

EcosimPro

Modelling and Simulation Software

Version 4.2 Upgrade

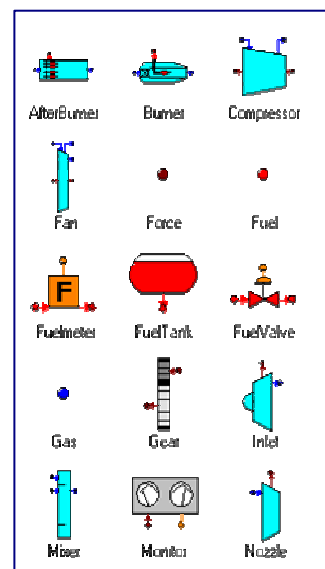
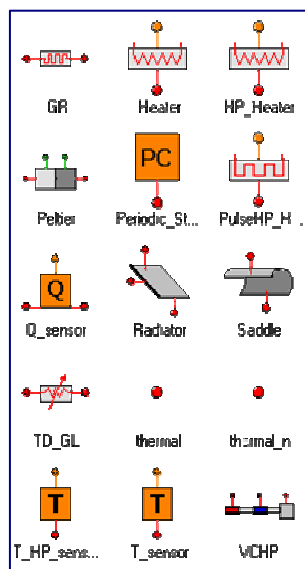
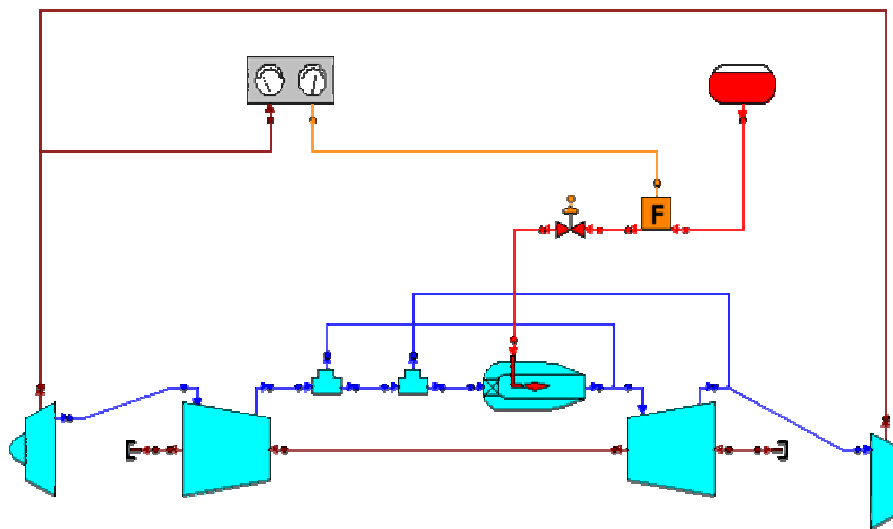


TABLE OF CONTENTS

	PAGE
1. INTRODUCTION	2
1.1 HOW TO INSTALL VERSION 4.2.....	2
2. STOP FUNCTION	3
3. FILEPATH	5
4. NEW ASSERTIONS	7
4.1 ASSERT STATEMENT IN CONTINUOUS BLOCK	7
5. ISSUEERROR() FUNCTION	9
6. INIT BLOCKS WITH PRIORITIES.....	10
7. UNITS.....	13
8. LIBRARY VERSIONING.....	15
9. USING LIBRARIES IN EL.....	16
9.1 LIBRARY STATEMENT	17
10. FUNCTION FOR OBTAINING UNITS	19
11. FUNCTION FOR OBTAINING RELEASE REPORT.....	21
12. FUNCTION FOR OBTAINING AN INTERNAL RESIDUE	23
13. UPGRADE MONITOR	25
13.1 GENERAL.....	25
13.2 MONITOR TOOLBAR	25
13.3 MONITOR WORKSPACE	27
13.4 PLOT PROPERTIES.....	28
13.5 TABLE VISUALIZATION	28
14. CALCULATION WIZARD	35
14.1 PARAMETRIC CALCULATION.....	36
14.2 STEADY CALCULATION.....	41



1. INTRODUCTION

This document outlines the main differences and improvements between versions 4.0 and 4.2 of EcosimPro.

1.1 HOW TO INSTALL VERSION 4.2

The following steps are recommended to install EcosimPro 4.2:

1. Close all previous versions of EcosimPro and exit the program
2. Make a backup copy of your libraries
3. Uninstall your current version of EcosimPro from the EcosimPro menu
4. Install the new version following the standard procedure in the Installation Manual
5. Copy your libraries to a new location.
6. Run EcosimPro.exe
7. If you have any problems, contact the EcosimPro software support group at support-sw@ecosimpro.com

2. STOP FUNCTION

The STOP statement can be used to halt a simulation at any time. The syntax is as follows:

```
STOP "The ball reaches the ground"
```

The following example shows the behaviour of a rubber ball falling from a certain height and the simulation halts when the height of the bounce becomes less than a certain value:

```
COMPONENT bouncingBall
  DATA
    REAL g = 9.80665           "Gravity (m/s^2)"
    REAL k = 0.8               "Restitution coefficient (-)"
  DECLS
    REAL h = 10.              "Height (m)"
  DISCRETE
    -- Event when bouncing on the ground
    WHEN (h < 0) THEN
      h' = -k * h'
    END WHEN
    -- Event to detect stopping the simulation
    WHEN (h < -1e-6) THEN
      STOP "***** End of simulation: height < -1e-6"
    END WHEN
  CONTINUOUS
    h'' = -g
END COMPONENT
```

In the above example, when the variable 'h' becomes negative the simulation halts and the following message is shown on screen:

```
--- STOP SIMULATION ---
***** End of simulation: height < -0.003
```


3. FILEPATH

The FILEPATH data type is identical to STRING and allows the same operation to be carried out. The only difference is that if a type is defined as FILEPATH, when the object editor is used it allows the user to select a file path (relative or absolute) of a file in the system using an adequate browser.

The FILEPATH type will normally be used to represent the path of a file. An absolute path, such as the following, can be represented:

```
FILEPATH myFile ="C:/my programmes/files/file3455.txt"
```

The relative path of a library can also be represented with the form @LIB_NAME@. For example:

```
FILEPATH myFile ="@ELECTRICAL@/data/file45.txt"
```

There are three very useful functions for dealing with file paths:

```
STRING expandFilePath(IN FILEPATH path)
```

This function, given a relative path to a library, will return an absolute one replacing the relative path by the absolute one. For example, assuming that the ELECTRICAL library is located at D:/myLibs/ELECTRICAL, and the following is typed:

```
finalPath= expandFilePath( myFile )
```

the variable finalPath will contain the absolute path "D:/myLibs/ELECTRICAL/data/file45.txt"

```
BOOLEAN existsFile(IN FILEPATH filePath)
```

This function returns TRUE if the file passed to it as an argument exists, and FALSE if it does not exist.

For example:

```
INIT
if ( existsFile( expandFilePath( myFile ) ) == TRUE )
    -- the file has been found
```

```
FILEPATH checkFileExists(IN FILEPATH filePath)
```

This function is a combination of the two previous ones. The user passes a relative or absolute path to it, the function expands the path if it is a relative one, and check if the file exists. If it does not exist, it emits an error message which halts the application, and if it does exist, it returns the complete path.

This function is very useful and can be used when an external function has to be called that has to read an external file, and the user wants to ensure that the file exists. If it does not exist an error message will be emitted within the program as opposed managing the error externally which is always more complicated and out of control.

For example, given an external function in C++ with a file name as an argument, it is possible to pass it the absolute path of a file after having checked that it exists:



```
"C++" FUNCTION NO_TYPE readFile (IN FILEPATH file)

COMPONENT test
DECLS
    FILEPATH fpath= "%ELECTRICAL%/data/file45.txt"
INIT
    readFile ( checkFileExists( fpath ) )
END COMPONENT
```

4. NEW ASSERTIONS

4.1 ASSERT STATEMENT IN CONTINUOUS BLOCK

EL has a powerful method for checking integrity based on assertions or confirmations. The user may include ASSERT statements in the sequential instructions block, to ascertain whether a condition is met or not. The syntax is as follows:

```
ASSERT expression (NOTE | WARNING | ERROR | KILLPOINT | FATAL) message
```

When the expression evaluates to FALSE, the assertion is activated and a message, which you specify, is displayed. Assertions are a way of controlling the consistency of the model by periodically checking on the values of fundamental variables. There are different types of assertions, which give rise to different actions:

NOTE. Simple emission of a message to the user. The simulation continues.

WARNING. Something unexpected has occurred, but it is not serious. The simulation continues.

ERROR. A serious assertion. The simulation continues

KILLPOINT. A serious assertion. The simulation kills the current calculation but it continues with the following ones.

FATAL. A serious assertion which does not allow the simulation to continue

Example:

```
tp_out.radiation= evalRadiation(tp)
ASSERT (tp_out.radiation < maxRad 0.) FATAL "DANGER, RADIATION!"
```

In this example, the variable tp_out.radiation has been calculated as well as its integrity, which must be lower than a limit. If the condition is FALSE the specified message is emitted and the simulation halts.

