Management memory: JEVEUX

Summary:

This document presents the various functions of a called software library JEVEUX allowing to create, use, discharge on disc and to destroy named objects.

This library solves in particular the problem of the dynamic allocation (prohibited in FORTRAN 77) like that of the persistence of the objects on disc for a “resumption” of calculation.

This library is at the base of the structuring of the data in Code_Aster.

Let us announce that JEVEUX is not the only means of allocating memory, but any other dynamic allocation must be carried out with precaution and preferably with two macros as_deallocate and provisions as_allocate.

Contents

1 General presentation

1.1 Introduction

The management of the storage areas assigned to each structure of data constitutes a significant portion of the organization of an application of scientific calculation written in language FORTRAN.

Standard FORTRAN 77 proposes a very reduced number of possibilities of establishment of these structures of data:

- simple variable,
- fixed table of dimension,
- common zone of fixed size shared between several units of program.

These possibilities are insufficient to carry out a dynamic management of the resources.

In addition, for any application leading to quantities of data of a volume higher than that of the memory available on a given machine, the writer of the application must deal with the overflows report and program all the exchanges between the main memory and the auxiliary storage (files).

The software package JEVEUX allows to deal with, directly in the source text FORTRAN several levels of operations:

- standardisation of types FORTRAN usable on different machines (entirety, reality, complex, logical, character). The criteria of portability are those definite with software package ENVIMA (Manuel de Développement - Booklet [D6.01.01]: the descriptor of environment machine),
- the creation of structures of data items (objects JEVEUX) of size defined in the execution, shareable between several units of documented programs and car,
- the assumption of responsibility of all the transfers between main memory and auxiliary storage.

Note:

R1: This document presents the whole of the functions of the software package Jeveux *. A certain number of these functions does not have with being knownbe of the "average" programmer (those which should be called only by the supervisor, cf § 4.21 Routines of initialization used by the supervisor.). It also should be noticed that the majority of the uses of* Jeveux *in Aster the management of "simple" objects concerns (of the vectors). The reader in a hurry (or frightened by the whole of this document) will be able to be convinced that management of a vector by* Jeveux *is very simple by looking at the example of the §5.1 Creation and re-use of a vector of reality.*
R2: The developer of Code_Aster which uses Jeveux *must (besides this document have taken knowledge of another document: "Use of Jeveux"* [D2.06.01].

1.2 Objects JEVEUX : simple object and collection

1.2.1 The simple object

The simple object is the structure of basic information of the software package. A simple object is consisted by the descriptor and of only one segment of values. The number of attributes of a simple descriptor of object is fixed for a version of the software package.

Note:

The number of attributes is identical for all the simple objects.

```
Simple object = Name + Attributes + 1 Segment of values
```

The name of a simple object consists of 24 natures taken among the following:

- capital letters of With with Z, small letters of has with Z and figures from 0 to 9 ;
- four characters Spéciaux:

the white	" "
the point	". "
the underlined white	" _"
and commercial one	"&"

- the character Spécial $ is licit but nonaccessible to the user.

The principal attributes are the Suivants:

- the kind : described the structure of the simple object

E	simple element (variable)
V	vector (table with an index)
NR	repertoire of names (this kind will be defined in the following paragraph)

The kind makes it possible to associate the segment of values with the classical concepts of variable, table FORTRAN or to define a more complex structure of data.

- the type : defines type FORTRAN of the scalar variables of the object

I	entirety	(type FORTRAN INTEGER)
S	entirety	(type FORTRAN INTEGER*4)
R	reality	(type FORTRAN REAL*8)
C	complex	(type FORTRAN COMPLEX*16)
L	logic	(type FORTRAN LOGICAL)
K8, K16, K24, K32, or K80	characters	(type FORTRAN CHARACTER)

- the length : the length of the associated segment of values defines (the number of scalars).

1.2.2 The collection

Collection:

A collection is a structure of data making it possible to share certain attributes of the whole of the objects composing it. It authorizes indifferently the access by name or number to the objects of collection and to manage objects variable length.

Two types of collection are possible:

- one comprising a segment of values per object of collection: the dispersed collection,
- the other only one segment of values for all the objects of the collection: the contiguous collection.

Note:
A collection cannot be built a posteriori by regrouping of simple objects.

A collection is carried out starting from the descriptor of collection and one or more segments of values.

Descriptor of collection:

The descriptor of collection is the whole of information defining the collection: the name and attributes.

The attributes common to all the objects of the collection are in particular: the kind, the type, the length if it is constant, etc….

