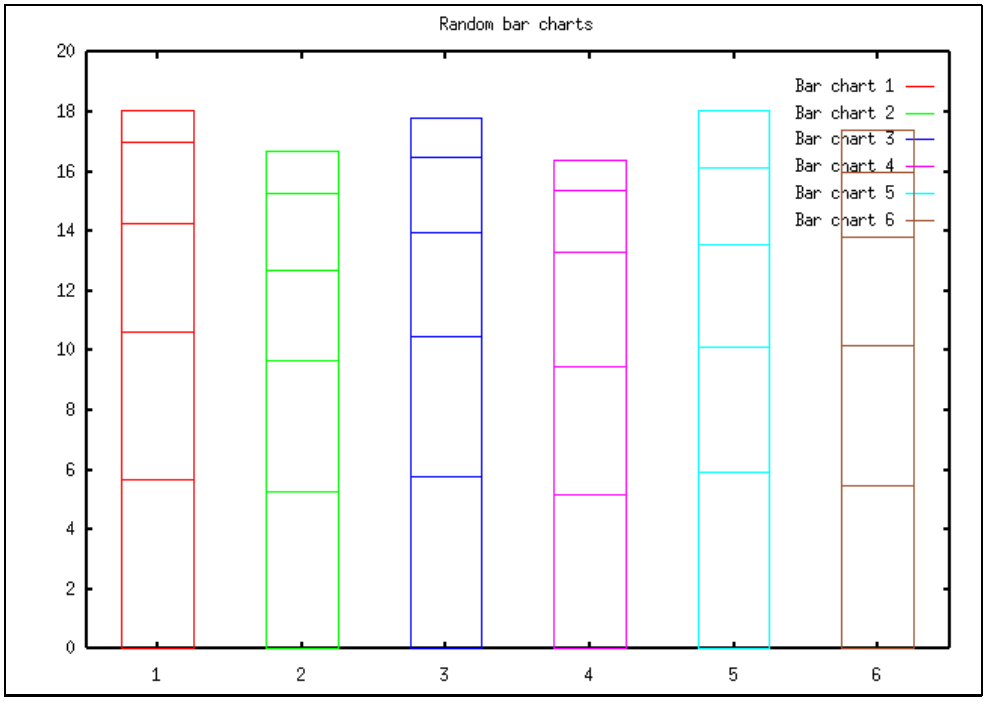

Figure 2.2: GNUPLOT bar charts plotting

2.3 QNAP2 error messages

2.3.1 Introduction

This chapter describes the most important QNAP2 error messages. Each message is described in the following way:

- Message number,
- Message wording,
- Message explanation and action to be taken.

The messages are arranged following the lexical order of their numbers. **This is not an exhaustive list!** If the user ever meets an error message which is not clear enough, he/she is advised to contact SIMULOG (e-mail: *support@simulog.fr*) to get the information he/she needs.

Several message numbers can correspond to the same wording. It is due to the fact that an error message can be called at several places inside QNAP2 and that a message number correspond to one and only one of these calls.

2.3.2 List of messages

- 050101

```
==>ERROR (COMPILE) : INCORRECT SYNTAX
```

The QNAP2 algorithmic language syntax is not respected. For example, IF without THEN or '(' without ')'...
The user has to correct the line where the error occurred according to the algorithmic language syntax.

- 050103

```
==>WARNING (COMPILE) : ";" HAS BEEN ADDED BEFORE THIS ELEMENT
```

QNAP2 adds ";" at the end of the preceding algorithmic statement. To avoid this warning message, the user has to add ";" in the source file.

- 050201

```
==>ERROR (COMPILE) : EXPRESSION IS NOT WELL DEFINED
```

An expression in the algorithmic language is not correct. For example, an assignment operation may be wrong, because there is only one operand.
The user has to check the syntax of the source lines.

- 050202

```
== > ERROR (COMPILE) : INCORRECT MIXED TYPES
```

This message means that a value of a specific type (integer, real, reference to an object) is assigned to a variable of another type. For example, when a WITH operation uses a reference the type of which is different from the type of the listed entities.
The user has to ensure that the types of the concerned entities are the same, possibly by using the operators IS, IN or :: .

- 050203

```
==>ERROR (COMPILE) : INCORRECT TYPE FOR AN OPERAND OR AN ARGUMENT
```

A data used in an operation or as a parameter of a procedure does not belong to the expected type.
The user has to check the syntax of the performed operation and the types of the concerned entities.

- 050206

```
==>ERROR (COMPILE) : THIS IDENTIFIER HAS NOT BEEN DECLARED ...
                     OR IS NOT KNOWN IN THIS CONTEXT ...
```

An *identifier* is the name of a user-defined variable (integer, real, object type, user procedure, ...) or a QNAP2 key-word.
QNAP2 does not recognize the printed identifier in the model source code. The user has to declare the identifier, if it has not been already done, and/or to check its spelling.

