
Title : *Main operating principles of Code_Aster*
Author(s) : **J.M. PROIX, J.R. LEVESQUE**
Translator(s) : **C. LUZZATO**

Date : 26/06/03
Key : U1.03.00-D Page : 1/15

Organisme(s) : EDF-R&D/AMA

User Manual
Booklet U1.0-: Synopsis
Document: U1.03.00

Main operating principles of Code_Aster

Summary:

This document presents a quick overview of the operating principles of *Code_Aster*, as well as basic guidance rules.

It remains a general description of *Code_Aster*. Readers will have to refer to other documents to obtain information concerning specific details.

1 General Principles

Version 6 of *Code_Aster* enables structural calculations of thermal, mechanical, thermo-mechanical or coupled thermo-hydro-mechanical phenomena, with linear or non-linear behaviours, as well as closed space acoustics calculations.

The non linear calculations concern material behaviours (plasticity, viscous plasticity, damages, metallurgic effects, concrete hydration and drying, ...), important deformations or rotations, and frictional contact. Users should refer to the presentation chart of version 6 for a detailed description of the different features.

Usual research in the industrial sector requires mesh generators and graphical representation tools, which are not included in the Code. However other tools can be used for such tasks through interface procedures integrated within the Code.

To conduct a study, the user usually needs to prepare two files:

- The **mesh** file:

To define a general geometric and topologic description of the mesh, without choosing the formulation type of the finite elements used, or the physical phenomenon that will be modelled.

This mesh file is usually created through an interface integrated with *Code_Aster* from a file produced by mesh generating software used in pre-processing (GIBI, GMSH, IDEAS ...).

The information that must be included in the file is specific to *Code_Aster*. It defines usual entities of the finite element method:

- **nodes**: points defined by a **name** and by their **cartesian coordinates** in **2D** or **3D** space
- **mesh elements**: **named** topologic shapes, plane or with volume (point, segment, triangle, quadrangle, tetrahedral, ...), upon which may be applied different types of finite elements, boundary conditions or loading.

To improve the quality standard and to simplify modelling operations and result analysis, we can define, in the mesh file, levels of higher entities. These can have any characteristic in common, and can be used by directly calling their name.

- **node groups** : named list containing names of nodes,
- **mesh element groups** : named list containing names of mesh elements.

Note that all the manipulated geometric entities (nodes, mesh elements, node groups, mesh element groups) are **named by the user** and can be used at any moment by calling their name (**8 characters maximum**). The user will be able to use this functionality to explicitly identify specific parts of the studied structure and thus render result analysis easier. The numbering of the entities is never explicit: it is only used internally to point at the values of the different associated variables.

- the **command** file : to define the command text which enables the user to:
 - read and perhaps complete the data in the mesh file (or in other external result files)
 - assign modelling data to mesh elements or nodes,
 - induce different processing operations: calculations, specific post-processing,
 - edit results in different files.

The command text refers to the names of the mesh elements defined in the mesh file. It also allows defining new groups at any time.

From a computing point of view, the mesh and command files are **ASCII** open source files. Their main characteristics are given below:

mesh file syntax:

- the length of the line is limited to 80 characters, the valid characters are:
 - the 26 upper case letters **A-Z** and the 26 lower case letter **a-z** automatically converted in upper case letters, except in texts (given in quotes),

- the 10 single digit numbers **0-9** and number representation symbols (+ - .),
- the **underscore** symbol `_` which can be used in key words or names,
- a word must always begin with a letter,
- the **space** is always a separator,
- The `%` character indicates the beginning of a **comment** line, which continues until the end of the line.
- The other reading rules are detailed in fascicule [U3.01.00]

Command file syntax

- The syntax is linked to the Python programming language, allowing to include instructions from that language
- The `#` character indicates the beginning of a **comment** line, which continues until the end of the line.
- Commands must begin in column 1, unless they are part of an indented bloc (loop, test)

The other reading rules are detailed in fascicule [U1.03.01].

2 Meshing

2.1 General information

The structure and syntax of the **mesh** file are detailed in **fascicule [U3.01.00]**.

This file can be written (for simple meshes) or manually modified using any text editor. It is read in open source, and is structured in blocs of information or sub files by specific key words.

Many conversion utilities are available to allow the conversion of mesh files created by other software (IDEAS, GIBI, GMSH...) or in MED format.

2.2 The Code_Aster mesh file

The *Aster* mesh file is read from the first line, and up to the first occurrence of a line beginning by the word **FIN**. **This key word is compulsory**. The mesh file is structured in independent sub files beginning with a **key word** and ending with the set key word **FINSF**.

This file must contain at least two sub files:

- The coordinates of all of the mesh nodes in a 2D (COOR_2D) or 3D (COOR_3D) Cartesian coordinate system.
- The description of all of the mesh elements (TRIA3, HEXA20, etc ...), which will later be affected physical properties, finite elements, boundary conditions, or loading.