The attributes specific to each object of collection are managed separately. From the point of view of the user, an object of collection has all the attributes of a simple object, it could be used same manner.

```
Collection = Name + Attributes + 1 or several Segments of values
```

The name of a collection makes up in a way identical to that of a simple object.

The specific attributes of a collection are:

- the access:

who defines the access mode to the objects of the collection: the access can be carried out by name (the collection known as is then named), by number (the collection known as is numbered),

- storage:

who describes the organization in memory of the values associated with the objects with the collection, the values can be contiguous (only one segment of values), or dispersed (a segment of values per object of collection). All the objects of a contiguous collection are followed in the segment of values associated,

- the length:

it is constant if all the objects of the collection share the same length, it is variable if the objects are different lengths,

- the maximum number of objects of the collection.

Two attributes of collection can be divided between several collections. In the case of a collection of objects variable length, the vector lengths can be re-used for another collection containing the same number of objects. In the same way, two collections can share the names of the objects of collection. These objects are called pointer external.

Types of collection:

The numbered collection is made up of objects whose key of access is a variable entirety of 1 with the maximum number of objects of the collection. In a numbered collection all the objects preexist.
The named collection uses an ordinary person object of repertoire kind of names which contains the list of the names of objects of the collection managed by associative addressing. This repertoire of names can be defined by the user or automatically created. The names of the objects are inserted in the chronological order of creation of the names. One can reach an object of collection named by his name and/or the sequence number of insertion.

2 Management of the objects

2.1 Attributes

2.1.1 General information

The attributes to which the user has access are, with two exceptions, nonmodifiable directly.

They are affected during the creation of the descriptor (for example the standard) then fixed until destruction of this descriptor. The system manages besides the attributes which evolve during the application (for example the address memory of the segment of values).

One distinguishes three categories of attributes for the objects JEVEUX :

- generic attributes: class, kind, type,…
- attributes of collection.
- attributes of the objects of collection.

All the attributes are consultable constantly by the user. In the continuation of the document, for each attribute, one indicates:

- the reference symbol by which one can consult it,
- type FORTRAN of each attribute with following conventions:

Code	Type	Declaration FORTRAN
I	entirety	INTEGER
S	entirety	INTEGER*4
R	reality	REAL*8
C	complex	COMPLEX*16
L	logic	LOGICAL
Ki	character	CHARACTER*i
K*	chain	CHARACTER* (*)

2.1.2 Generic attributes

Attributes affected by the user (nonmodifiable after creation of the descriptor):

CLAS	K1	class of fastening of the object to a database
GENR	K1	kind of the object: - E simple variable - V vector, - NR repertoire of names of the type K8, K16 or K24.
TYPE	K1	type FORTRAN of the object: I, R, C, L or K
LTYP	I	length of the type: managed automatically for the types I, R, C and L, - standardized for the characters with values 8,16,24,32 and 80
length of the object: LONMAX NOMMAX	 I I	 component count of the object of kind V maximum number of names of the object of kind NR

Modifiable attributes constantly by the user:

These attributes are not essential to the operation of the software package. They make it possible to the user to supplement the description of the objects JEVEUX that it handles.

LONUTI	I	component count of the object of kind V used
DOCU	K4	four free characters left to the user

Attributes affected and managed by the system and consultable by the user:

These attributes are described as information, it can be consulted at the time of the clarification of an application (search for error,…).

NOMUTI	I	many names of the object of kind NR actually used
DATE	I	entirety comprising the month, the day, the year, the hour and minutes of the completion of the work having carried out the last modification of the segment of values (last unloading on the database)

IADM	I	address memory: relative position in memory JEVEUX of the segment of values and addresses pointer towards the zone dynamically allocated memory (stored in 2 entireties for each object)
IADD	I	address disc: contains the number of the recording of the segment of values in the database and the relative position in the recording (stored in 2 entireties for each object)
LONO	I	length in unit of addressing of the object (place occupied by the segment of values measured in length of the type)
USE	K16	use of an object JEVEUX : at the same time information contains on the use in writing or reading, the state (déchargeable, removable,…) segment of values when it is present in memory and the registered trademarks at the time of the first request in reading and writing

The latter are used only in-house with JEVEUX.

2.1.3 Attributes of collection

Common attributes affected by the user or managed by the system:

Common attributes (CLAS, GENR, TYPE, LTYP, DOCU, etc) with the objects of collection are accessible in the same way that the attributes from a simple object and answer the same rules.

Attribute affected and managed by the system and consultable by the user:

NUTIOC	I	many objects actually created in the collection

Note:

This attribute is updated only at the time of the call of the routine of creation of object of collection JECROC .