5. ISSUEERROR() FUNCTION

The `issueError()` function is used to generate error messages from the EL code. Its syntax is as follows:

```
NO_TYPE issueError(IN INTEGER errorId, ...)
```

`errorId` is the error number to be displayed. The three dots mean that it can take a variable number of arguments. Normally these arguments are used to pass the values of variables to be printed in the error message (in a similar way to the `WRITE()` function)

The user can create a file with error codes for each library in the following way:

- Create a text file called `LIBNAME.errors.txt` and place it just below the directory of the library. For example, if the library is `ELECTRICAL`, the file will be called `ELECTRICAL.errors.txt` and will be in the `ELECTRICAL` directory.
- The file format must include a header with the field names separated by tabs. Below, in each row is each error code. For example:

```
CODESEV  CAT SOURCE  MESSAGE
255 9     24  98   The beta(BETA=%g) is too high
256 99    25  98   The compressor is out out limits (%g,%g)
```

where:

- `CODE` is the number of the error code. Codes 212, 213, 214, 215 and 216 are reserved for the different types of assertions, and so should not be used.
- `SEV` is the severity, as follows:

```
Note:      0-6
Warning:   7
Error:     8
Killpoint: 9
Fatal:    99
```

- `CAT` is the category, which can be any positive number
- `SOURCE` is the source of the error, represented by a positive integer. Usually a different number is used for each library, so that the source of the error can be identified rapidly.
- `MESSAGE` is the message to be displayed in the event of error and is free format. If an argument is required one of the following may be used:

```
%d integer
%g real
%s string
```

In this way when the function is called arguments can be passed to it. Similar to the `EL WRITE()` statement.

For example, to create a new error, with code 312, type killpoint, category 22 and source 98. The message has to inform that a value is outside the limits, and has to print both the value and the limit, which are passed to it as arguments.

To do this, make a new entry in the error file, as follows:

```
CODESEV  CAT SOURCE  MESSAGE
312 9     22  98   The variable %s with value %g is out of limits(%g,%g)
```

If the function is then called:



```
issueError(312,"speed",4.54, 5.5, 10.5)
```

and it is executed, the following message is printed:

```
*** KILLPOINT level 2 (code 9:312:22:98) path: .) in file "exp1.exp" at line  
14 ***
```

```
The variable speed with value 4.54 is out of limits(5.5,10.5)
```

The current calculation will halt and the program will continue with the next one.

6. INIT BLOCKS WITH PRIORITIES

Priorities can be assigned to INIT blocks. By default, the priority of an INIT block is zero. To assign a priority the user can add the optional clause PRIORITY after the word INIT. The following statements are valid:

```
INIT PRIORITY 100      The INIT block has a priority of 100
```

```
INIT PRIORITY -40     The INIT block has a priority of -40
```

```
INIT                  The INIT block has a priority of 0
```

When the final INIT blocks of all the components are sorted, the following sorting rules by priority are followed:

- First, they are ordered by priority.
- If the priorities are equal, the INITs of parent components go first, then those of the final components and lastly those of the aggregated components.

For example, given the following example:

```
COMPONENT Parent
```

```
DECLS
```

```
  REAL x1
```

```
INIT
```

```
  x1= 1
```

```
END COMPONENT
```

```
COMPONENT Compo1
```

```
DECLS
```

```
  REAL x2
```

```
INIT
```

```
  x2= 2
```

```
END COMPONENT
```

```
COMPONENT Compo2
```

```
DECLS
```

```
  REAL x3
```

```
INIT PRIORITY 100
```

```
  x3= 3
```

```
END COMPONENT
```

```
COMPONENT Final IS_A Parent
```

```
DECLS
```

```
  REAL x4
```

```
TOPOLOGY
  Compo1 obj1
  Compo2 obj2
INIT
  x4= 4
END COMPONENT
```

When a partition is made and the INIT blocks are sorted, it will be done as follows:

```
--init(obj2.Compo2,100)
obj2.x3 = 3
-- init(Parent,0)
x1 = 1
-- init(Final,0)
x4 = 4
-- init(obj1.Compo1,0)
obj1.x2 = 2
```

As can be seen, the first one is from Compo2, as it has the highest priority (100). The rest have been sorted following the above rules: first the parent component, then the final component, and lastly the aggregate component with the same priority (all with default priority 0).

7. UNITS

When a variable is declared the user may specify the units worked with by using the reserved word UNITS followed by a string. For example:

```
REAL speed          UNITS "m/s"  
REAL acceleration UNITS "m/s**2"
```

It is important to note that the string representing the units is taken literally as a comment so it is the modeller's responsibility to ensure that it is correct.

These units are used later by different editors and in reports, etc and provide valuable information to users.

It is possible to create STRING type constants and then use them as units. For example:

```
CONST STRING u_speed= "m/s"  
CONST STRING u_accel= "m/s**2"  
  
REAL speed          UNITS u_speed  
REAL acceleration UNITS u_accel
```

In this way the constant can be used each time the units have to be written.

8. LIBRARY VERSIONING

The libraries have a mechanism whereby it is possible to tag them with a version number. It is a special structure called #LIBRARY_INFO, and consists of creating an EL file for each library as follows:

```
#LIBRARY_INFO
  #VERSION = "1.0"
  #DATE = "12/02/2009"
  #AUTHOR= "None"
  #COMMENTS= "no comments"
END #LIBRARY_INFO
```

The following can then be indicated within the structure:

- Current version of the library with the format "MAV.MIV(.UPV)", where MAV (Major Version), MIV (Minor Version) and UPV (Upgrade Version) are positive integers. The major and minor versions are mandatory in any case. For example, the following are valid version numbers: : "1.2", "3.4.55", "4456.22" and "0.0" whereas the following are not valid: "-3.1", "45", "A.1" and "1.1.5.3"
- Date of creation. A string in free format
- Author. A string in free format
- Comments. A string in free format. If more than one line is required, the symbol "\" has to be placed at the end of each line. For example:

```
#LIBRARY_INFO
  #VERSION = "1.0"
  #DATE = "12/02/2009"
  #AUTHOR= "None"
  #COMMENTS= "This is the first \
              Version of the Programme, \
              Ready to use by external users"
END #LIBRARY_INFO
```

This information will be used at different points within the program.

- Library documentation. Each time the generation of the library documentation is requested, this information will be printed.
- Partition documentation, reports, etc. Information is given on the library version when they are generated.
- For conditional compilations with the USE statement (see next point)



9. USING LIBRARIES IN EL

By default the tool will compile all items of a file into the library under which the file is located. In other words, if the user has a file named myFile.el under ELECTRIC\sources directory, the components in the file will be added to the ELECTRIC library.

If they are items from other libraries, the user must pre-declare their usage with the statement USE such as:

```
USE CONTROL
COMPONENT foo
TOPOLOGY
    Cntrl_PID pid    -- from CONTROL library
END COMPONENT
```

The USE clause advises that the library CONTROL must be loaded because some code will make reference to components (or other items) from those libraries. In this case it uses the CONTROL.Cntrl_PID component.

If the name of the component is to be repeated both in the CONTROL and the current library, the user must write LIB_NAME.ITEM_NAME to define the exact item to be used. For example:

```
USE CONTROL
COMPONENT foo
TOPOLOGY
    CONTROL.Cntrl_PID pid    -- from CONTROL library
END COMPONENT
```

The USE statement allows to specify a required library version. For example:

```
USE CONTROL VERSION "3.4"
```

In this case, when the file is compiled, the program will check if the current CONTROL library is version "3.4" or compatible. If it is not, an error code will be generated and compilation halted.

Compatibility between versions is defined as follows:

- if the major version and the minor version of the current CONTROL library fits with the required one ("3.4" in the previous example).
- if the major version of the current CONTROL library fits with the required one in the USE (3) and the minor version of the current one is greater than the minor version of the required (4)
- any other combination is not compatible and an error is issued.

For example, for the previous USE it would be compatible if the version of the CONTROL library is "3.4", "3.4.1", "3.4.32", "3.5", "3.12.34", etc. However, it would not be compatible if it were "2.1", "3.3", "3.3.4", "4.0", "12.34", etc.

In the event of incompatibility it is the responsibility of the modeller to check if the components are still compatible and can update the version in the USE statement.

9.1 LIBRARY STATEMENT

In some cases the user can force the library to be used as a depository for the items instead of the current library. This is done by means of the LIBRARY statement. For instance, suppose we have a file named myFile.el under the sources directory of the CONTROL library, if the user writes:

```
LIBRARY ELECTRIC
CONST REAL PI = 3.141592
```

the item PI will not be added to the CONTROL library (as expected) but to the ELECTRIC library. In general, this practice is not recommended because the programming will lose clarity since some files of a library can modify the database of a different library. This only has real application when programming outside the standard tool graphical user interface which is beyond the scope of this manual.