It can contain node groups (GROUP_NO) or mesh element groups (GROUP_MA) to facilitate assignment operations, but also result analysis.

At this stage, it is essential to explicitly create the mesh elements located on the stress application borders for loading and boundary conditions. We shall thus find in the weaving file:

- *The edge mesh elements for the necessary 2D elements,*
- *The face mesh elements for the necessary massive 3D elements,*
- *The mesh element groups for associated edges and/or faces.*

This constraint is bearable when an interface is used. Indeed, the interface works from the indications given during the mesh generation (see documents `PRE_IDEAS [U7.01.01]` or `PRE_GIBI [U7.01.11]`).

2.3 Mesh element description

The description convention for the topology of the mesh elements, and conditions of use of the different kinds of mesh elements, are described in fascicule [U3.01.00].

The main types of recognised mesh elements are identified by the following reserved key words [U3.01]:

/ POI1				punctual mesh element
/ SEG2	/ SEG3	/ SEG4		segment with 2, 3 or 4 nodes
/ TRIA3	/ TRIA6	/ TRIA7		triangle with 3, 6, or 7 nodes
/ QUAD4	/ QUAD8	/ QUAD9		quadrangle with 4, 8 or 9 nodes
/ HEXA8	/ HEXA20	/ HEXA27	hexahedra	with 8, 20 or 27 nodes
/ PENTA6	/ PENTA15		pentahedra	with 6 or 15 nodes
/ TETRA4	/ TETRA10		tetrahedral	à 4 or 10 nodes
/ PYRAM5	/ PYRAM13		pyramid	with 5 or 13 nodes

2.4 Interfaces

These interfaces allow the conversion of files, with or without format, used by various software or calculation codes, into the conventional *Aster* mesh file format.

The available interfaces are compatible with files from the IDEAS mesh generator, the GIBI mesh generator from CASTEM 2000, and the GMSH mesh generator. These interfaces can also convert mesh files in the MED exchange format.

2.4.1 The IDEAS universal file

The interface is called with the command `PRE_IDEAS` [U7.01.01]
The convertible file is the universal file defined in the I-DEAS documentation (see Fascicule [U3.03.01]).
The version of the IDEAS software used is automatically recognised.

An IDEAS universal file contains several independent blocs called “**data sets**”. Each “**data set**” is numbered and enclosed in a chain of `-1` . The “data sets” recognised by the interface are described in fascicule [U3.03.01].

2.4.2 The GIBI mesh file

The interface is called with the command `PRE_GIBI` [U7.01.11]).
The convertible file is the ASCII file outputted by the command `SAUVER FORMAT` from CASTEM 2000. A precise description of the interface is given in [U3.04.01].

2.4.3 The GMSH mesh file

The interface is called with the command `PRE_GMSH` [U7.01.31]).
The convertible file is the ASCII file outputted by the command `SAVE` from GMSH.

2.4.4 The MED format mesh file

The interface is called with the command `LIRE_MAILLAGE (FORMAT : 'MED')` [U4.21.01]).

MED (Modelling and Exchange of Data) is a neutral data format developed by EDF R&D to exchange data between calculation codes. The MED files are portable binary files. Reading and MED file with `LIRE_MAILLAGE` enables you to reacquire a mesh created with any other code capable of producing MED files on any other machine. This data format is specifically used to exchange mesh files and results between ASTER and the mesh refining tool HOMARD. A precise description of the interface is given in [U7.01.21].

2.5 Using incompatible mesh files

The finite elements method recommends the use of even meshes, without discontinuity, to obtain a precise convergence towards the solution of the continuous problem. However, it can be necessary to use incompatible meshes for certain models: the meshes do not correspond on either side of the boundary. The connection of these two meshes is then managed at the command file level by the key

word LIAISON_MAIL from the command AFFE_CHAR_MECA. This allows the connection of a finely meshed zone to a zone where only a coarse mesh is necessary.

2.6 Adaptive mesh

It is possible to adapt the mesh from an initial mesh to minimise calculation error. This is done with the macro command MACR_ADAP_MAIL, which uses the HOMARD software. The adaptive mesh generating software HOMARD works with meshes made from segments, triangles, and tetrahedral.

This mesh adaptation occurs after code ASTER the computed the calculation error.. Depending on its value for each mesh element, HOMARD will automatically modify the mesh accordingly. It is also possible to interpolate temperature or movement fields with the nodes from the old mesh and save the result in the new mesh [U7.03.01].