Attributes affected by the user and nonmodifiable:

ACCESS	K*	type of access: NAME or NUMBER (see the assignment of this attribute by JECREC)
STORAGE	K*	Mode of storage of the values: - CONTIG : all the objects are contiguous in the segment of values - DISPERSE : the objects are filed as needs

MODELONG	K*	Mode of definition length of the objects of collection: - CONSTANT : all the objects are of the same length (attribute length of the object of reference) - VARIABLE : each object can have a different length
LONT	I	overall length of a contiguous collection
NMAXOC	I	maximum number of objects of the collection

2.1.4 Attributes of the objects of collection

The rules of name of attribute of the objects of collection are the same ones as for the simple objects.

2.2 Associated databases

With each class it is possible to associate a database on disc. This possibility is by no means obligatory: one can work without writing and reading on disc (what can result in saturating the memory available and to the stop with the application).

The number of classes on which one can work simultaneously is limited to 5. It is possible to constantly open or close a class during the application.

The opening of a preexistent database makes it possible to recover the whole of the information stored on the latter. At the end of the work, it is possible to destroy a whole class.

A database is a file of direct access; it is defined by:

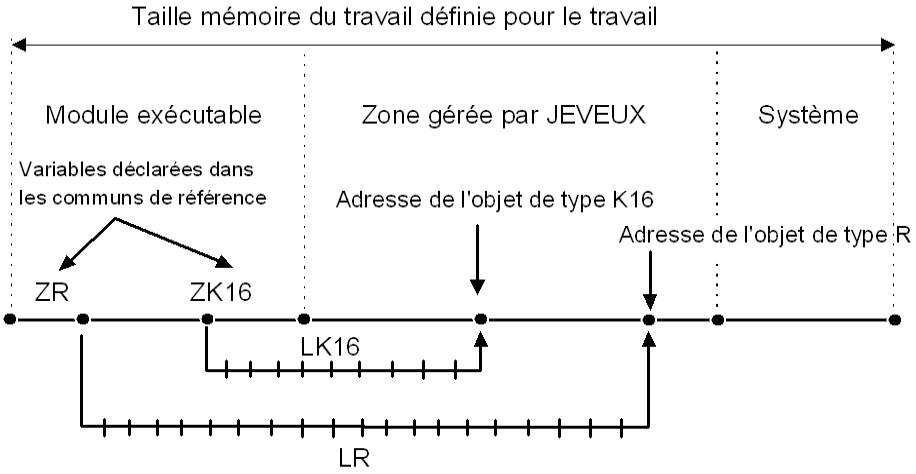
- a name of class (K1),
- a local name of the file (K8),
- a block length,
- a number of blocks.

Note:
In Code_Aster, all the classes are associated with bases cf § 3.3 Environment of Code_Aster.

2.3 The access to the values

The user reaches the values of an object JEVEUX by a relative address in a table defined by one of variables FORTRAN of reference of the suitable type. These variables of reference (ZI, ZR,…) are placed in commun runs standardized confer [D2.06.01]: Use Jeveux “Release of the segments of values and concept of mark”.

Note:
The returned address remains valid as long as the user did not declare that it did not use any more the object.



- ZR (LR) 1 era component of the values of the object of the real type
- ZK16 (LK16 - 1 + I) I ème component of the values of the object K16

2.4 Concepts of management of the memory

The access to the objects JEVEUX be carried out by name. These names are stored in a simple object of repertoire kind of names accessible only by the software package and managed by a method of associative addressing.

To order an application, the user must gather the objects JEVEUX in one or more as a preliminary open classes. With each class a catalogue is associated: it is the unit made up of the repertoire of names and the attributes of the objects JEVEUX class. The class one of the attributes is defined in the creation of the descriptor of an object JEVEUX. The scan for the noun is carried out among all the classes open to this moment

Note:
A name cannot thus appear more once among the unit of the open classes.

With each class, it is possible to associate a file of direct access (or database), which will contain at the end of the application, the descriptors and the segments of values of all the objects JEVEUX class. This kind of file gives access quickly the various recordings, an index describing the position of each one of them.

The exchanges between the main memory and the databases associated with the classes are entirely dealt with by the software package. When the main memory is saturated (that it is not possible any more to allocate a zone dynamically), this one discharges or destroyed the segments from values declared unutilised. At the end of the application, all the objects JEVEUX present in memory are discharged in the associated database, as well as the catalogues (simple objects “system”), which allows the later re-uses.

The file of direct access will be valid only if the index were brought up to date during closing, it is thus essential to properly stop the application via the routine JEFINI. Any segment of values in memory is framed by eight entireties (four front, four behind) making it possible to manage the segment of values, to indicate its use (free, used in reading, déchargeable,….), to store the associated identifier, and to ensure a partial protection the overflows (the software package controls the integrity of the values contained by these entireties at the time of any request).