10. FUNCTION FOR OBTAINING UNITS

The `unitsOfVariable()` function can be used to obtain the units of a variable as follows:

```
EXPERIMENT exp2 ON turbojet.default
DECLS
    STRING unitsv
BODY
    unitsv= unitsOfVariable("Bleed.g_branch[2].WF")
    WRITE("unitsv= %s\n",unitsv)
END EXPERIMENT
```

The output of this experiment could be:

```
unitsv= kg/s
```

If the variable did not have any units or if it did not exist, an empty string is returned. If the user wants to check first whether the variable exists, the `existsVariable()` function can be used.

11. FUNCTION FOR OBTAINING RELEASE REPORT

The versionReport() function returns a string with the identification of the version of the program including the following:

- Names of experiment, library, component and partition
- User who generated the experiment
- Date and time of creation
- Libraries used and their versions
- Name and version of program

An example of the use of this function is as follows:

```
EXPERIMENT exp2 ON turbojet.default
DECLS
  STRING vreport
BODY
  vreport= versionReport()
  WRITE("vreport= %s\n",vreport)
END EXPERIMENT
```

where vreport was previously declared as STRING type. It will be printed out:

```
Experiment: exp2 Library: GAS_TURBINE_EXAMPLES Component: turbojet
Partition: default
User: pce Date: 06/28/07 Time: 16:25:17
Library dependency list: CONTROL V0.0 GAS_TURBINE V0.0 GAS_TURBINE_EXAMPLES
V0.0 MATH V0.0
Programme & Version: PROOSIS V1.2.0
```


12. FUNCTION FOR OBTAINING AN INTERNAL RESIDUE

If the user wants to know the value of an internal residue of the Jacobian, the `getResidueValue()` function can be used with the following format:

```
residue1 = getResidueValue(1)
```

This function returns a REAL with the value of the internal residue 1. The internal residues are associated to the dynamic and algebraic variables of the Jacobian equations system.

To know the jacobian variables visualize the partition information file. In this file you can find a table with the Jacobian variables and their associated closure equations. The function allows the user to obtain any of the N residues at any time.

If the residue number you pass to the function is lower than zero or greater than the number of internal residues (or Jacobian variables), the function returns 0.0 and displays an error message.

13. UPGRADE MONITOR

13.1 GENERAL

This section will review the latest changes made in this version in comparison to the previous version. An important change is that now the user can load an experiment, close it and then reload it or another one from the monitor. This is done by using the “Close Model” option or by pressing “Ctrl+X”, as shown in Fig.1 below.

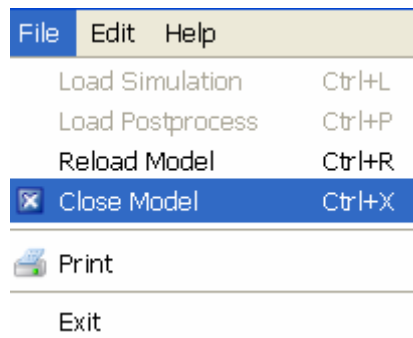


Fig.1. Close Model

13.2 MONITOR TOOLBAR

Depending on whether a simulation or a post-process is loaded, certain buttons appear on the toolbar. In the case of a simulation, the toolbar is as shown in Fig.2 below:



Fig.2. Toolbar experiment

As can be seen, some buttons from the previous version have disappeared and the following new ones have appeared:

New Table Widget (Ctrl + w): creates a new table

If a post-process is loaded, the new toolbar, with less buttons, appears as shown in Fig.3 below:

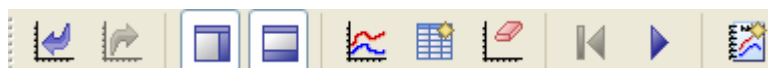


Fig.3. Toolbar post-process

There is a new button in this version:



New Postprocess Report(): creates a new report

The old functions which were previously accessible from the toolbar can now be accessed from a context menu activated by the right-clicking with the mouse.

Add new tab: By right-clicking on a tab, the user can add a new one, by selecting the “Add Tab” option on the context menu as shown in Fig.4 below (this can also be done by pressing “Ctrl+T”).

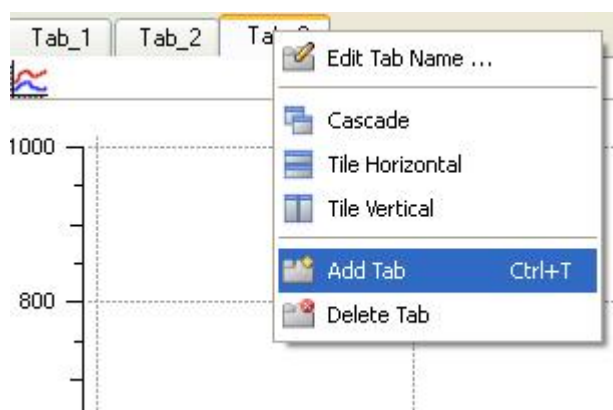


Fig.4. Add tab

Delete tab: In the same way, the user can delete a tab by selecting the “Delete Tab” option from the context menu, as shown in Fig.5 below:

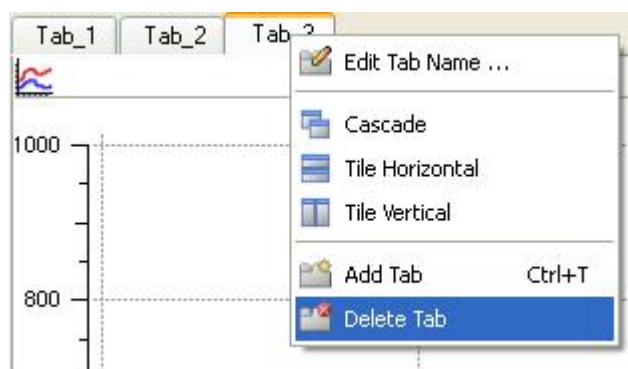


Fig.5. Delete tab

Edit plot: To edit a plot, right-click on the plot to be edited and select the “Edit plot” option. This can also be done by pressing “Ctrl.+E”, as shown in Fig.6 below:

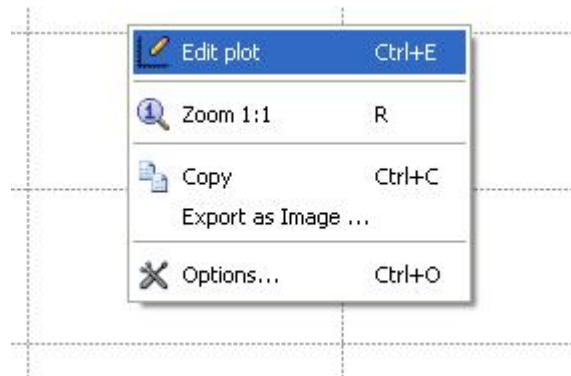


Fig.6. Edit plot

Edit watch: To edit a watch right-click on the watch to be edited and select the “Edit watch” option. This can also be done by pressing “Ctrl.+E”, as shown in Fig.7 below:

Variable	Value	Units
m1	2042.6	--
m2	660.6	--
m3	291.8	--
h	20.1	--
k1		--
k2		--
y3		--
y2	0	--

Fig.7. Edit Watch

The “Remove” option eliminates the selected variables from the match.

13.3 MONITOR WORKSPACE

Some changes have also been made to the general monitor workspace, such as the inclusion of units in the plots. They are shown on the y-axis only in the case that all the variables selected in the plot are in the same units. Fig.8 below shows that all the units of the variables x , $y1$, $y2$ and $y3$ are in meters (m).

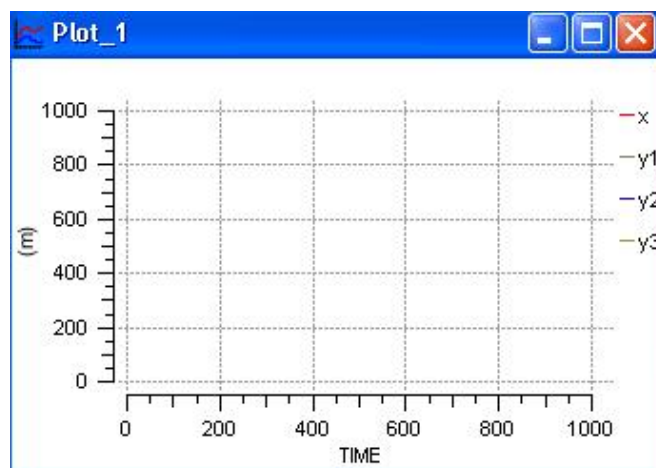


Fig.8. Units in a plot

13.4 PLOT PROPERTIES

The configuration options of a plot have increased in comparison with the previous version. New tabs have been added in the properties dialogue: “Select Group” for when the plot is not a map and “Map Style” for when it is a map.