3 Commands

3.1 The command file

The command file contains a group of commands written in a language specific to Code_Aster. In addition to the characteristics described in paragraph 1, detailed syntax of the language can be found in fascicule [U6.02.00]. These commands are analysed and executed by a software layer of Code_Aster called "supervisor"

3.2 The role of the supervisor

The supervisor completes different tasks, as for example:

- A **verification phase**, where the command file is interpreted
- An **execution phase, which managed the execution of the** interpreted commands.

These tasks are detailed in fascicule [U1.03.01].

The command file is processed from the first occurrence of **DEBUT()** or **POURSUITE()**, and up to the first occurrence of the command **FIN()**. The commands located before **DEBUT()** or **POURSUITE()** and after **FIN()** are not executed, but must have correct syntax.

- **Syntax verification phase :**
 - All commands are read and syntax is verified. A message is displayed for all detected syntax errors, but the analysis is not stopped.
 - All concepts used as arguments are verified to check that they have all been declared, in a past command, as concepts produced by an operator. The concept type is also checked to verify that it matches the type required for this argument.
- **Execution phase :**
 - The supervisor successively activates the various operators and procedures, which execute the scheduled tasks.

3.3 Principles and syntax of the command language

Because of its modular conception, Aster can be understood as a succession of independent commands:

- The **procedures**, which do not directly produce results, but manage and insure exchanges with external files
- The **operators**, which operate mechanisms relating to data management and calculation, and produce a **concept** result to which the user gives a name.

These concepts represent data structures that the user can manipulate. The **concepts** are given a **type** when they are created and can only be used as entry arguments which require the same type.

Title : Main operating principles of Code_Aster
Author(s) : J.M. PROIX, J.R. LEVESQUE
Translator(s) : C. LUZZATO

Date : 26/06/03
Key : U1.03.00-D Page : 6/15

The procedures and the operators therefore exchange the necessary information and values using **named concepts**.

The complete syntax and its implication on the writing of the command file are detailed in fascicule [U1.03.01]. An example depicting a few commands is given below (extract from the example commented in [U1.05.00]):

```
mail = LIRE_MAILLAGE ( )

mod1 = AFFE_MODELE(MAILLAGE = mail,
                  AFFE=_F(TOUT='OUI',
                          PHENOMENE='MECANIQUE',
                          MODELISATION='AXIS'))

f_y = DEFI_FONCTION ( NOM_PARA = 'Y'
                    VALE =_F ( 0., 20000.,
                               4., 0. )
                    )

charg = AFFE_CHAR_MECA_F ( MODELE = mod1
                          PRES_REP =_F ( GROUP_MA = ('lfa', 'ldf'),
                                          PRES = f_y ) )
.....
res1 = MECA_STATIQUE( MODELE=mod1,
                      .....
                      EXCIT=_F(CHARGE = charg),
                      .....)

res1 = CALC_ELEM ( reuse=res1, RESULTAT=res1,
                  .....
                  MODELE=mod1,
                  OPTION=('SIGM_ELNO_DEPL', 'EPSI_ELNO_DEPL'))
```

Let's examine key points from the previous example:

- Each command begins in the first column,
- The command operand and element lists are always in parenthesis,
- a `nom_de_concept` can only appear **once** as a concept product in the command text, left of the = sign,
- **multiple use of an existing concept as a concept product** is only possible for operators specified to that effect. When this functionality is used (re-entering concept), the command uses the reserved key word "reuse"

This operation:

- Overwrites the initial values. As an example, the LDL^T factorisation of a rigidity matrix is given below:
`matass = FACT_LDLT (reuse=matass, MATR_ASSE= matass)`
- Or
- Enriches the concept.

3.4 Overload rule

An overload rule, used particularly for assignment operations, has been added to the usage rules of a `mot_clé_facteur` with various lists of operands:

- Assignments are completed by adding up the effects of the various `mot_clé`,
- In case of conflict, the last `mot_clé` is used

Example: different materials MAT1, MAT2 and MAT3 have to be assigned to specific mesh elements:

```
mater = AFFE_MATERIAU ( MAILLAGE= mon_mail
                      AFFE =_F( TOUT = 'OUI', MATER = MAT1 ),
                      _F( GROUP_MA = 'mail2', MATER = MAT2 ),
```

```
_F( GROUP_MA = 'mail1', MATER = MAT3 ),  
_F( MAILLE = ( 'm7', 'm8' ), MATER = MAT3 ) )
```

- First, the material MAT1 is assigned to all mesh elements,
- Then, material MAT2 is assigned to the mesh element group mail2 which contains the mesh elements m8, m9 and m10.
- Finally, material MAT3 is assigned to the mesh element group mail1 (m5, m6 and m7) and to mesh elements m7 and m8, which is cause for conflict as mesh element m7 is already part of group mail1. The overload rule is thus applied and the following material field is obtained :

```
MAT1 : mesh elements m1 m2 m3 m4  
MAT2 : mesh elements m9 m10  
MAT3 : mesh elements m5 m6 m7 m8
```

3.5 Databases associated with a study

In *Code_Aster*, the management of all of the data structures associated with the manipulated concepts is based on the software JEVEUX. JEVEUX supports **memory allocation** set by the user during the execution request (**Memory** parameter expressed in MegaBytes). This space is usually insufficient to keep all of the data structures in the central memory. The software therefore manages the data exchanges between the central memory and the file's auxiliary memory.