2.5 Requests of use of the objects

The software package JEVEUX provides various subroutines and functions allowing to manage all the objects. The various following requests are distinguished:

- the creation of the descriptor of an object JEVEUX or of an object of collection,
- the allowance of the object: search for place in memory for the segment of values associated, initialization of the values or second reading on the database, finally supply with the appealing unit of program of a relative address of the segment of values compared to a variable of reference Z* . This allowance can be formulated in reading only or read/write;
- consultation of the descriptor which makes it possible to dynamically recover the value of an attribute;
- impressions of the descriptor, the segment of values, the catalogue or the state of the memory managed by JEVEUX ;
- the release which puts an end to the allowance of the object and returns the segment of values unutilised;
- destruction of the descriptor and the segment of values of an object JEVEUX or of an object of collection.

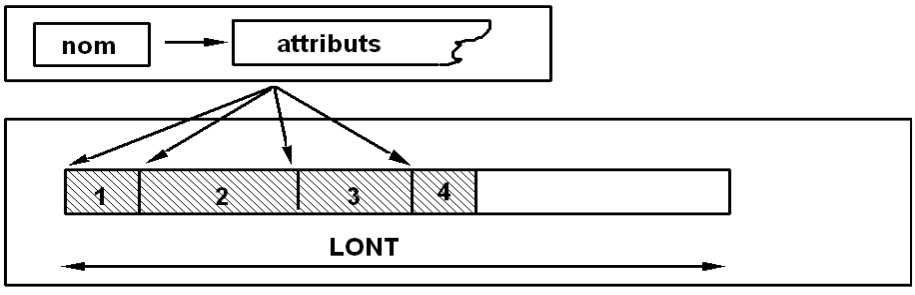
2.6 Characteristics of the collections

The dimensioning of a collection depends on its type. For each type, one describes Ci below the mode of definition of the segment (S) of values associated.

2.6.1 Contiguous collection variable length

One defines:

- that is to say the overall length of the segment of values by the attribute LONT (overall length) of the collection. One can in this case create the collection and define (without particular order) later on the length of each object before using it;
- that is to say the length of whole or part of the objects of the collection by giving the vector lengths managed by the user, or by bringing up to date the attribute length for each object, the overall length of the segment of values will be calculated by the software package.



collection contiguë de longueur variable

Note:

It is advised to define the length of all the objects of a contiguous collection before the first request of allowance in memory,
The length of the segment of values LONT is fixed at the time of the first request of allowance in memory,
A contiguous collection variable length cannot thus be increased after the first access,
all the objects created in a contiguous collection are managed together,
The associated segment of values can be used like a vector of values by being unaware of cutting into object of collection.

2.6.2 Contiguous collection constant length

One defines the length common to all the objects of collection; the length of the segment of values will be equal to the product constant length by the maximum number of objects of the collection.

Caution: when it is done JEECRA LONMAX collection, it is necessary to indicate the length of only one element of the collection and not the overall length.

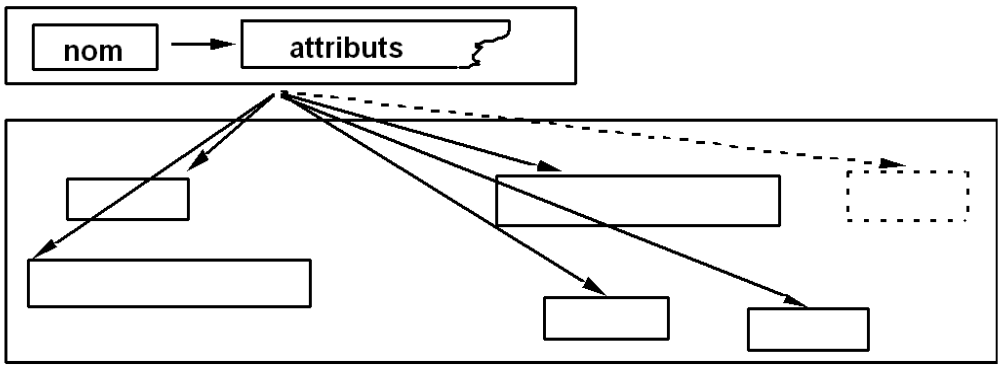
2.6.3 Dispersed collection variable length

One defines:

- raising amongst objects,
- the length of each object, as needs and without particular order, before the allowance of the object.

Note:

The obstruction manpower of this kind of collection is limited to the cumulated length of the objects actually created.



collection dispersée de longueur variable

Note:

For a dispersed collection, one can use an object (of which the length was defined) before to have finished defining the whole of the objects.