- 050207

```
==>ERROR (COMPILE) : THIS VARIABLE CANNOT BE MODIFIED ...
```

It is forbidden to modify the internal variables used by QNAP2, such as the number of customers in a station or the address of an object entity.
The user has to check the validity of the variable modification.

- 050208

```
==>ERROR (COMPILE) : THIS IDENTIFIER CANNOT BE INDEXED ...
```

An indexed notation has got the form: `object.attribute`. The printed identifier corresponds to *object*. This message means that the *object* identifier has been defined as a simple variable, and not as an object instance.
The user has to modify the faulty statement or the *object* declaration.

- 05020F

```
==>ERROR (COMPILE) : "ALL" CANNOT BE USED FOR CUSTOMERS
```

The ALL key-word is used to build a list of entities of the same type. But CUSTOMER entities cannot be listed by this means, to prevent from building too large lists.
To build such a list, the user has to build a list of queues, and for each queue, access to its customers by: `queue.FIRST - customer.NEXT` ...

- 05020Q

```
==>ERROR (COMPILE) : THIS TYPE CANNOT QUALIFY THIS REFERENCE
```

The :: operator checks that a specific entity belongs to the specified type. This message means that the variable preceding the :: does not belong to the type specified after the :: .
The user has to check the model coherence.
Example:

```
/DECLARE/
  REF CUSTOMER @_c;
/EXEC/
BEGIN
  WITH @_c::QUEUE DO PRINT("???");
END;
/END/
```

- 05020U

```
==>ERROR (COMPILE) : CALL TO A GENERIC PROCEDURE FORBIDDEN
```

A generic procedure is different from the other ones because it has no algorithmic code, it only defines a procedure signature (the list of arguments and their types).
Such a procedure cannot be activated because there is no corresponding algorithmic code. It is necessary to use a reference on this procedure and to assign it to a regular procedure (the signature of which is equal to the one defined by the generic procedure).
The user has to see how to use generic procedures, see the key-words: PROCEDURE - GENERIC - REF .

- 050301

```
==>ERROR (COMPILE) : EXPRESSION IS NOT WELL DEFINED
```

An expression in the algorithmic language is not correct. For example, an assignment operation may be wrong, because there is only one operand.
The user has to check the syntax of the source lines.

- 050302

```
050302  ==> ERROR (COMPILE) : INCORRECT MIXED TYPES
```

This message means that a value of a specific type (integer, real, reference to an object) is assigned to a variable of another type. For example, when a WITH operation uses a reference the type of which is different from the type of the listed entities.
The user has to ensure that the types of the concerned entities are the same, possibly by using the operators IS, IN or :: .

- 050303

```
==>ERROR (COMPILE) : INCORRECT TYPE FOR AN OPERAND OR AN ARGUMENT
```

A data used in an operation or as a parameter of a procedure does not belong to the expected type.
The user has to check the syntax of the performed operation and the types of the concerned entities.

- 050304

```
==>ERROR (COMPILE) : INCORRECT NUMBER OF ARGUMENTS
```

The arguments of a procedure are not valid. The error may be due either to the types of the arguments or to their number.
The user has to check the syntax of the concerned procedure and to modify the model according to it.

- 050403

```
==>ERROR (COMPILE) : INCORRECT TYPE FOR AN OPERAND OR AN ARGUMENT
```

A data used in an operation or as a parameter of a procedure does not belong to the expected type.
The user has to check the syntax of the performed operation and the types of the concerned entities.

- 050404

```
==>ERROR (COMPILE) : INCORRECT NUMBER OF ARGUMENTS
```

The arguments of a procedure are not available. The error may be due either to the types of the arguments or to their number.
The user has to check the syntax of the concerned procedure and to modify the model according to it.

- 060101

```
==>ERROR (CONTROL) : INCORRECT SYNTAX
```

The QNAP2 language syntax is not respected in a /CONTROL/ block.
The QNAP2 algorithmic language syntax is not respected. For example, IF without THEN or '(' without ')'...
The user has to correct the line where the error occurred according to the algorithmic language syntax.

- 060103

```
==>ERROR (CONTROL) : CLASS MAX. NUMBER CAN NO LONGER BE EXTENDED
```

The default maximum number of customer classes is 20. This value can be modified by the command: /CONTROL/ NMAX= .... The error message is printed if queues and/or classes have already been defined when the user tries to modify this value.
The user is allowed to modify the maximum number of classes only if neither queues nor classes have been defined before.

- 060105

```
==>WARNING (CONTROL) : ";" HAS BEEN ADDED BEFORE THIS ELEMENT
```

QNAP2 adds ";" at the end of the preceding algorithmic statement. To avoid this warning message, the user has to add ";" in the source file.