When it is created by the code, each mesh entity is assigned a **direct access file**. Each of these files can be considered as a **database**, as at the end of the execution it contains the **repertoire** (names and attributes) which allows the user to exploit all segment values contained there.

Code_Aster uses several databases:

- the GLOBALE database which contains all of the concepts produced by the operators, as well as the contents of some catalogues upon which the concepts are based; the file associated to this database can be used for future research. It must thus be managed by the user.
- The other databases are used only by the Supervisor and the operators during the execution, and therefore do not require any user intervention.

A study is conducted by using a succession of several commands:

- Procedures, to exchange files with the outside world
- Operators, to create concepts produced during the modelling and calculation operations.

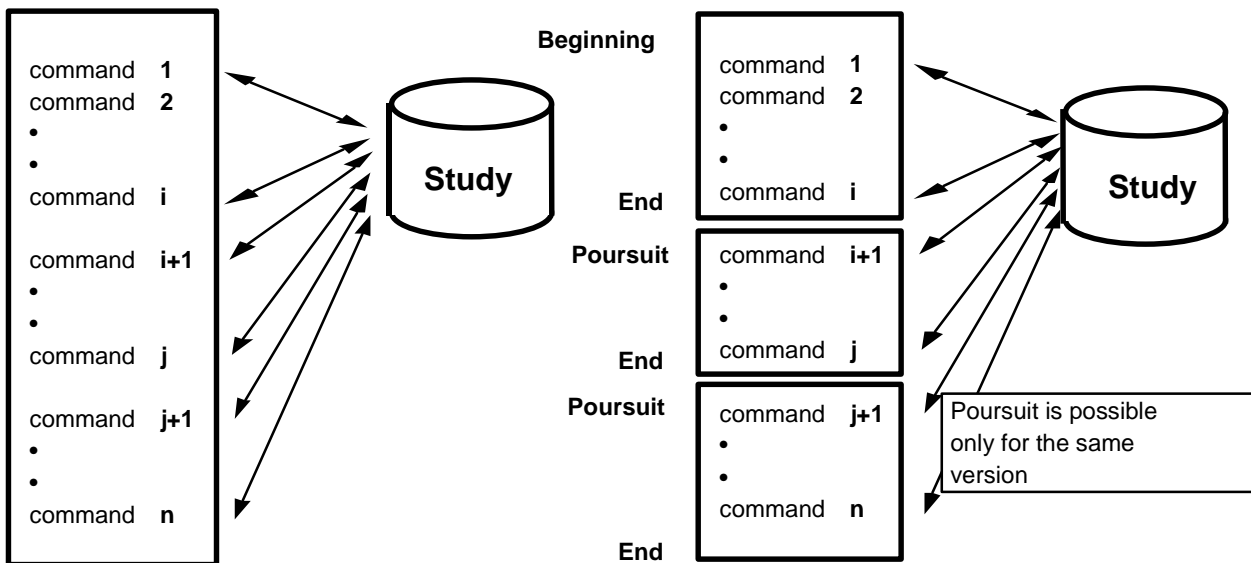
The commands which correspond to such a succession of operation can be produced in different ways from the executable module of *Code_Aster*:

- In one sequential execution, without user intervention,
- By fractioning the project into several successive executions and reusing past results. From the second execution onwards, the database is accessed in **pursuit** mode; When this is the case, the last command can be recalled if it had stopped prematurely (lack of time, incomplete or incorrect data detected during the execution phase, ...)

To manage these possibilities, three commands play an essential role. They correspond to the procedures which activate the supervisor:

- **DEBUT ()** **Mandatory** for the **first** execution of the conducted study,
- **POURSUIITE ()** **Mandatory** from the **second** execution of the conducted study,
- **FIN ()** **Mandatory** for all executions.

For a set study, a command file with the following structure can be submitted:

**Note:**

- The command `INCLUDE` allows the user to include in a command flow the contents of another command file. This enables the user to keep two main command files readable and to store cumbersome numerical data (ex: function definition) in annexed files.
- The command files can be divided into several files which will be executed one after the other; the database will be saved after each execution. To do this, successive command files bearing the suffix `.com1`, `.com2`, ..., `.com9` must be defined. The executions of files are consecutive. Only the database of the last successful execution is kept.