2.6.4 Dispersed collection constant length

One defines:

- raising amongst objects,
- the length common to all the objects.

2.6.5 Basic concepts

Object JEVEUX :

The software package JEVEUX allows to manage two types of structures of data accessible per name within the application. The use wanted that one calls these structures of data objects JEVEUX. By definition, an object JEVEUX is:

- that is to say a simple object,
- that is to say a collection of objects.

An object JEVEUX is the unit consisted a descriptor and one or more segment (S) of values.

Descriptor:

A descriptor is a set of information made up of a name and various attributes. The descriptor makes it possible to reach, starting from the name, (X) the segment (S) of values of the object JEVEUX. The attributes describe the structure of the object JEVEUX.

Name:

A name consists of a succession of alphanumerics limited to 24 characters (with the name is associated, by a table and a function of coding, an entirety giving access the object by associative addressing).

Attribute:

The attributes are the parameters (of fixed number for the simple objects) making it possible to define and describe the structure of data (variable, table). For example the type of the values and the length of the table are part of the whole of the attributes.

Segment of values:

A segment of values is a continuous succession of words or bytes (according to the unit of addressing of the machine) used to store values in main memory or auxiliary storage (file).

Note:

Except exception (repertoires of names), a segment of values is a standard succession of values in the same way.

2.7 Release of the segments of values

A request of access in reading or writing on a segment of values associated with an object JEVEUX cause a loading in memory of the contents of the associated segment of values. When a finished memory capacity is managed, it arrives one moment when it is not possible any more to find of place to charge a new object. It is then necessary to cause unloadings on disc or to eliminate from the segments of values become useless. This mechanism cannot be completely taken charges some by JEVEUX : the programmer must have indicated the objects concerned as a preliminary. But some care must be observed: several units of program can use simultaneously the same address memory associated with an object. The setting in memory of a segment of values is accompanied by the assignment of a whole mark which measures the level of depth in the subprogram calls for each new request. The release can be carried out only if the level of call to this moment is identical to the mark associated with the segment with values. One calls current mark the level of call in progress. The placement of this mechanism imposes a rule of use of JEVEUX very strict: any routine which calls JEVEUO or WKVECT must carry out a call to JEMARQ like first achievable instruction and a call to JEDEMA like last achievable instruction. The routine JEMARQ allows to bring up to date the value of the current mark, by incrementing it of 1, assigned to all the segments of values charged later on. The routine JEDEMA release all the segments of values associated with the current mark then décrémente the latter.
Certain configurations tolerate exceptions to this rule: they are for example loops on blocks of matrices. It is then necessary progressively to release the objects, it is the routine JELIBE who is used in this case.

It is sometimes necessary to have permanently or all the length of a ordering of Code_Aster of certain objects (for the Supervisor of execution, the coded material), of the specific requests are used, which affect a special mark.

The value of the current mark can be consulted at any moment using the routine JEVEMA.

3 Environment of application

3.1 Print files

Impression of the descriptors, the segments of values, the catalogue, the state of the memory managed by JEVEUX and of the error messages is directed towards definite logical units at the beginning of application (cf §3.3 Environment of Code_Aster).

3.2 Assistance with clarification

The errors met and detected by the software package cause the impression of an error message, and in certain cases the databases opened by the application close. The software package then prints out of the memory which it manages and catalogues of all the open classes.

Note:

JEVEUX use its own routines of impression of messages different from those of Code_Aster to avoid calls dynamically recursive.

All objects JEVEUX, saved as a preliminary, are recoverable in a later execution.

3.3 Environment of Code_Aster

Two databases are usable within the application of Code_Aster:

base 'TOTAL'	associated with the class 'G'
base 'BIRD'	associated with the class 'V'

The real size of the file is dynamic; it depends on the volume of information to store indeed. Cette size is limited by the conditions of operating and a parameter preset among the values characterizing the platform. On the platform of Linux reference 64 bits, the size initial at 48 Go is fixed. This value is used to dimension 2 objects used by the manager of memory, it will be modified automatically in the course of execution if need be. It is possible of modifier this value while passing an argument on the command line of achievable behind the keyword "- max_base size" where size is an actual value measured out of Mo.
On the platforms 32 bits, the size initial at 2,047 Go (2 147,483,647) is fixed, but the code manages several files to go beyond this limit when the parameter "- max_base" is modified.
For the Total base, which can be saved and re-used in data of a calculation, maximum size initial in "CONTINUATION" is preserved such as it is if the parameter "- max_base" is not used, but perhaps redefined with the need for this manner.
automatic redimensioning is implemented at the time of the writing on the associated files, on the basis of criterion of filling fixed at 50 % amongst recordings, the utility JJAGOD is called for redimensionner the systems objects $$USADI and $$ACCE which depend directly on the maximum size of the files used.