- 060107

```
==>ERROR (CONTROL) : BAD FILE OR LOGICAL UNIT SPECIFICATION
```

/CONTROL/ UNIT = *type(file)* command is used to modify the output file. This message means that the new output file must be explicitly defined.
The user has to modify the command or (better) to use procedures (e.g., FILASSIGN) to manage files.

- 060108

```
==>ERROR (CONTROL) : ATTEMPT TO ASSIGN OUTPUT TO A FILE CLOSED
                     OR OPEN IN A WRONG MODE. IGNORED
```

/CONTROL/ UNIT = *type(file)* command is used to modify the input-output files. This message means that the new output file (FSYSOUTP) is closed or opened in "read" mode.
The user has to modify the command or (better) to use procedures (FILASSIGN - OPEN) to manage files.

- 06010C

```
==>ERROR (CONTROL) : ATTEMPT TO ASSIGN INPUT TO A FILE CLOSED
                     OR OPEN IN A WRONG MODE. IGNORED
```

/CONTROL/ UNIT = *type(file)* command is used to modify the input-output files. This message means that the new input file (FSYSINPU, where the modele code is read) is closed or opened in "write" mode.
The user has to modify the command or (better) to use procedures (FILASSIGN - OPEN) to manage files.

- 06010D

```
==>ERROR (CONTROL) : ATTEMPT TO ASSIGN PRINT TO A FILE CLOSED
                     OR OPEN IN A WRONG MODE. IGNORED
```

/CONTROL/ UNIT = *type(file)* command is used to modify the input-output files.
This message means that the new file specified to write the user messages (FSYSPRINT) is closed or opened in "read" mode.
The user has to modify the command or (better) to use procedures (FILASSIGN - OPEN) to manage files.

- 06010E

```
==>ERROR (CONTROL) : ATTEMPT TO ASSIGN GET TO A FILE CLOSED
                     OR OPEN IN A WRONG MODE. IGNORED
```

/CONTROL/ UNIT = *type(file)* command is used to modify the input-output files.
This message means that the new file specified to read data (FSYSGET) is closed or opened in "write" mode.
The user has to modify the command or (better) to use procedures (FILASSIGN - OPEN) to manage files.

- 060201

```
==>ERROR (DECLARE) : THIS IDENTIFIER HAS ALREADY
                     BEEN DECLARED ...
```

An *identifier* is the name of a user-defined variable (integer, real, object type, user procedure, ...) or a QNAP2 key-word.
This message is printed when the user tries to declare an identifier that has already been defined. The user has to check the previous declarations and the identifier spelling.

- 060202

```
==>ERROR (DECLARE) : INCORRECT STRING LENGTH  ...
```

A STRING variable is defined by:
`STRING [ (length) ] id [= string ] ;`
where length is an integer representing the maximum number of characters in the string.
This message is printed if the number of characters declared for the string is less than 0 or greater than 256.
The user has to check the syntax of strings declaration and the integer value given for their length.

- 060302

```
 ==>ERROR (DECLARE)  : INCORRECT SYNTAX
```

The QNAP2 algorithmic language syntax is not respected. For example, IF without THEN or '(' without ')'...
The user has to correct the line where the error occurred according to the algorithmic language syntax.

- 060303

```
 ==>ERROR (DECLARE) : THIS TYPE IS UNKNOWN ...
```

To declare an object instance, the object type identifier is specified before the name of the instance. If an unknown identifier is detected at the begining of a declaration sentence, the current message is printed.
The user has to check the concerned identifier syntax.
Example:

```
      1 /DECLARE/ foo bar;
                    |
 (060303)  ==>ERROR (DECLARE) : THIS TYPE IS UNKNOWN ... foo
      2
      3 /END/
STOP: QNAP2 : END OF EXECUTION
```

- 060306

```
 ==>ERROR (DECLARE) : THIS VARIABLE CANNOT BE INITIALIZED
```

This error appears in a DECLARE block when a variable, declared as an object attribute, or a protected variable, is initialized.
The user has to modify the declaration.
Example:

```
      1 /DECLARE/ OBJECT foo;
      2             INTEGER I=4;
                              |
 (060306)  ==>ERROR (DECLARE) : THIS VARIABLE CANNOT BE INITIALIZED
      3           END;
                    |
 (060302)  ==>ERROR (DECLARE) : INCORRECT SYNTAX
      4
      5 /END/
STOP: QNAP2 : END OF EXECUTION
```

- 060307

```
==>ERROR (DECLARE) : ATTRIBUTES CANNOT BE DECLARED
                     IF OBJECTS OF THE CORRESPONDING TYPE
                     HAVE BEEN CREATED
```

Attributes cannot be added to an object type if objects of this type already exist. The user has to modify the model structure or to create two different types.