3.6 Value assignment help

3.6.1 Value substitution

Many different commands are available to help the user define the values used as arguments, and thus parameterise his command file:

- Give a name to one or several values:

```
nom = DEFI_VALEUR ( TYPE = [valeur] ) ;
```

 or simply:

```
nom = valeur
```
- Evaluate certain mathematical expressions:

```
EVAL (expression)
```

For example:

```
Eptub = 26.187E-3
Rmoy  = 203.2E-3
Rext  = DEFI_VALEUR ( R8 = EVAL ( "" Rmoy+(Eptub/2)"" ) )
cara  = AFFE_CARA_ELEM ( MODELE = modele
  POUTRE =_F ( GROUP_MA = tout , SECTION : 'CERCLE',
    CARA   = ( 'R','EP' ) , VALE = ( Rext , Eptub)) )
```

3.6.2 Each time the supervisor encounters the name chosen by the user, its value is updated. Functions of one or more parameters

It is usually necessary to use physical quantities depending on other parameters.

These can be:

- Defined on an external file read by the command `LIRE_FONCTION`.
- Defined in the command file by :
 - `DEFI_CONSTANTE` creates a concept fonction with one constant value,

- `DEFI_FONCTION` creates a concept `fonction` for a function associated to a real parameter,
- `DEFI_NAPPE` creates a concept `fonction` for a list of functions of same order, each element of the list corresponding to the value of another real parameter.

The concept created by these operators is of the type `fonction` and can only be used as an argument for operands which accept this type. Operators which accept an argument of the type `fonction` have a suffix F (ex: `AFFE_CHAR_MECA_F`). In that case, the functions are defined point by point with linear interpolation activated by default. They are thus linear in some parts.

The created functions are discrete tables of the order specified at creation. When searching the table, direct or table interpolation (linear or logarithmic) methods are used, depending on the specified characteristics. When the function is created, the user can request or prevent that the function be prolonged beyond the range of definition of the table using various rules.

- Defined by using their analytical operator `FORMULE` : for example :

```
omega = 3.566 ;
```

```
linst = ( 0., 0.01, 0.02, 0.03, 0.04, 0.05, 0.06, 0.07, 0.08, 0.09, 0.10,  
0.20, 0.40 )
```

```
F = FORMULE(REEL = ''(REEL:INST) = COS(OMEGA*INST)'' )  
F1=CALC_FONC_INTERP( FONCTION=F, VALE_R= linst,  
NOM_RESU='ACCE', )
```

The analytical function $F(t)=\cos(t)$ is then calculated by `CALC_FONC_INTERP` for the instants of the `linst`, list of `t` instants.

3.7 How to write a command file with EFICAS ?

The quickest way to write a command file for *Code_Aster* is to use an example already written by others. The database of tests of *Code_Aster* provides a good starting point for new models.

Better still, the tool **EFICAS** is an interactive and convivial way for users to write their command file. For each command, a list of possible key words is presented and all of the syntax is automatically verified. EFICAS also gives direct access to the user manual (fascicules [U4] et [U7]).

4 The main steps of a project

Usually, the main steps of a project can be defined as follows:

- Preparation of the work, which is completed after reading the mesh,
- Modelling, during which are assigned all of the characteristics of the finite elements and materials, as well as the characteristics of the boundary conditions and loading,
- The calculation can then be undertaken by using general solution methods [U4.5-]. These can eventually use calculation commands and vectors and matrix assembly commands[U4.6-]
- Post processing operations associated to the calculations [U4.8-],
- Printing results [U4.9-]
- Exchange of results with other software (for graphical visualisation for example) [U7.05-]

Another way of using *Code_Aster* consists in exploiting other industry specific tools. They are available in the code as `MACRO_COMMANDES` : a few of them are listed below :

- `ASCOUF` (fractured and uneven bend modelling),
- `ASPIC` (cracked and non-cracked piquing modelling),
- `GOUJ2ECH` (modelling of the behaviour of threaded assemblies).

4.1 Beginning the project and acquiring the mesh

We shall not cover the possible fragmentation of the command file, as it has already been presented in a previous paragraph.

The first executable command is:

```
DEBUT ( )
```

Arguments for this command are only necessary for maintenance or when working on substantially big projects.

There are two ways to read a mesh coming from an external mesh generator:

- Convert the text from the external software by using a separate execution. This enables the user to modify the file in a text editor and to save the file.

```
DEBUT ( )  
PRE_IDEAS ( )  
FIN ( )
```

A standard project could begin as follows:

```
DEBUT ( )  
ma = LIRE_MAILLAGE ( )
```

- Convert the file just before reading it

```
DEBUT ( )  
PRE_IDEAS ( )  
ma = LIRE_MAILLAGE ( )
```

4.2 Assigning modelling data to the weave

To create a model for a mechanical, thermal or acoustic problem, it is crucial to assign to the topologic entities of the mesh:

- A finite element model,
- Material properties (behaviour law and parameters for that law),
- Additional geometrical or mechanical characteristics,
- Boundary conditions or loading.