Note:

A third base of name 'BASEELEM' is used to store and read again the compiled catalogue of the elements.

The impressions of error messages are carried out by the utilities of impression of message with following conventions:

- programming error (misuse of JEVEUX) : message of class 'S' with dead halt;
- error of exploitation (access to a non-existent object,…) : message of class 'F' with attempt at closure open databases.

The utilities of impression of error messages of Code_Aster can sometimes be employed in order to communicate with the Supervisor and to stop the code properly by validating the concepts created. Names of print units indicated at the time of the call of the routines of impression (JEIMPO, JEIMPR,…) are the following:

- RESULT : results of calculation,
- MESSAGE : messages of control and error messages.

4 Provided subroutines and use

4.1 Preliminaries

In this chapter, one indicates by:

```
nom_os    a simple name of object
nom_co    a name of collection
nom_oc    a name of object of collection
num_oc    a number of object of collection
```

During the description of the arguments of the routines, one specifies the type of each argument and if it must be provided as starter (in) or recovered in exit (out). The modified arguments are announced by (VAr).

4.2 Functions of access for the objects of collection

The access to the objects JEVEUX and with the objects of collection is carried out by name. In the case of a named collection, it is necessary to pass, like argument, one of the two groups of information:

- the name of the collection and the name of the object of collection,
- the name of collection and the sequence number of the object of collection in the repertoire.

To limit the number of routines user and to standardize the arguments of call, one introduced functions of the type FORTRAN CHARACTER *32. Those return a character string made up of the name of collection and an interpretable suffix by the software package. They affect a commun run which ensures the transfer towards the routine called of the name of object of collection, of the sequence number in the repertoire. In order to synchronize well the call with the name of collection and the assignment of the commun run, it is obligatory to invite these functions in the arguments of the routines concerned.

Note:
These functions should be used only at the time of the call of a routine of request on an object.

Functions of the type CHARACTER*32 (to be declared in any subroutine of call):

The three following functions must be only called like argument of the routines acting on objects of collection

CHARACTER*32 FUNCTION JEXNOM (name, nom_oc)

in	name	K2	name of repertoire or collection
in	nom_oc	K*	name of object of collection.

CHARACTER*32 FUNCTION JEXNUM (name, num_oc)

in	name	K24	name of repertoire or collection
in	num_ocv	I	number of object of collection.

Example of use:

```
CAL JEVEUO (JEXNUM (NOMCO, NUMO), 'L', vx=TABL)
```

In the case of a contiguous collection, a function of the type FORTRAN CHARACTER*32 is used to reach the vector cumulated lengths.
This vector of entireties contains $n+1$ values for a collection of n objects: the component V_i this vector provides the relative address of the object i in the segment of values of the collection; the length of this object is obtained by the difference $V_{i+1}-V_i$.

CHARACTER*32 FUNCTION JEXATR (nom_co, arg)

in	nom_co	K24	name of collection
in	arg	K8	'LONCUM' 'LONUTI', 'LONMAX'

In the same way, the argument LONUTI, respectively LONMAX, gives access the vector lengths used, respectively maximum, by a simple request of the type:
CAL JEVEUO (JEXATR (nom_co, 'LONUTI'), 'It, vi=LUTI).
Notation: one will indicate, from now, a name of object JEVEUX, or a call to the one of these functions within the arguments of a routine by nom_o.

4.3 The creation of the descriptors

The descriptor of a simple object or a collection is created in general by several calls. The first call makes it possible to create the descriptor by defining the name of the object JEVEUX, and by indicating the values of the obligatory attributes (nonmodifiable later on):

- the class of fastening,
- the kind,
- the type.

In the case of a collection, one must define moreover:

- the access,
- storage,
- mode of definition length,
- the maximum number of objects.

4.3.1 Creation of the simple descriptor of object

SUBROUTINE JECREO (nom_os, lis_at)

in	nom_os	K24	simple name of object.

in	lis_at	K*	defining text STANDARD CLAS GENR and if need be LTYP, when it is provided, the length of the type must be stuck to the type.

For example: 'G V K16'. The first character must be not white, at least a white separates each value D' attribute; the capital letters are obligatory.