- 060309

```
==>WARNING (DECLARE) : ";" HAS BEEN ADDED BEFORE THIS ELEMENT
```

QNAP2 adds ";" at the end of the preceding algorithmic statement. To avoid this warning message, the user has to add ";" in the source file.

- 06030J

```
==>ERROR (DECLARE) : "ANY" IS NOT A LEGAL IDENTIFIER
```

The ANY keyword can be used only after the REF keyword to define a reference to any object type. This message is printed if ANY is used without the REF keyword. The user has to modify the declaration.
Example:

```
     1 /DECLARE/ QUEUE ANY @_ptr;
                         |
(06030J)  ==>ERROR (DECLARE) : "ANY" IS NOT A LEGAL IDENTIFIER
     2
     3 /END/
```

- 06030L

```
==>ERROR (DECLARE) : THIS IDENTIFIER HAS ALREADY
                     BEEN DECLARED ...
```

An *identifier* is the name of a user-defined variable (integer, real, object type, user procedure, ...) or a QNAP2 key-word.
This message is printed when the user tries to declare an identifier that has already been defined. The user has to check the previous declarations and the identifier spelling.

- 060601

```
==>ERROR (STATION) :  INCORRECT SYNTAX
```

The QNAP2 language syntax is not respected in a /STATION/ block.
The QNAP2 algorithmic language syntax is not respected. For example, IF without THEN or '(' without ')'...

The user has to correct the line where the error occurred according to the algorithmic language syntax.

- 060606

```
==>WARNING (STATION) : ";" HAS BEEN ADDED BEFORE THIS ELEMENT
```

QNAP2 adds ";" at the end of the preceding algorithmic statement. To avoid this warning message, the user has to add ";" in the source file.

- 0A0101

```
==>ERROR (EDIT) : INCORRECT SYNTAX
```

The QNAP2 algorithmic language syntax is not respected. For example, IF without THEN or '(' without ')'...
The user has to correct the line where the error occurred according to the algorithmic language syntax.

- 0A0102

```
==>WARNING (EDIT) : END OF FILE DETECTED ON FILE ...
```

This error is generated when the input file has been entirely read and no end of model has been specified.
It is usually due to the fact that the /END/ key-word has been forgotten at the end of the source file. The user only has to add /END/ at the end of the file.

- 0A0105

```
==>ERROR (EDIT) : MEMORY OVERFLOW
```

This message means that the input file cannot be read because QNAP2 memory space is full.
The user can either try to reduce the size of its model (too many queues or too many customers) or use a QNAP2 executable with a larger memory space. To get such a QNAP2 executable, the user can generate it himself or ask SIMULOG for it. In both cases, it is advisable to contact SIMULOG to get more information.

- 0A0201

```
==>ERROR (EDIT) : INCORRECT SYNTAX
```

The QNAP2 algorithmic language syntax is not respected. For example, IF without THEN or '(' without ')'...
The user has to correct the line where the error occurred according to the algorithmic language syntax.

- 0A0205

```
==>ERROR (EDIT) : MEMORY OVERFLOW
```

This message means that the input file cannot be read because QNAP2 memory space is full.
The user can either try to reduce the size of its model (too many queues or too many customers) or use a QNAP2 executable with a larger memory space. To get such a QNAP2 executable, the user can generate it himself or ask SIMULOG for it. In both cases, it is advisable to contact SIMULOG to get more information.

- 0A0501

```
==>ERROR (COMPILE) : THIS IDENTIFIER HAS NOT BEEN
                     DECLARED ...
```

An *identifier* is the name of a user-defined variable (integer, real, object type, user procedure, ...) or a QNAP2 key-word.
QNAP2 does not recognize the printed identifier in the model source code. The user has to declare the identifier, if it has not been already done, and/or to check its spelling.

- 0B0902

```
==>ERROR (INTER) : FILE ...  IS NOT WELL BUILT
```

This message is printed when an input/output operation is performed on a file which has not been assigned.
The user has to assign and open (FILASSIGN and OPEN procedures) the file.
Example:

```
     1 /DECLARE/ FILE f1;
     2           INTEGER I;
     3 /EXEC/ I:=GET (f1,INTEGER);
 (0B0C04)  ==>ERROR (INTER) : FILE ... f1       IS NOT WELL BUILT
 (0I0500)                     LINE NUMBER :      3
     4 /END/
STOP: QNAP2 : END OF EXECUTION
```

- 0B0905

```
==>ERROR (INTER) : CANNOT ASSIGN OPEN FILE ...
```

The procedure FILASSIGN assigns a physical file (the name of which is given as second argument) to the file object given as first argument.
The preceding error message is printed if the QNAP2 file is already opened. The user has to check that the specified file is closed before assigning it.