These assignments are done by using operators with the prefix AFPE_. The syntax and execution of these operators follow the rules mentioned previously in the paragraph describing how to use key words

4.2.1 Defining an assignment domain

To proceed with an assignment, it is vital to define an assignment domain. This domain must be based on the topologic entities of the mesh file. Five key words can be used to that effect, depending on the operator's specifications:

- Indicate the entire mesh with TOUT= 'OUI'
- Assigne to mesh elements with MAILLE= (list of mesh element names)
- Assign to mesh groups with GROUP_MA= (list of mesh group names)
- Assign to nodes with NEUD= (list of node names)
- Assign to node groups with GROUP_NO= (list of node group names)

4.2.2 Assign the finite element type

The mesh elements of the studied structure that are not yet topologic must have assigned to them:

- One or various phenomena to be studied 'MECANIQUE', 'THERMIQUE', 'ACOUSTIQUE';
- A finite element model compatible with the topologic description of the mesh. This assignment induces the creation of an explicit list of degrees of freedom for each node, and the creation of an interpolation law within the element.

Title : *Main operating principles of Code_Aster*
Author(s) : **J.M. PROIX, J.R. LEVESQUE**
Translator(s) : **C. LUZZATO**

Date : 26/06/03
Key : U1.03.00-D Page : 11/15

The operator `AFFE_MODELE` [U4.41.01] will be used for this. It can be called several times on the same weave and follows the rules of overload and remanence.

Note:

For projects where several phenomena are studied ('MECANIQUE', 'THERMIQUE'), it is essential to construct a model for each phenomenon, calling `AFFE_MODELE` as many times as needed. On the other hand, for a set calculation (mechanical, thermal, ...) only one model should be used.

See fascicule [U2-], and [U3-] for details about the characteristics of the available finite elements.

4.2.3 Assigning material properties

At this stage, it is necessary to assign material properties and associated parameters to each finite element of the model (except for discreet elements which are directly defined by a rigidity matrix, a mass matrix, and /or a dampening matrix). In other words, `DEFI_MATERIAU` is called to define a material and `AFFE_MATERIAU` is called to define a material field using mesh association. For a set calculation, only one material field should be used.

The validated characteristics from the material catalogue can also be used with the command `INCLUDE_MATERIAU` [U4.43.02].

A certain number of behaviour models can be used: elastic, orthotrope elastic, thermal, acoustic, elastoplastic, elastoviscoplastic, and damages. Note that several material characteristics can be defined for one material: elastic and thermal, elasto-plastic, thermo-plastic, ...

4.2.4 Assigning characteristics to elements

For a 'MECANIQUE' phenomenon, the geometric definition deduced from the mesh is not sufficient to completely describe certain types of elements.

The meshes must be assigned the following missing characteristics:

- For **hulls**: the constant width on each mesh element and a reference bound used to represent the state of constraint.
- For **beams, bars** and **pipes**: the characteristics of the transversal section, and eventually the orientation of the section around the neutral axis.

These operations can be accessed with the operator `AFFE_CARA_ELEM` [U4.42.01]). For ease of use, this command follows the rules of overload and propagation.

This operator carries another function: **matrices** of rigidity, mass or dampening can be inserted directly inside the POI1 meshes (or nodes) or SEG2 meshes. These matrices correspond to discreet finite element types with 3 or 6 degrees of freedom per node `DIS_T` or `DIS_TR`. They must be assigned when the `AFFE_MODELE` operator is called.

4.2.5 Assigning boundary conditions and loading

These operations are usually essential. They are carried out by several operators whose names begin with `AFFE_CHAR` or `CALC_CHAR`. Several operator calls can be made on one model during the study of the boundary conditions and loading.

The operators to be used depend on the studied phenomenon:

'MECANIQUE'	<code>AFFE_CHAR_CINE</code>	
	<code>AFFE_CHAR_MECA</code>	data of type reel (real) only
	<code>AFFE_CHAR_MECA_F</code>	data of type function (function) only
'THERMIQUE'	<code>AFFE_CHAR_THER</code>	data of type reel (real) only
	<code>AFFE_CHAR_THER_F</code>	data of type function (function) only
'ACOUSTIQUE'	<code>AFFE_CHAR_ACOU</code>	data of type reel (real) only

A seismic load response can be calculated with reference to its support point with the command `CALC_CHAR_SEISME`.