4.3.2 Creation of the descriptor of collection

SUBROUTINE JECREC (nom_co, lis_at, access, stock, modlon, nmaxoc)

in	nom_co	K24	name of collection
in	lis_at	K*	defining text STANDARD CLAS GENR and if need be LTYP, when it is provided, the length of the type must be stuck to the type.
in	access	K*	'NO': collection named with internal repertoire (names of 8 characters to the maximum), objects of collection created by name and accessible then by name or number. 'NAKED': sequentially numbered collection, the objects of collection are accessible only by number.
in	stock	K*	mode of storage of the collection: 'CONTIG' contiguous objects of collection in the segment of values, 'DISPERSE' objects of collection independent from/to each other.
in	modlon	K*	mode of definition length of the objects of collection 'CONSTANT': all the objects of the collection are length identical; 'VARIABLE': each object of the collection can have a different length.
in	nmaxoc	I	maximum number of objects of the collection.

Note:

It is not authorized any more to create collections being based on a repertoire of names created in addition and being able to be divided.
It is not authorized any more to create collections being pressed on a vector length created in addition and being able to be divided.

4.3.3 Assignment of an attribute (after creation of the descriptor)

It is often necessary to supplement the assignment of the attributes of the descriptor (definition length of a vector, overall length of a collection,…).

SUBROUTINE JEECRA (nom_o, nom_at, ival=ival, cval=cval)

in	nom_o	K24	name of object JEVEUX
in	nom_at	K*	name of attribute (cf appendix 2: list of the attributes).
in	ival	I	whole value for an attribute of the whole type.
in	cval	K*	text for an attribute of the type CHARACTER.

Note:

Two arguments ival and cval are optional, cval must be different from the null string. The two arguments can be provided at the time of the call to the routine with or without the respective identifier "ival=" and "cval=".

```
CAL JEECRA (nom_o, 'DOCU', cval= 'JPJP')
CAL JEECRA (nom_o, 'LONMAX', ival=500)
```

4.4 Insertion of a name in a repertoire or creation of an object of collection

It is the same routine which is used to create a name of object of collection and to insert a name in a simple object of repertoire kind.

SUBROUTINE JECROC (JEXNOM (nom_o, name))

in	nom_o	K24	simple name of object of repertoire kind, or name of named collection
in	name	K*	name of object to be inserted in the repertoire, or name of object of collection

SUBROUTINE JECROC (JEXNUM (nom_co, num_oc))

in	nom_co	K24	name of collection
in	num_oc	I	number of object of collection

Note:

For a numbered collection, this call brings up to date the attribute NUTIOC.

4.5 The request of allowance

It is via the routine JEVEUO that the user recovers a pointer on the segment of values. Alternatively, the routine JEVEUO also allows to recover a relative address compared to a variable Z* measured in the length of the type of the object JEVEUX.
That then enables him to use the associated segment of values. At the time of this call if the segment of values is not present in memory, the software package carries out a dynamic allocation. If the object JEVEUX does not have image on disc, the segment of values is initialized according to the type of the object (zero or white). In the contrary case one recovers on disc the preceding values. One calls use the access term ('E' or 'L') with the segment of values. The segment of values remains allocated with the affected use as long as the user did not carry out of new request and as long as the call to JEDEMA on the good level was not carried out. A request in writing on an object allocated in reading will modify the use of it. A request in reading on an object allocated in writing will not affect the use.

SUBROUTINE JEVEUO (nom_o, concealment, jtab, VI, vr,…, vk80)

in	nom_o	K24	access to the segment of values defined by: nom_os : simple object, nom_co : contiguous collection, JEXNOM (nom_co, nom_oc) : object of named collection, JEXNUM (nom_co, num_oc) : object of numbered collection, JEXATR (nom_co, 'LONCUM') vector cumulated lengths of contiguous collection.
in	concealment	K*1	access term or use of the segment of values, 'E' in read/write (allows to modify the segment of values), 'L' in reading (not of modification of the segment of values).
out	jtab	I	address of the first value of the object in table FORTRAN associated with the variable Z* corresponding to the type of the object
out	VI	I	Table of entireties
out	vr	R	Table of realities
…			
out	vk80	K80	Table of character*80

The arguments all of exit are of the optional arguments and they are excluded mutually.
One recommends to use the arguments preferentially: vl, VI, vi4, vr, vc, vk8, vk16, vk32 and vk80. Each argument is adapted to the type of the object: LOGICAL, INTEGER,…
The argument jtab returns a "address" in a table put in COMMON : ZI, ZR,…. Its use makes in general the programming less readable.

Note:
The request JEVEUO is prohibited on the objects of repertoire kind.