The properties can be configured either by right-clicking with the mouse on the plot and selecting the “Options” option, or by pressing “Ctrl+O”, as shown in Fig.8 below:

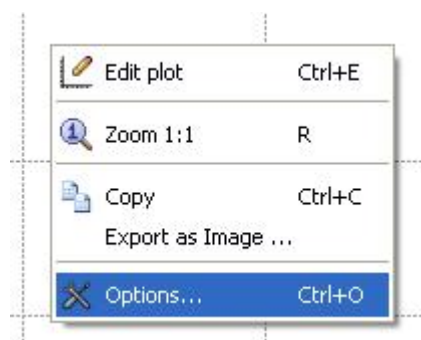


Fig.9. Plot properties

13.5 TABLE VISUALIZATION

The “table widgets” show, in table format, the information relative to the different calculations carried out during the simulation of an experiment. They also show the values of the variables during each one of these calculations.

To create a “table widget” or table, press the “New Table Window” button or press “Ctrl+W”.

A menu then appears and the user can select the variables to be shown, as per Fig.10 below. If “ShowTime” is activated, then the time will also be shown. If “Row/Column” is activated, then the variables will be shown in rows. If “Row/Column” is not activated, then the variables will be shown in columns.

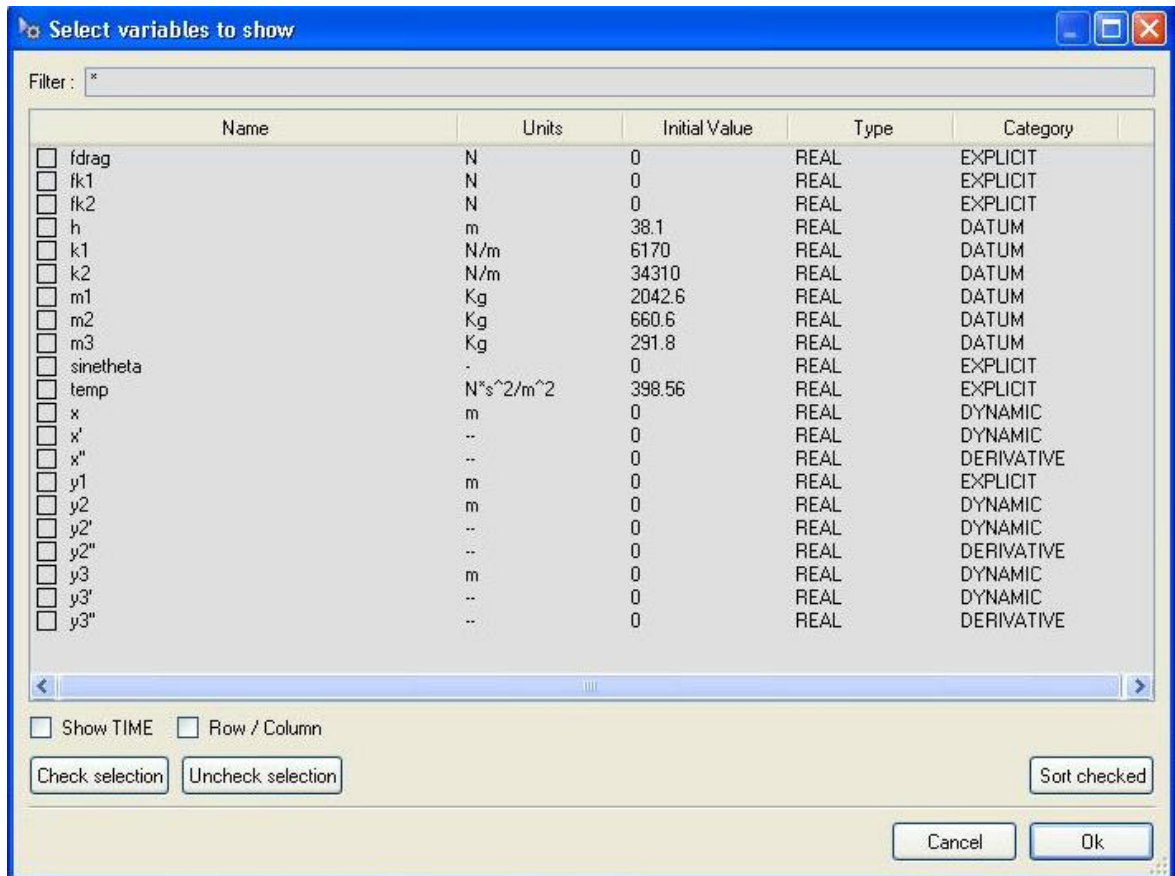


Fig.10. Table widget

If the “Cancel” button is pressed, the table will not be created and all the selections made in the dialogue box will be voided. Pressing the “OK” button will create the table as per the parameters selected. If no variable is selected and the OK button is pressed, the table will be created but with no information on any variable.

For example, if the user wants to view the values of variables x , x' and x'' , to show the time, and to sort them by rows, the dialogue box should be filled in as shown in Fig.11 below:

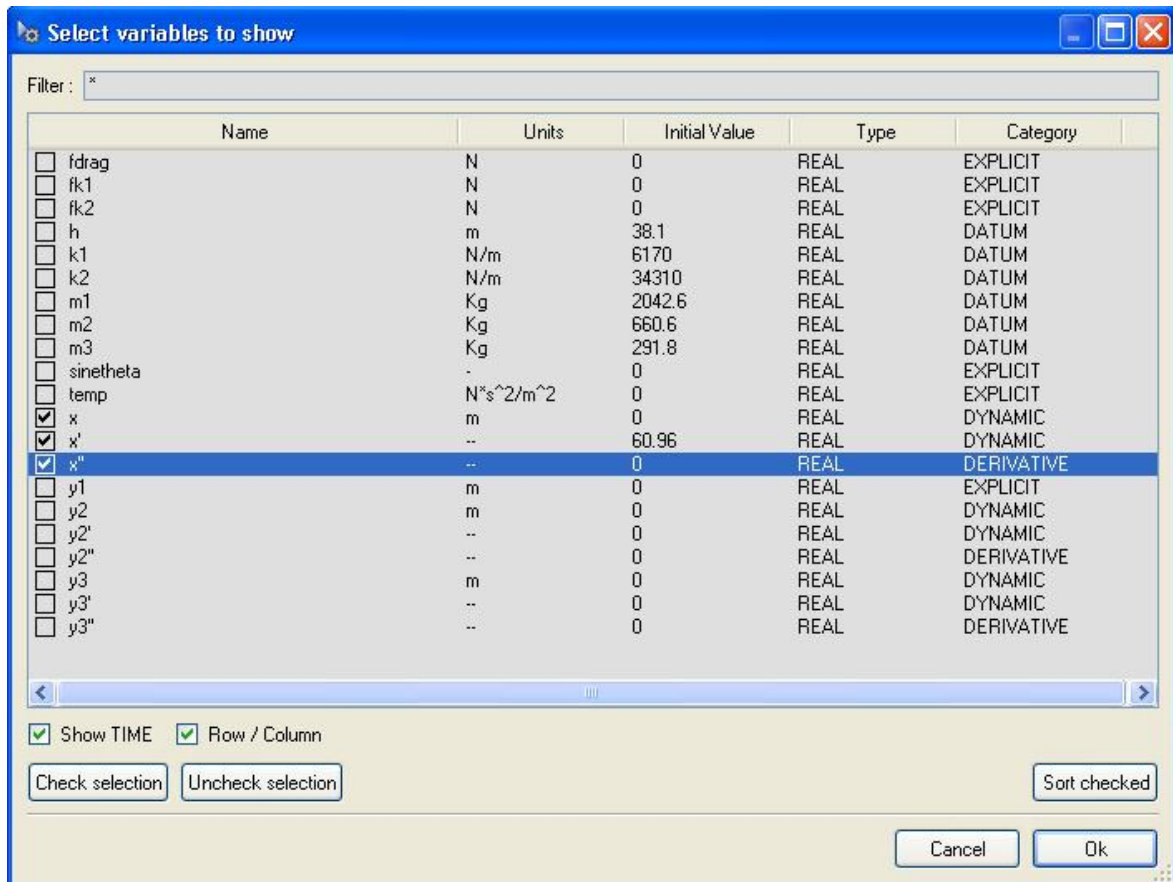


Fig.11. Table widget with data

If the user now presses “OK”, the table will be created as shown in Fig.12 below:

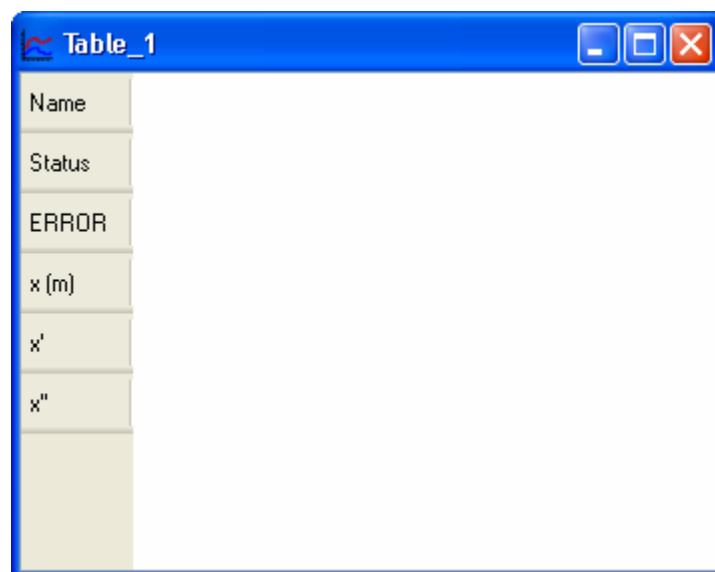


Fig.12. Table widget (rows)