- 0B0A02

```
 ==>WARNING (INTER) : SYSTEM FILE ...  WILL NOT BE CLOSED
```

The QNAP2 procedure CLOSE is used to close a file. Be careful that the predeclared file FSYSINPU and the implicit files cannot be closed.
To suppress this warning message, the user has to check that he/she is not trying to close an implicit file.

- 0B0A05

```
 ==>ERROR (INTER) : BAD FILE DEFINITION IN CLOSE PROCEDURE
```

This message is printed if a QNAP2 file is closed without having been assigned before.
The user has to assign the file or to ensure that the file has been opened.
Example:

```
      1 /DECLARE/ FILE f1;
      2 /EXEC/ CLOSE (f1);
 (0B0A05)  ==>ERROR (INTER) : BAD FILE DEFINITION IN CLOSE PROCEDURE
 (0I0500)                     LINE NUMBER :      2
      3 /END/
STOP: QNAP2 : END OF EXECUTION
```

- 0B0A07

```
 ==>ERROR (INTER) : UNABLE TO CLOSE FILE ...
```

This message is printed if QNAP2 cannot close a file using the CLOSE procedure.
The reason is given in a following message.
The problem is often due to bad access rights to the considered file for the user.

- 0B0C01

```
 ==>ERROR (INTER): THE DATA WHICH HAS BEEN GIVEN IS NOT CORRECT
                   TRY AGAIN ...
                   ... LINE NUMBER :
```

The QNAP2 procedures GET and GETLN read values of specific types on a file.
The current message means that the data being read do not belong to the expected type. New read operations can be performed depending on the value of the ERRRETRY file attribute.
The user is advised to check that the read data belong to the expected type.

- 0B0C04

```
==>ERROR (INTER) : FILE ...  IS NOT WELL BUILT
```

This message is printed when an input/output operation is performed on a file which has not been assigned.
The user has to assign and open (FILASSIGN and OPEN procedures) the file.

- 0B0C08

```
==>ERROR (INTER) : FORMAT WIDTH EXCEEDS BUFFER SIZE.
                   FILE ...
```

The QNAP2 procedures GET and GETLN read a value on a file with a specific format. The format specifies the number of characters to be read. This number must be lower than the buffer size.
The user has to modify the reading format in order to clear this error message.

- 0B0E05

```
==>ERROR (INTER) : END OF FILE DETECTED ON FILE ...
```

This message means that a reading operation is requested on a file that has been entirely read.
The user has to check the file contents and the reading operations already performed on it.

- 0B0F07

```
==>ERROR (INTER) : UNABLE TO OPEN FILE ...
```

This message is printed when QNAP2 does not succeed in opening a file because of the system.
The user has to check he can access the file.

- 0B0G05

```
==>ERROR (SUPER) : UNABLE TO OPEN FILE FOR RESTORE OPERATION
```

This message is printed when QNAP2 does not succeed in opening a file in which an execution context has been saved.
The user has to check he can access the file.

- 0B0G0A

```
==>ERROR (SUPER) : MODEL NOT FOUND ON RESTORE FILE ...
```

This message means that the restore operation failed for one of the following reasons:

- the string specified for the save operation is different from the string specified for the restore operation,
- an error occurred during the file reading.

The user has to:

- read the file attribute ERRSTATUS to get more information about the error,
- check that the strings specified for the SAVE and RESTORE operations are the same.

- 0B0G0B

```
==>ERROR (SUPER) : SPACE SIZES NOT IDENTICAL :
                              WORDS FOR "SAVE",
                              WORDS "RESTORE"
```

The QNAP2 versions used for the model save and restore are different, the difference is the size of the QNAP2 memory space.
The user has to modify the size of the QNAP2 memory space, so that the QNAP2 executable used to restore the model has the same size as the executable used to save it.

- 0B0G0D

```
==>ERROR (SUPER) : VERSION OF QNAP2 HAD CHANGED BETWEEN
                   SAVE AND RESTORE. YOU MUST SAVE
                   YOUR MODEL AGAIN WITH THE NEW VERSION
```

The QNAP2 versions used for the model save and restore are different.
It is mandatory that the save and restore operation be performed by the same QNAP2 version.

- 0B0H07

```
==>ERROR (SUPER) : ABORT: THE NETWORK IS PROBABLY SATURATED
```

This message occurs when the execution context cannot be saved because the QNAP2 memory space is full.
This message means that the input file cannot be read because QNAP2 memory space is full.
The user can either try to reduce the size of its model (too many queues or too many customers) or use a QNAP2 executable with a larger memory space. To get such a QNAP2 executable, the user can generate it himself or ask SIMULOG for it. In both cases, it is advisable to contact SIMULOG to get more information.