The boundary conditions and loading can be defined with respect to their nature:

- At the nodes,

Title : *Main operating principles of Code_Aster*
Author(s) : **J.M. PROIX, J.R. LEVESQUE**
Translator(s) : **C. LUZZATO**

Date : 26/06/03
Key : U1.03.00-D Page : 12/15

- On border mesh elements (edge or face) or finite element support mesh elements, created in the mesh file. The necessary type of finite element will have been assigned to the mesh by the `AFFE_MODELE` operator.

For a detailed description of the operands and the operators, and the orientation rules of the support mesh elements (general plane, local plane or other plane) see documents[U4.44-01], [U4.44-02], and [U4.44-04].

The boundary conditions can be treated in two ways:

- by « eliminating » the imposed degrees of freedom (for linear mechanical problems without kinematic boundary conditions (blocked degrees of freedom)) **with no linear relationships**. In that case the boundary condition will be defined by the command `AFFE_CHAR_CINE`.
- By dualisation [R3.03.01]. Because it is so general, this method can be applied to all types of boundary conditions (imposed degrees of freedom, linear relationships between degrees of freedom ...); the method used consists in adding 2 LAGRANGE multipliers for each imposed degree of freedom or each linear relation.

Each concept, which is created by calling the (`AFFE_CHAR`) operator types, corresponds to a system of inseparable boundary conditions and loading. In the calculation commands, the user can affiliate these concepts by giving to the operand a concept list of that type.

4.3 Undertaking calculations using general commands

4.3.1 THERMAL analysis

To calculate temperature fields corresponding to a linear or non linear thermal analysis of the following types:

- **Stationary** (instant 0),
- **Evolving**, for which the calculated instants are specified in a list of reals defined earlier on.

The commands to be used are:

- `THER_LINEAIRE` for a linear analysis [U4.54.01],
- `THER_NON_LINE` for a non linear analysis [U4.54.02],
- `THER_NON_LINE_MO` for a problem of mobile loads in constant regime [U4.54.03].

These operators perform the calculations of matrices, elementary vectors and assemblies necessary for the completion of the chosen resolution method.

4.3.2 STATICAL analysis

To calculate the mechanical evolution of a structure upon which is imposed a list of loads

- `MECA_STATIQUE` [U4.51.01]: linear behaviour, with superposition of the effects of each loading,
- `MACRO_ELAS_MULT` [U4.51.02]: linear behaviour, whilst distinguishing the effects of each loading,
- `STAT_NON_LINE` [U4.51.03]: near static evolution of a structure on which is imposed a series of loadings with negligible or important transformation. The structure is constructed in a material with linear or non linear behaviour, and can take into account contact and friction.

If the mechanical calculation corresponds to a **thermo-elasticity** project, an **instant** of the thermal calculation previously obtained will be used. If the material has been defined with **temperature dependant characteristics**, these are interpolated with the temperature corresponding to the required instant.

For problems of coupled thermo-hydro-mechanics, the operator `STAT_NON_LINE` simultaneously solves 3 problems: thermal, hydraulic and mechanic.

These operators perform the calculations of matrices, elementary vectors and assemblies necessary for the completion of the chosen resolution method.

4.3.3 MODAL analysis

To calculate specific modes and specific values of the structure (corresponding to a vibration or buckling problem):

- `MODE_ITER_SIMULT` [U4.52.03]: calculation of the specific modes by using simultaneous iterations, the specific values and vectors are real or complex.
- `MODE_ITER_INV` [U4.52.04]: calculation of the specific modes by using inversed iterations; the specific values and vectors are real or complex.
- `MACRO_MODE_MECA` [U4.52.02]: simplifies the modal analysis by automatically fragmenting the frequency interval into sub intervals,
- `MODE_ITER_CYCL` [U4.52.05]: specific mode calculation for a cyclically repeating structure from a database of specific modes containing reals.

These four operators require the prior calculation of assembled matrix. [U4.61-].

4.3.4 DYNAMICAL analysis

To calculate the linear or non linear dynamic response of the structure. Several operators are available. Some examples are given below:

`DYNA_LINE_TRAN` [U4.53.02]: temporal dynamic response of a linear structure enduring a transitional load.

`DYNA_LINE_HARM` [U4.53.02]: Complex dynamic response of linear structure enduring a harmonic load,

`DYNA_TRAN_MODAL` [U4.53.21]: transitional dynamic response with coordinates generalised by modal recombination.

These four operators require the prior calculation of assembled matrix. [U4.61-].

`DYNA_NON_LINE` [U4.53.01]: temporal dynamic response of a non linear structure enduring transitional load, which also calculates the assembled matrices.

4.4 The results

The results produced by operators which calculate using finite elements [U4.3-], [U4.4-] and [U4.5-] are usually of two types:

- Either of the type 'field' (either given by element or at the nodes) when the operators only produce one field (example `RESO_LDLT`),
- Or of the type `RESULTAT`. It regroups sets of fields accessible via a variable which distinguishes them (instant for time dependent calculation, frequency for a result created by an search for specific modes or harmonic responses, ...).