The first column contains different names some of which correspond to the variables selected (*TIME*, *x*, *x'* and *x''*.) while the others have the following meanings:

Name: name of the calculation performed

Status: status of convergence

ESI: error code

Both the variables and the above parameters will have a value for each calculation carried out during the simulation. If the user starts the simulation of the experiment, the table will fill up as shown in Fig.13 below:

	1	2	3	4	5	6
Name	transient #1	transient #1	transient #1	transient #1	transient #1	transient #1
Status	INTEG_BEGIN	IS_EVENT	IS_EVENT	IS_EVENT	IS_EVENT	IS_CINT
ESI	0	0	0	0	0	0
TIME	0	4.52827e-05	4.52827e-05	0.00573156	0.00573156	0.05
x (m)	0	0.00276043	0.00276043	0.349396	0.349396	3.04799
x'	60.96	60.96	60.96	60.96	60.96	60.9593
x''	0	0	-4.37708e-11	-8.8741e-05	-8.8741e-05	-0.0581889

Fig.13. Table widget (simulation running)

For each calculation, a column showing the values of the selected parameters is inserted.

Once a table has been created it is possible to edit it to change the variables to be viewed or the layout by columns or files. It is also possible to copy the table and to insert it into a text file or Excel file. To do this, right-click on the table and a context menu will appear, as shown in Fig.14 below.



	1	2	3	4	5	6
Name	transient #1	transient #1	transient #1	transient #1	transient #1	transient #1
Status	INTEG_BEGIN	IS_EVENT	IS_EVENT	IS_EVENT	IS_EVENT	IS_CINT
ESI	0	0	0	0	0	0
TIME	0	4.52827e-05	4.52827e-05		0.000173156	0.05
x (m)	0	0.00276043	0.00276043		0.000000396	3.04799
x'	60.96	60.96	60.96			60.9593
x''	0	0	-4.37708e-11	-8.8741e-05	-8.8741e-05	-0.0581889

Fig.14. Table widget menu

To edit the table, click on the “Edit table” option, and to copy it, click on the “Copy” option. It is also possible to do this by pressing “Ctrl+E” or “Ctrl+C” respectively.

If the user wants to edit the table, the same dialogue box as when the table was first created will appear, only with the currently selected elements marked. For example, to change the layout to columns and to show variable y_1 instead of x , the editing dialogue would be as shown in Fig.15 below:

Name	Units	Initial Value	Type	Category
<input type="checkbox"/> fdrag	N	6547.72476	REAL	EXPLICIT
<input type="checkbox"/> fk1	N	2733.37543	REAL	EXPLICIT
<input type="checkbox"/> fk2	N	6221.308	REAL	EXPLICIT
<input type="checkbox"/> h	m	38.1	REAL	DATUM
<input type="checkbox"/> k1	N/m	6170	REAL	DATUM
<input type="checkbox"/> k2	N/m	34310	REAL	DATUM
<input type="checkbox"/> m1	Kg	2042.6	REAL	DATUM
<input type="checkbox"/> m2	Kg	660.6	REAL	DATUM
<input type="checkbox"/> m3	Kg	291.8	REAL	DATUM
<input type="checkbox"/> sinetheta	-	0.961492625	REAL	EXPLICIT
<input type="checkbox"/> temp	$N^{\circ}s^2/m^2$	279.162159	REAL	EXPLICIT
<input checked="" type="checkbox"/> x	m	133.292477	REAL	DYNAMIC
<input checked="" type="checkbox"/> x'	--	9.78636505	REAL	DYNAMIC
<input type="checkbox"/> x''	--	-2.57330884	REAL	DERIVATIVE
<input checked="" type="checkbox"/> y1	m	100.530785	REAL	EXPLICIT
<input type="checkbox"/> y2	m	50.043887	REAL	DYNAMIC
<input type="checkbox"/> y2'	--	4.78544334	REAL	DYNAMIC
<input type="checkbox"/> y2''	--	-1.14223002	REAL	DERIVATIVE
<input type="checkbox"/> y3	m	49.8625606	REAL	DYNAMIC
<input type="checkbox"/> y3'	--	4.84302748	REAL	DYNAMIC
<input type="checkbox"/> y3''	--	-1.1186318	REAL	DERIVATIVE

Fig.15. Table widget edition

By pressing “Cancel” the changes would not take effect, and by pressing “OK”, the changes would be applied and the table would be shown as in Fig.16 below:

	Name	Status	ERROR	x (m)	x'
1	transient #1	INTEG_BEGIN	0	0	60.96
2	transient #1	IS_EVENT	0	0.00276043	60.96
3	transient #1	IS_EVENT	0	0.00276043	60.96
4	transient #1	IS_EVENT	0	0.349396	60.96
5	transient #1	IS_EVENT	0	0.349396	60.96
6	transient #1	IS_CINT	0	3.04799	60.9593
7	transient #1	IS_CINT	0	6.09577	60.9486
8	transient #1	IS_CINT	0	9.1423	60.9046
9	transient #1	IS_CINT	0	12.1851	60.7944

Fig.16. Table widget (columns)

As can be seen, the parameters are now displayed by columns and variable *y1* is shown instead of variable *x*. As the *y1* variable was inserted at an advanced stage of the simulation, its value has not been calculated for those instants and its value is shown as “- -”.

If an error occurs during the simulation, it is shown by highlighting in red the calculations that gave rise to the error, as shown in Fig.17 below.

	Name	Status	ERROR	y[1]	y[4]
1	transient #1	INTEG_BEGIN	0	3	0
2	transient #1	INTEG_NOK	215	3	0

Fig.17. Table widget error

14. CALCULATION WIZARD

To start the Calculation Wizard, right-click the predefined partition of a component and select the option “New Calculation”.

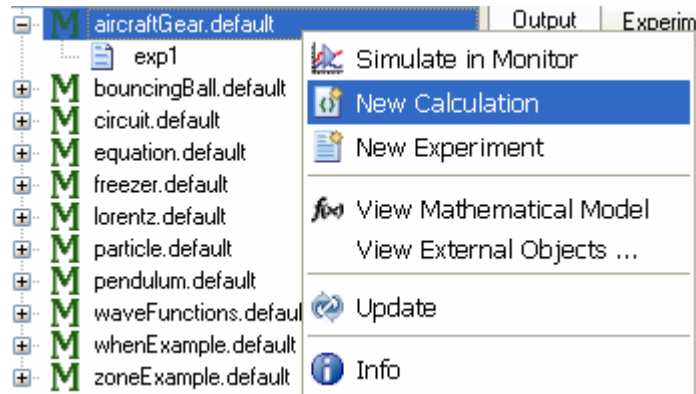


Fig.18. Calculation Wizard Selection

In the Calculation Wizard window that opens, right-click on the new calculation to add different “cases” and rename or delete the experiment.

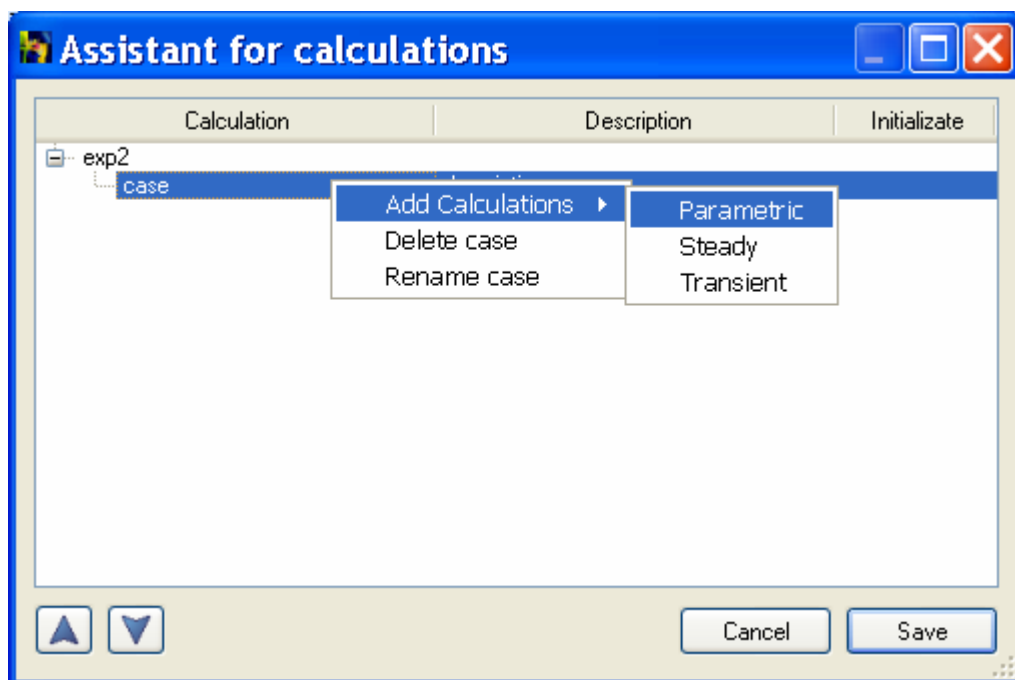


Fig.19. Calculation Wizard Window

Similarly, by right-clicking on the added “case”, you can add different calculations and rename or delete the “case”.