- 0B0I03

```
 ==>ERROR (INTER) : FILE ...  IS NOT WELL BUILT
```

This message is printed when an input/output operation is performed on a file which has not been assigned.
The user has to assign and open (FILASSIGN and OPEN procedures) the file.

- 0G0102

```
 ==>ERROR (EVAL) : NO STATION IN THE NETWORK
```

A network checking occurs when a resolution method (MARKOV - SIMUL- SOLVE) is invoked. But a resolution has no meaning if no station is defined.
The user has to check that queues are created before the model resolution.

- 0G0108

```
 ==>ERROR (EVAL) : TRANSITION TO AN UNDEFINED QUEUE
```

This message means that the customers leaving a queue are sent to an undefined station. This message is printed only if the transition is defined with a reference to a station equal to NIL.
Example:

```
      1 /DECLARE/ QUEUE OBJECT foo;
      2            INTEGER N;
      3           END;
      4           REF foo D;
      5           QUEUE A,B,C;
      6           REAL r1=-1;
      7
      8 /STATION/ NAME=A;
      9           INIT=1;
     10           SERVICE= EXP(1);
     11           TRANSIT=B,0.5,C,r1,D;
 [...]
     23 /EXEC/ BEGIN
     24          r1:=0.1;
     25          SOLVE;
     26        END;
 (0G0108)  ==>ERROR (EVAL) : TRANSITION TO AN UNDEFINED QUEUE
                                 ... STATION : A
     27
     28 /END/
STOP: QNAP2 : END OF EXECUTION
```

- 0G0109

```
 ==>ERROR (EVAL) : TRANSITION TO A QUEUE NOT IN THE NETWORK
```

The NETWORK procedure is used to specify a sub-network, in this case, the resolution will apply only to the stations in the sub-network.
This message means that some customers are sent to a station which does not belong to this sub-network.
The user can remove the NETWORK procedure from the model, extend the list of queues in the sub-network or modify the customer transitions so that they stay in the sub-network.

- 0G010B

```
 ==>ERROR (EVAL) : TRANSITION TO AN UNDEFINED CLASS
```

This message means that the class of customers leaving a queue is changed to an undefined class.
This message is printed only if the class is defined with a reference equal to NIL.
Example:

```
      1 /DECLARE/
      2           REF CLASS @_C;
      3           CLASS C1,C2;
      4           QUEUE A,B,C;
      5           REAL r1=-1;
      6
      7 /STATION/ NAME=A;
      8           INIT=1;
      9           SERVICE= EXP(1);
     10           TRANSIT=B,0.5,C,@_C,r1;
 [...]
     22 /EXEC/ BEGIN
     23          r1:=0.1;
     24          SOLVE;
     25        END;
 (0G010B)  ==>ERROR (EVAL) : TRANSITION TO AN UNDEFINED CLASS
                                 ... STATION : A        CLASS :
     26
     27 /END/
```

- 0G010K

```
 ==>ERROR (EVAL) : A SOURCE, A RESOURCE OR A SEMAPHORE
                   CANNOT BE INITIALIZED WITH CUSTOMERS
```

The INIT parameter, in a /STATION/ block, specifies the number of customers in a queue before the beginning of the resolution.

Source, resource or semaphore stations take specific roles and cannot contain customers before the resolution begins. Such stations are defined by the following commands:

- TYPE = SOURCE;
- TYPE = RESOURCE;
- TYPE = SEMAPHORE;

The user has to suppress the INIT command or modify the station type.

- 0I0507

```
==>ERROR (INTER) : INDEX OUT OF BOUNDS. VALUE :
```

This message means that the index computed by QNAP2 is out of the array bounds. The user has to compare the index value with the bounds declared for the array.

- 0I050B

```
==>WARNING (INTER) : ATTEMPT TO DIVIDE BY ZERO,
                     MAXIMUM VALUE ASSUMED
```

A division by zero has been detected by QNAP2 in the model algorithmic code. The result of the operation is then equal to the maximum real or integer value the machine can manage.
It is highly advisable not to allow such operations.

- 0I050H

```
==>ERROR (INTER) : INCORRECT CONTROL VALUES IN A "STEP/UNTIL"
                   CLAUSE, THE SECOND BOUND CANNOT BE REACHED
```

This message is printed if the higher bound of a STEP UNTIL command is lower than the lower bound.
It is mandatory to modify the values of the bounds.