A `RESULTAT` type field in a concept can be accessed:

- Via an access variable which can be :
 - A simple number referring to the order in which the fields were stored,
 - A parameter defined with regard to the type of the `RESULTAT` concept:
 - Frequency or mode number for a `RESULTAT` of the type `mode_meca`,
 - An instant for a `RESULTAT` of the type `evol_elas`, `temper`, `dyna_trans` ou `evol_noli`.
- Using a symbolic field name referring to the type of field: displacement, speed, constraint state, general strain...

In addition to the access variables, other parameters can be attached to a concept of the type `RESULTAT`. The content of these concepts is thoroughly described in fascicule [U5-].

The different fields are incorporated in a `RESULTAT` concept :

- By the operator which created the concept, a general command (`MECA_STATIQUE`, `STAT_NON_LINE`, ...) or a simple command (`MODE_ITER_SIMULT`, `DYNA_LINE_TRAN`, ...),

Title : *Main operating principles of Code_Aster*
Author(s) : **J.M. PROIX, J.R. LEVESQUE**
Translator(s) : **C. LUZZATO**

Date : 26/06/03
Key : U1.03.00-D Page : 14/15

- By executing a command which enables the user to add a calculation option in the form of a field by element (`CALC_ELEM`) or of a field at the nodes (`CALC_NO`) ; we then explicitly say that we enrich the concept:

```
result = operateur (reuse=result, RESULTAT = result ...) ;
```

4.5 Exploiting the results

The previous commands have enabled the user to build different concepts which can be exploited through post- processing calculation operators:

- General post-processing operators (see fascicule [U4.81]), for example `CALC_ELEM`, `CALC_NO`, `POST_ELEM`, `POST_RELEVE_T`,
- Failure mechanics operators (see fascicule [U4.82]), for example `CALC_G_THETA`,
- Metallurgic operators : `CALC_META`,
- Statical mechanics post-processing (see fascicule [U4.83]), for example `POST_FATIGUE`, `POST_RCCM`.
- Dynamic mechanics post-processing (see fascicule [U4.84]), for example `POST_DYNA_ALEA`, `POST_DYNA_MODA_T`.
- Operators which extract concepts:
 - From a field in a `RESULTAT` concept: `RECU_CHAMP` [U4.63.1],
 - From a generalised coordinate field used for modal basis dynamics calculations: `RECU_GENE` [U4.63.2],
 - From a component's evolving function, and based on a `RESULTAT` concept: `RECU_FONCTION` [U4.63.3],
 - And resituate a dynamic response in the physical base `REST_BASE_PHYS`,
 - From a function or layer post-processing operator `CALC_FONCTION` which enable the user to search for peaks, extremums, linear combinations etc...[U4.21.9].

Finally, two procedures, `IMPR_RESU` [U4.91.01] and `IMPR_COURBE` [U4.33.01], allow the user to print the results and eventually create files exploitable in other software, specifically graphical visualisation software. Examples of graphical visualisation software are IDEAS, GMSH, or GIBI. These can be used regardless of the mesh file selected at the beginning.

5 Print files and error messages

Aster writes the information relative to the calculations in three file. The signification of each file is given below:

File	Information
ERREUR	Errors found when executing
MESSAGE	Information on the advancement of the calculation The command file and its interpretation by Aster. Time taken to execute each command. "system" messages
RESULTAT	The results requested written expressively by the user, as well as error messages.

Other files are used for the interface with the graphical analysis programs.

Different ERREUR (error) messages can be distinguished. Depending on their type, the error messages will appear in ERREUR, MESSAGE and or RESULTAT.

Message type	Output files
Fatal error message, appears after several error messages have been displayed. The concepts created during the execution are lost. This message is used when grave errors, which do not allow for other commands of <i>ASTER</i> to be executed, are detected.	ERREUR MESSAGE RESULTAT
Error message, the execution continues temporarily: this type of error enables the user to analyse a series of errors before the complete stop of the execution (the syntax analysis of the command file by the supervisor for example). <i>An <E> error message is always followed by an <F> error message.</i>	ERREUR MESSAGE RESULTAT
Error message, the concepts created during the execution are validated by the supervisor. The execution only stops once the GLOBALE database has been properly closed. This database can thus be used again in POURSUITE mode. This message is particularly effective to avoid system crashes during iterative calculations.	ERREUR MESSAGE RESULTAT
Alarm message. The number of alarm message is automatically limited to 5 successive and identical messages. To avoid error messages of type A, it is recommended that the user "repair" his command file.	MESSAGE RESULTAT
Informational message from the supervisor	MESSAGE