14.1 PARAMETRIC CALCULATION

As in the “Design Calculation”, double-click or right-click the case to add a parametric calculation.

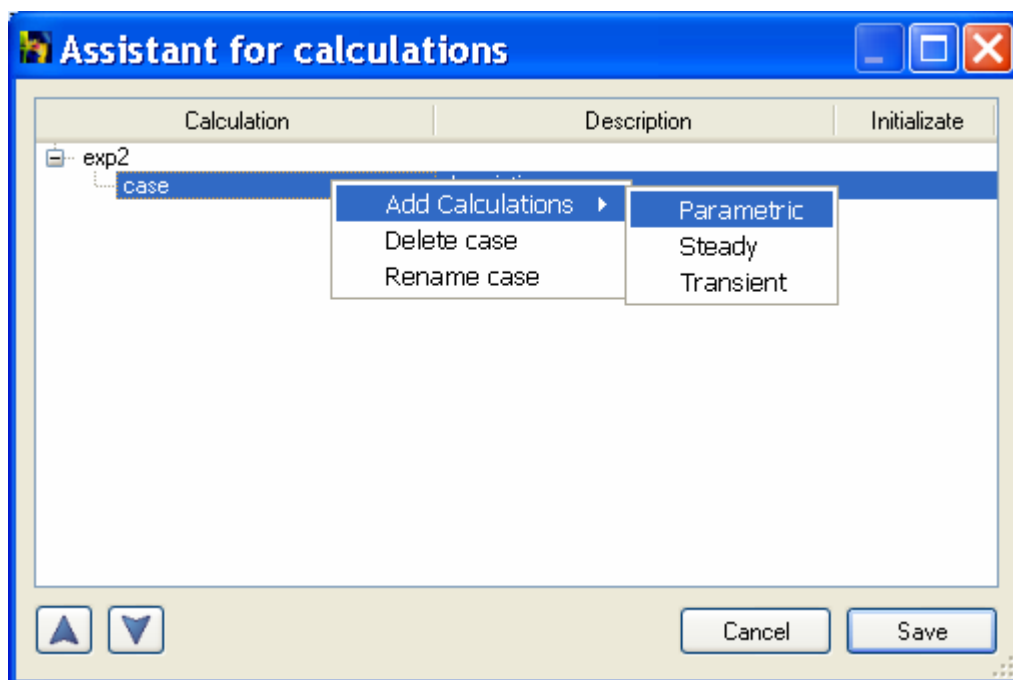


Fig.20. Parametric Calculation Selection

To edit the Wizard, double-click “Edit”.

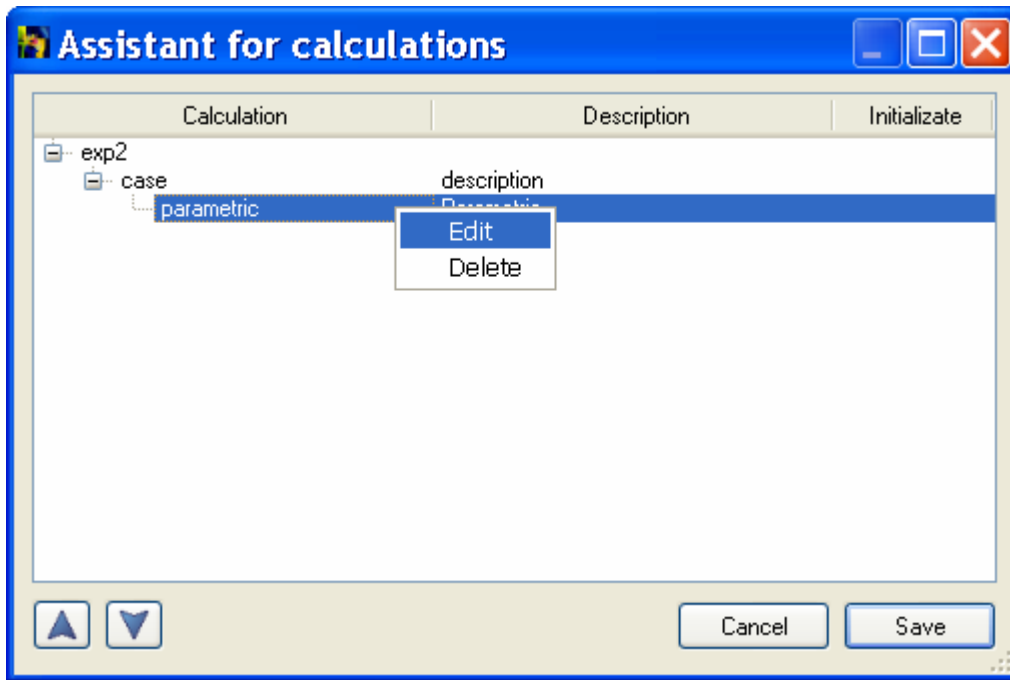
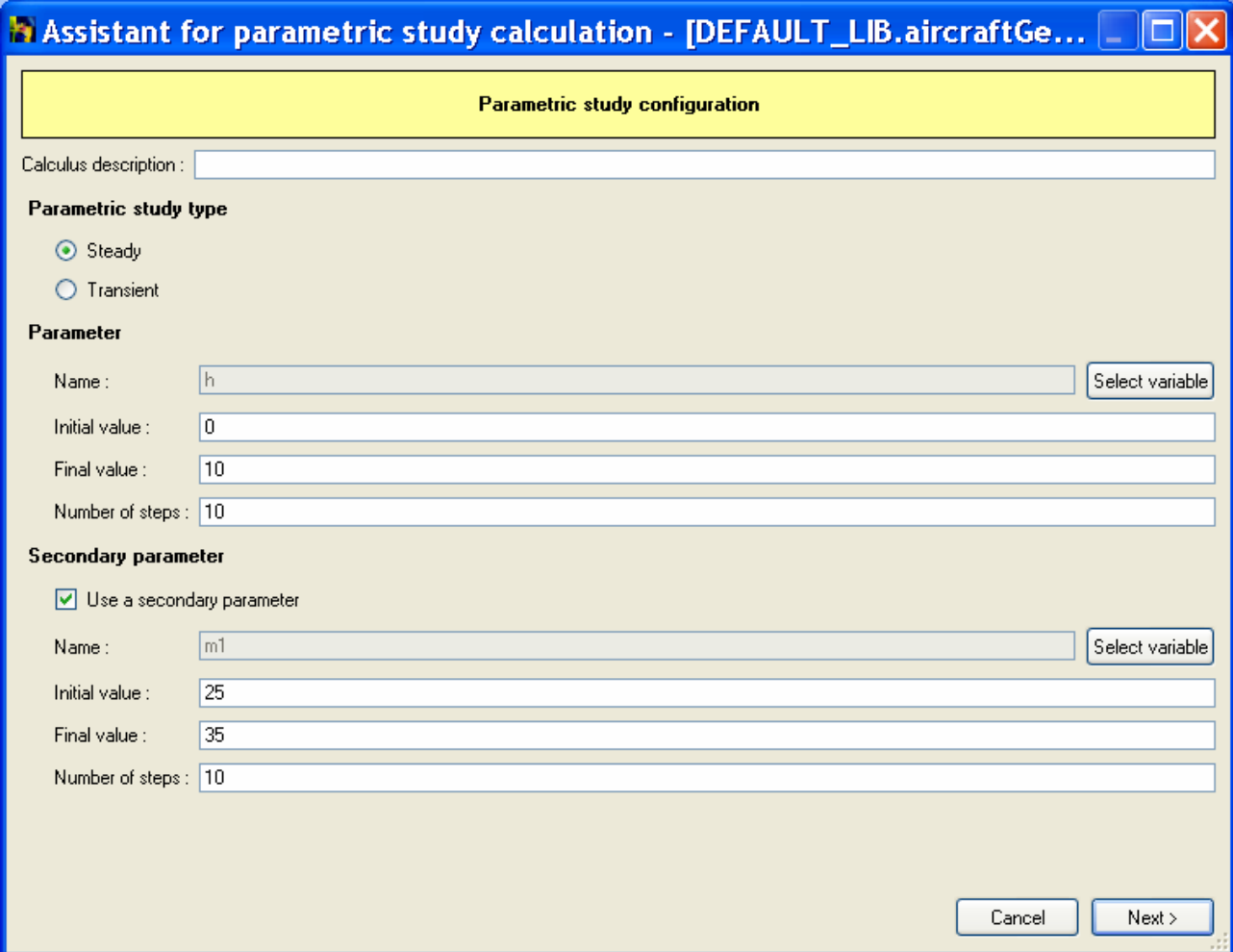


Fig.21. Parametric Calculation Edition

The first selection to be made is whether the parametric study is going to be steady or transient. A steady study can be carried out on one or two parameters. However, only one parameter is selected for a transient parametric study.