- 0I051D

```
==>ERROR (INTER) : OBJECT REFERENCED HERE DOES NOT BELONG
                   TO THE TYPE ...  EXPECTED FOR THIS
                   REFERENCE OR IS ALREADY DELETED ...
```

This message means that a reference refers to an object which does not belong to the specified type, or which has been deleted.
The user has to check the reference type using the IS or IN operators.
Example:

```
     1 /DECLARE/ QUEUE A,B;
     2           CUSTOMER OBJECT person;
     3             REAL d_entry;
     4           END;
     5           REF person @person;
     6           REF ANY rc;
     7
 [...]
    17 /STATION/NAME=A;
    18          TRANSIT=OUT;
    19          SERVICE=BEGIN
    20                    CST(2);
    21                    rc:=QUEUE;
    22                    WITH rc::person DO
    23                      PRINT("d_entry: ",d_entry);
    24                    END;
 [...]
 *** SIMULATION ***
 (0I0508)  ==>ERROR (INTER) : OBJECT REFERENCED HERE DOES NOT BELONG
                              TO THE TYPE ... person   EXPECTED FOR THIS
                              REFERENCE OR IS ALREADY DELETED ...
 (0I0500)                     LINE NUMBER :     22
 ... ACTIVE STATION : A        FOR CUSTOMER :      2(        )
 ... TIME =      2.000
                    LINE NUMBER :     22
    30
    31 /END/
STOP: QNAP2 : END OF EXECUTION
```

- 0I0603

```
 ==>WARNING (INTER) : RESULT NOT AVAILABLE ... ZERO ASSUMED
```

This message is printed if a non computed statistical result is requested by the user. The result is not computed because the sample is empty or because its computation has not been requested.
The user has to ask for the result computation using the appropriate SETSTAT: procedure and not to request non computed results.

- 0I0701

```
 ==>ERROR (INTER) : INVALID VALUE SPECIFIED FOR TMAX
```

This message is printed if the SETTMAX procedure is used with an invalid parameter: lower than zero or lower than the current date.
The user has to check the value of the parameter of the SETTMAX procedure.

- 0I0702

```
==>ERROR (INTER) : MEMORY OVERFLOW
```

This message means that the input file cannot be read because QNAP2 memory space is full.
The user can either try to reduce the size of its model (too many queues or too many customers) or use a QNAP2 executable with a larger memory space. To get such a QNAP2 executable, the user can generate it himself or ask SIMULOG for it. In both cases, it is advisable to contact SIMULOG to get more information.

- 0I0B01

```
==>ERROR (INTER) : A REFERENCE WITH VALUE "NIL" IS USED
```

This message is printed if we use a reference that refers to an object equal to NIL.
The user has to modify the reference value, so that it refers to an existing object.

- 0I0B02

```
==>ERROR (INTER) : A REFERENCE TO A DESTROYED OBJECT IS USED
```

This message means that a reference to a deleted object is used.
The user has to modify the reference value, so that it refers to an existing object. He/she can check the referenced object destruction using the DELETED function.

- 0J0501

```
==>ERROR (SUPER) : MEMORY OVERFLOW ...
```

This message means that the input file cannot be read because QNAP2 memory space is full.
The user can either try to reduce the size of its model (too many queues or too many customers) or use a QNAP2 executable with a larger memory space. To get such a QNAP2 executable, the user can generate it himself or ask SIMULOG for it. In both cases, it is advisable to contact SIMULOG to get more information.

- 0J0602

```
==>ERROR (FREELM) : DOUBLE FREE ON THE SAME AREA
                    BUG IN QNAP2
```

The same QNAP2 memory area has been freed twice. It is a QNAP2 internal error.
The user has to contact SIMULOG (e-mail: *support@simulog.fr*, tel: +33-(1)-30-12-27-77).

- 0J0901

```
==>ERROR (SUPER) : MEMORY OVERFLOW
                   ... (GETLIM)
                   ... ALREADY ALLOCATED :  WORDS
                   ... STILL AVAILABLE   :  WORDS
                   ... REQUESTED         :  WORDS
```

This message means that the input file cannot be read because QNAP2 memory space is full.
The user can either try to reduce the size of its model (too many queues or too many customers) or use a QNAP2 executable with a larger memory space. To get such a QNAP2 executable, the user can generate it himself or ask SIMULOG for it. In both cases, it is advisable to contact SIMULOG to get more information.

- 0J0H02

```
==>ERROR (SUPER) : INCORRECT SYNTAX
```

The QNAP2 algorithmic language syntax is not respected. For example, IF without THEN or '(' without ')'...
The user has to correct the line where the error occurred according to the algorithmic language syntax.

- 0J0H08

```
==>WARNING (SUPER) : ";" HAS BEEN ADDED BEFORE THIS ELEMENT
```

QNAP2 adds ";" at the end of the preceding algorithmic statement. To avoid this warning message, the user has to add ";" in the source file.

- 0J0H09

```
==>WARNING (SUPER) : SOME STATEMENTS ARE SKIPPED
                     UNTIL SOME COMMAND OCCURS
```

This message is printed after every message that occurs during the analysis of a command content. This message warns the user that the analysis of the model goes to the next command block.
The user has to correct the error to get a complete analysis of the model.

- 0J0H0A

```
==>ERROR (SUPER) : CANNOT LAUNCH EXECUTION
                   SOME FATAL ERRORS OCCURED
```

This message is printed if the compilation of the algorithmic code of the /EXEC/ command has failed.
The user has to correct the compilation errors.

- 0R0101

```
==>WARNING (SIMUL) : ARITHMETIC UNDERFLOW ON TIME MANAGEMENT
                     A SERVICE DELAY IS TOO SMALL
                     COMPARED WITH "TIME"
```

The QNAP2 real variables are in simple precision. Internally, the simulation time is managed in double precision. So, if you have a long simulation time with small delays, there may be losses of precision on the user-managed variables.
The preceding message means that the user will have to take care of the validity of the manually computed time-dependent results.
The user is advised to use the standard results computed by QNAP2, and not to build simulation models with two different time scales (because of the validity of the user-computed results and to avoid too long simulations).

- 0R0408

```
==>ERROR (SIMUL) : NEGATIVE VALUE ASSIGNED
                   TO A DELAY :
```

A CUSTOMER or a TIMER is asked to wait a negative time delay, which is forbidden.
The user has to ensure that this delay be always positive.

- 0R0409

```
==>ERROR (SIMUL) : UNDEFINED TRANSITION
```

No transition has been defined for the CUSTOMER in the specified QUEUE.
The user has to check the TRANSIT parameter of the queue.

- 0R0A01

```
==>WARNING (SIMUL) : NO TMAX SPECIFIED
                     TMAX=0. ASSUMED
```

The TMAX parameter has not been specified by the user at the beginning of the simulation. QNAP2 will assume that TMAX is null.
The user has either to change the TMAX value at the simulation beginning (with the SETTMAX procedure) or to add a /CONTROL/ TMAX=...; instruction before the simulation launching.

- 0R0M03

```
==>ERROR (SIMUL) : MEMORY OVERFLOW
```

This message means that the input file cannot be read because QNAP2 memory space is full.
The user can either try to reduce the size of its model (too many queues or too many customers) or use a QNAP2 executable with a larger memory space. To get such a QNAP2 executable, the user can generate it himself or ask SIMULOG for it. In both cases, it is advisable to contact SIMULOG to get more information.

- 130L02

```
==>ERROR (STATIS) : NO STATISTIC STRUCTURE CONNECTED
                    TO THE VARIABLE
```

The user tries to obtain a result (using a GETSTAT function) on a variable to which no statistic structure is connected.
The user has to check that the GETSTAT function arguments are queues or watched variables for which the specified result has been requested.

- 130L03

```
==>ERROR (STATIS) : UNAVAILABLE RESULT BECAUSE MISSING USER
                    SPECIFICATION
```

The user tries to obtain a result (using a GETSTAT function) the calculation of which has not been requested.
The user has to check that the GETSTAT function arguments are queues or watched variables for which the specified result has been requested.

- 130M0C

```
==>ERROR (STATIS) : NO STATISTIC ON THE STATION
```

The user tries to obtain a queue result (using a GETSTAT function) the calculation of which has not been requested.
The user has to check that the SETSTAT request corresponding to this result has been performed.

- 130M0J

```
==>ERROR (STATIS) : OPERATION FORBIDDEN DURING SIMULATION
```

Most of the SETSTAT procedures can not be called during the simulation.
The user has to perform the results requests before the simulation starts.

Addendum Reference Manual 3

3.1 New statistical functions.

GETSTAT:SAMPSIZE	Computes and returns the total number of measures of a discrete sample.
GETSTAT:SAMPTIME	Computes and returns the total sampling time on a continuous sample.
GETSTAT:THRUPUT:MEAN	Returns the mean throughput of a queue.
SETSTAT:THRUPUT:MEAN	Statistical results request on the throughput of a queue.

GETSTAT:SAMPSIZE

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;
```

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;
```

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

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;
```

3.2 New traced events.

"PMULT"	Event associated to the `PMULT` procedure.
"VMULT"	Event associated to the `VMULT` procedure.
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:LCLASSNB	Returns the number of classes of the requests specified for the current traced operation when this number is greater than one.
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.
GETTRACE:LPRIONB	Returns the number of the priorities of the requests specified for the current traced operation when this number is greater than one.
GETTRACE:NUMLISTGET	Returns one by one the values of the numbers specified for the current traced operation.
GETTRACE:PRILISTGET	Returns one by one the values of the priorities of the requests specified for the current traced operation.

"PMULT"

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

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

GETTRACE:CLLISTGET

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;
```

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: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: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;
```

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;
```

3.3 Miscellaneous.

HOSTSYS:SHELL	Issue a shell command.

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
```