Assistant for parametric study calculation - [DEFAULT_LIB.aircraftGe...

Parametric study configuration

Calculus description :

Parametric study type

Steady
 Transient

Parameter

Name :

Initial value :

Final value :

Number of steps :

Secondary parameter

Use a secondary parameter

Name :

Initial value :

Final value :

Number of steps :

Fig.22. Parametric Study Configuration

When defining the parameters, the following have to be fixed:

Name of the variable

Initial and final values

Number of steps

The tolerance, number of iterations, number of decimals, etc, is entered in the window illustrated below, where it is also possible to create a results file (Create report) configured by the user.

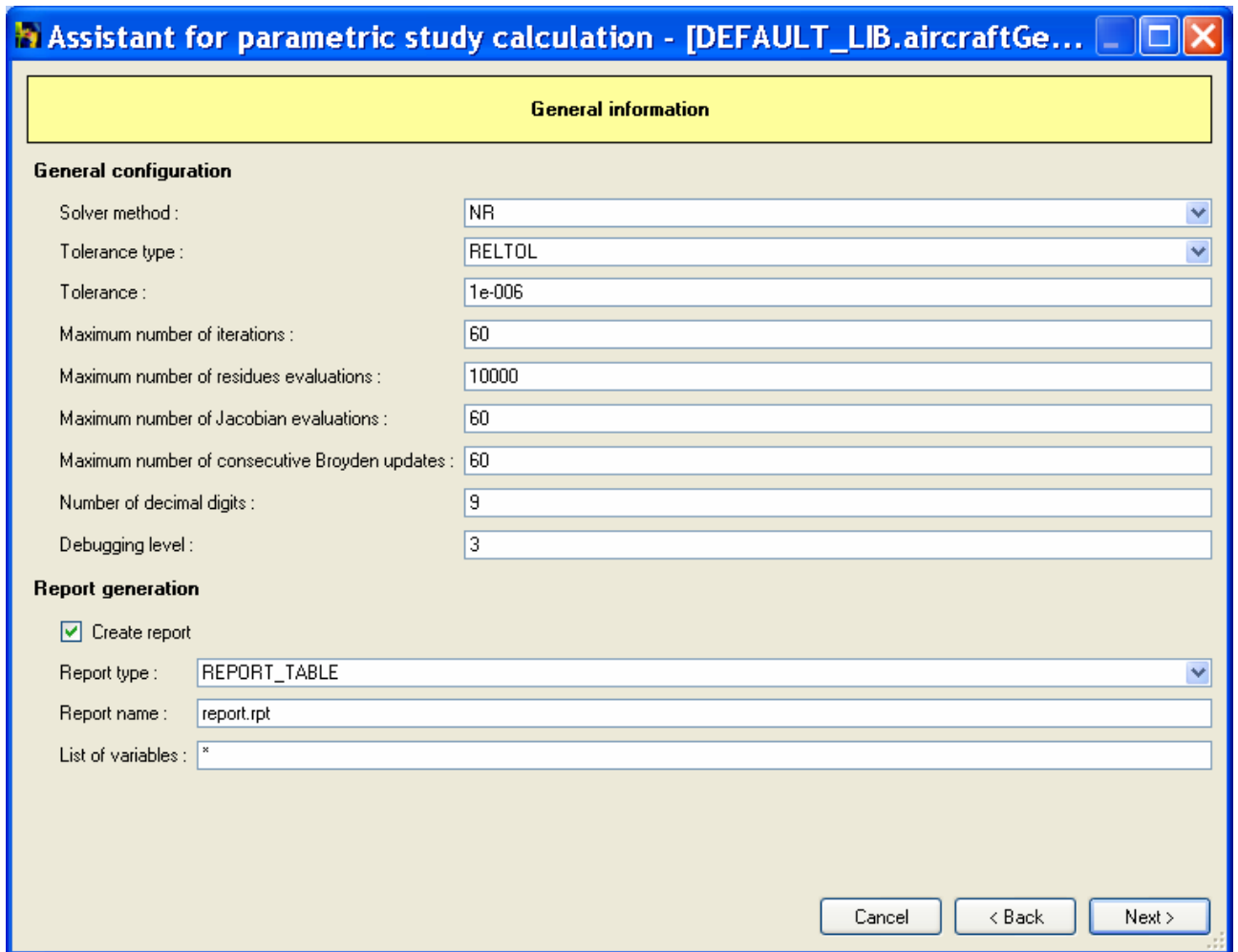


Fig.23. General Parametric Configuration

The user must then carry out the following:

Fix the values of the boundary variables

Initialize:

Dynamic and algebraic variables pertaining to the partition

Closure equations:

Non-linear dynamic and algebraic equations pertaining to the partition



Assistant for parametric study calculation - [DEFAULT_LIB.freezer.de...]

Set boundary conditions, initialize variables and configure non-linear boxes

Boundaries

	Name	Value	Description
1	mc	1	Heat capacity (J/°C)

Algebraic and dynamic variables

	Name	Value	DxAbs	DxRel	MaxStepAbs	maxStepRel	scrip
1	tr	2	1.5e-008	1.5e-008	1e+018	1e+018	Ro...
2	qc	5	1.5e-008	1.5e-008	1e+018	1e+018	Co...
3	te	70	1.5e-008	1.5e-008	1e+018	1e+018	Ev...

Non-linear boxes residues

	Left expression	Operator	Right expression	Toltype	Tolerance
1	0	=	tr'	1e-006	ABSTOL
2	qc - (wk * cam * (tc + 273.15) / (t...	=	0	1e-006	ABSTOL
3	te - (tr - qc / ua)	=	0	1e-006	ABSTOL

Fig.24. Values of Boundary, Dynamic and Algebraic Variables

Finally, the Wizard displays a summary of the design calculation that has been configured.

After saving the design calculation, the experiment can be executed.

The results can be displayed as follows:

```

Parameter :
1 : Compressor.CG1

Algebraics :
1 : Compressor.XN
2 : Compressor.BETA
3 : Inlet.W
4 : Turbine.BETA

```

Parameter	1	Values			
		1	2	3	4
1	5.000000[12578.06023	0.60629	101.00533	0.50038]
	5.050000[12529.53036	0.61935	101.88777	0.50054]
	5.100000[12473.41603	0.63218	102.74758	0.50065]
	5.150000[12408.65053	0.64490	103.58842	0.50071]
	5.200000[12329.28414	0.65760	104.39892	0.50069]
	5.250000[12227.58121	0.67041	105.15046	0.50054]
	5.300000[12110.25866	0.68356	105.81574	0.50027]
	5.350000[12007.20255	0.69714	106.41955	0.50004]
	5.400000[11917.74844	0.71155	106.90630	0.49982]
	5.450000[11829.80916	0.72769	107.12761	0.49951]
	5.500000[11746.73586	0.74536	107.12270	0.49913]

Fig.25. Parametric Results

The values of the dynamic and algebraic variables are displayed on the monitor for each parameter value

In a file configured by the user

14.2 STEADY CALCULATION

There are two ways to perform steady calculations:

With initialization: the simulation will use the values introduced into the Wizard as the values of the boundary variables

Without initialization: the simulation will use the values saved from the previous simulation as the values of the boundary variables

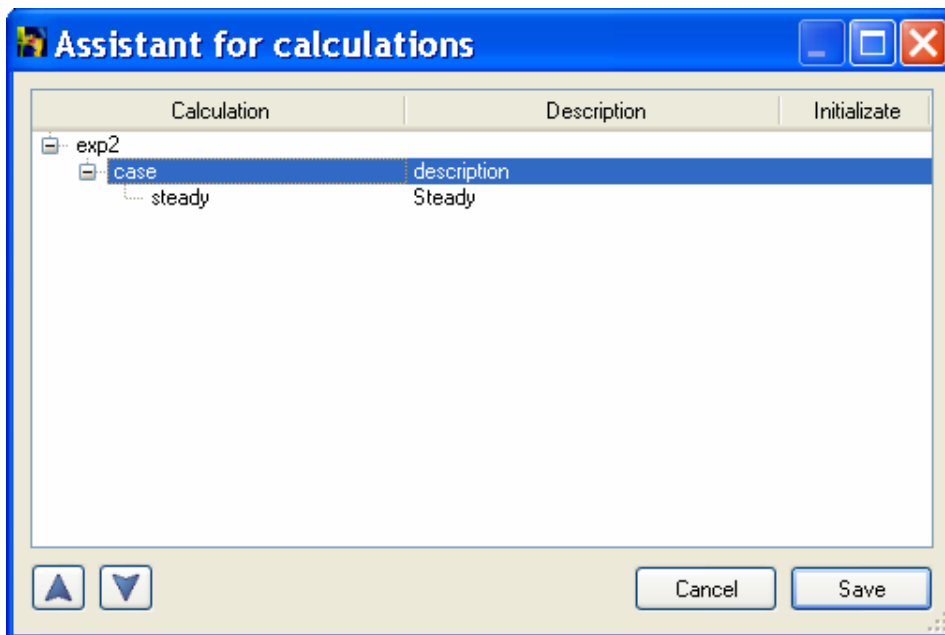
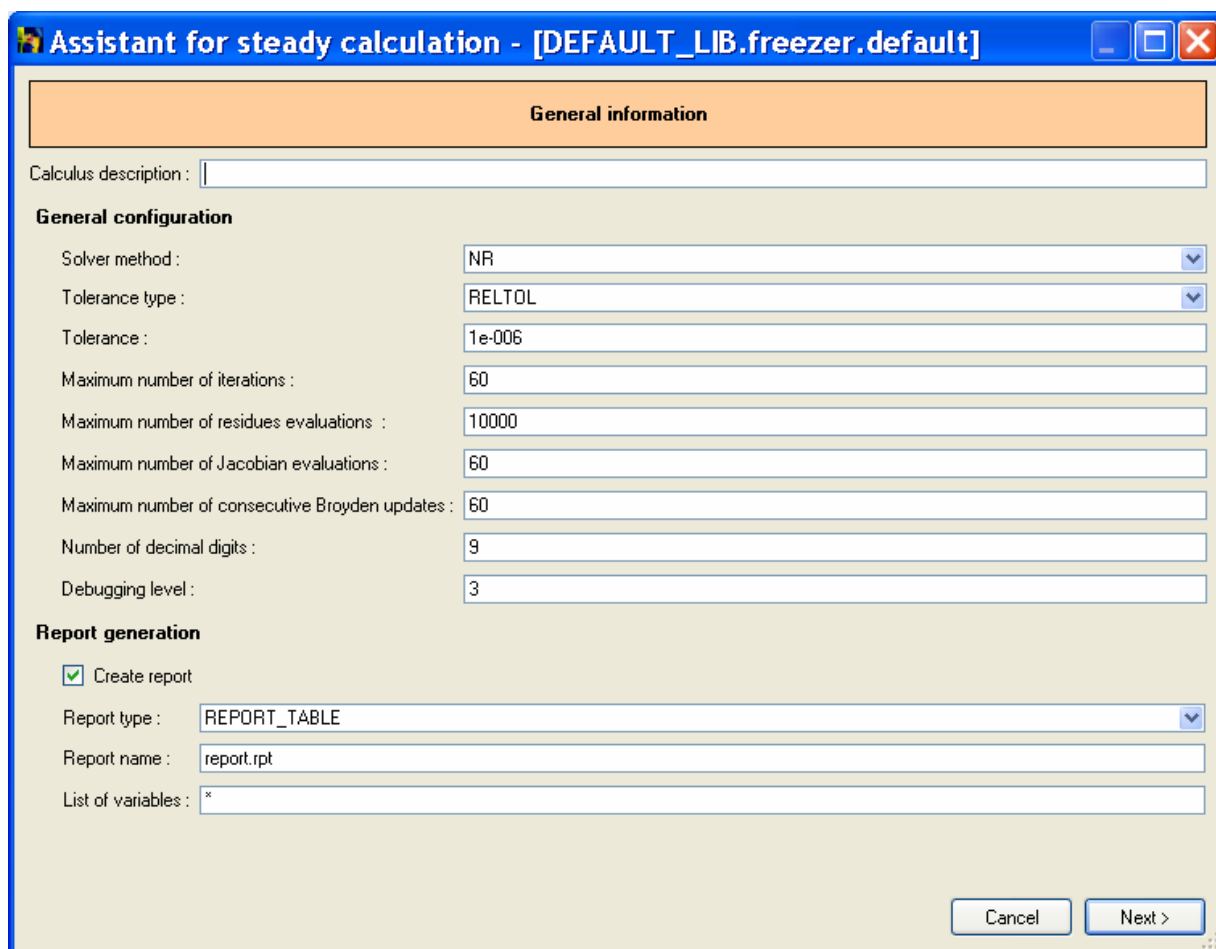


Fig.26. Steady Calculation Edition

The solver method, tolerance, number of iterations, number of decimals, etc, is entered in the window illustrated below, where it is also possible to create a results file (Create report) configured by the user.



Assistant for steady calculation - [DEFAULT_LIB.freezer.default]

General information

Calculus description : |

General configuration

Solver method : NR

Tolerance type : RELTOL

Tolerance : 1e-006

Maximum number of iterations : 60

Maximum number of residues evaluations : 10000

Maximum number of Jacobian evaluations : 60

Maximum number of consecutive Broyden updates : 60

Number of decimal digits : 9

Debugging level : 3

Report generation

Create report

Report type : REPORT_TABLE

Report name : report.rpt

List of variables : *

Cancel Next >

Fig.27. General Steady Information

The user must then carry out the following:

Fix the values of the boundary variables

Initialize:

Dynamic and algebraic variables pertaining to the partition

Closure equations:

Non-linear dynamic and algebraic equations pertaining to the partition

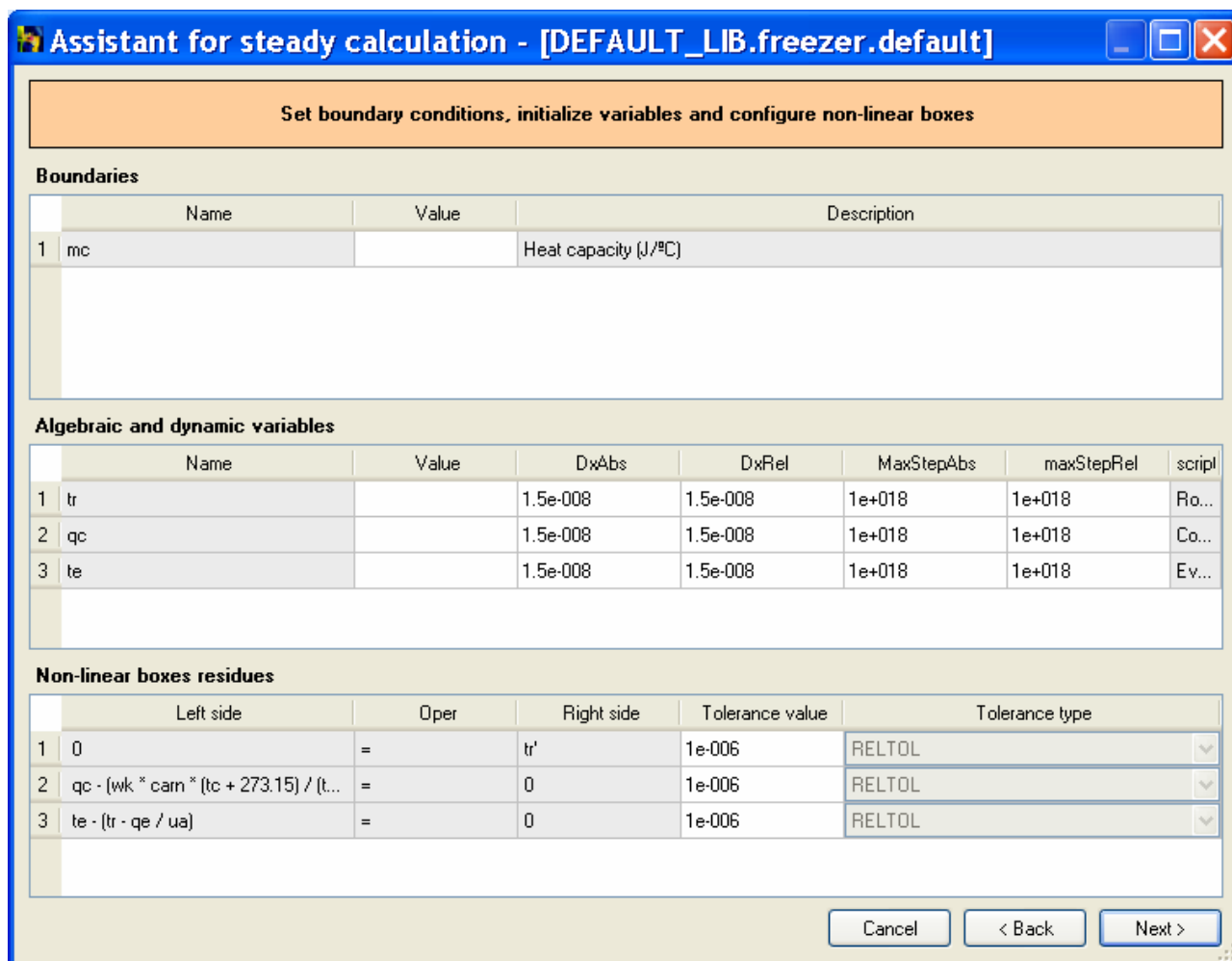


Fig.28. Values of Boundary, Dynamic and Algebraic Variables

Finally, the Wizard displays a summary of the design calculation that has been configured. After saving the design calculation, the experiment can be executed.

The results can be displayed as follows:

- In a sensitivity matrix on the monitor
- In a file configured by the user