4.5.1 Use of the arguments VI, vr,…, vk80

Example:

```
character (len=24):: obj_coor
real (kind=8), to point:: coordo (:) => no one ()

cal jeveuo (obj_coor, 'It, vr=coordo)
! kth value of the vector coordo:
x=coordo (K)
```

4.5.2 Use of the argument jtab

Example:

```
character (len=24):: obj_coor
integer:: jcoor

cal jeveuo (obj_coor, 'It, jcoor)
! kth value of the vector coordo:
x=zr (jcoor-1+k)
```

COMMON of reference

The request of allowance JEVEUO with the argument of exit jtab turn over the "address" of the object in a table put in COMMON : ZI, ZR,….

The correspondence enters type FORTRAN of the object and it COMMON of reference which corresponds to him is given by the following table:

type FORTRAN	COMMON	name of the variable
I	IVARJE	ZI
S	I4VAJE	ZI4
R	RVARJE	ZR
C	CVARJE	ZC
L	LVARJE	ZL
K8	KVARJE	ZK8
K16	KVARJE	ZK16
K24	KVARJE	ZK24
K32	KVARJE	ZK32
K80	KVARJE	ZK80

The block of declarations below must thus be inserted in any routine wanting to read or write in an object jeveux :

```
C --------- BEGINNING DECLARATIONS NORMALISEES  JEVEUX --------------------
      INTEGER            ZI
      COMMON  / IVARJE/ZI (1)
      INTEGER*4          ZI4
      COMMON  / I4VAJE/ZI 4(1)
      REAL*8 ZR
      COMMON/RVARJE/ZR (1)
      COMPLEX*16 ZC
      COMMON/CVARJE/ZC (1)
      LOGICAL ZL
      COMMON/LVARJE/ZL (1)
      CHARACTER*8        ZK8
      CHARACTER*16               ZK16
      CHARACTER*24                        ZK24
      CHARACTER*32                                ZK32
      CHARACTER*80                                       ZK80
      COMMON  / KVARJE/ZK 8(1), ZK 16(1), ZK 24(1), ZK 32(1), ZK 80(1)
```

```
C --------- END  DECLARATIONS  NORMALISEES  JEVEUX --------------------
```

4.6 The request of permanent allowance during an order

JEVEUT allows to allocate objects JEVEUX in their affecting a mark different from the current mark being worth -1. The objects will not be affected by the calls to JEDEMA but will have to be released explicitly by a call to JELIBZ.

The syntax of call is identical to that of JEVEUO.

4.7 The request of permanent allowance during the execution

This request is exclusively reserved to the Supervisor. JEVEUS allows to allocate objects JEVEUX in their affecting a mark different from the current mark being worth -3. They will be released only at the end of the execution.

The syntax of call is identical to that of JEVEUO.

4.8 Surcouche of creation and allowance of an object of vector kind

WKVECT is a surcouche JEVEUX making it possible to allocate a vector (creation and presence in memory), it allows "the economy" of the call to three subroutines JEVEUX.

SUBROUTINE WKVECT (nom_os, lis_at, length, jtab, VI, vr,…, vk80)

in	nom_os	K24	simple name of object
in	lis_at	K*	defining text STANDARD CLAS GENR LTYP, GENR is worth here obligatorily V.
in	length	I	whole value associated with the attribute LONMAX, length of the vector.
out	jtab	I	address of the first value of the object in table FORTRAN associated with the variable Z* corresponding to the type of the vector.
out	VI	I	Table of entireties
out	vr	R	Table of realities
…			
out	vk80	K80	Table of character*80

The allowance of the object is carried out by default as a first writer (cel=' E').

Note:

Arguments of exit: jtab, VI, vr,…, vk80 are excluded mutually.
No object of name nom_os does not have to exist in the bases (under penalty of program stop).

Example of use:

Allowance on the basis BIRD (V) of a vector of work (V) of REAL*8 (R) of length 100; the name of the table is &&OP000.TAMPON.

```
real (kind=8), to point:: VTRAV => no one ()

CAL WKVECT ('&&OP000.TAMPON', 'V V R', 100, vr=VTRAV)
```

When it is not necessary to have a named access, to allocate a vector of work used within a routine, it is possible to call on the macros **as_allocate** and **as_deallocate**.

4.9 Alternative to the allowance of objectS of work

Although it is possible to allocate objects of work using JEVEUX within a routine, this use is not always relevant. If the allocated zones are used only temporarily, or of size not requiring unloading on disc, it is more advantageous to call on the function **as_allocate** and **as_deallocate** . These last coat the standard functions and make it possible to invite the mechanisms of unloading of objects JEVEUX in order to release from the memory. They at any moment communicate the volume of memory allocated or released, all the dynamic allocations in FORTRAN must imperatively pass by this mechanism. These utilities (macro order) make it possible to allocate and of désallouer a vector of the various types managed by JEVEUX. These two macros use optional arguments.

AS_ALLOCATE (size=size, vl=vl, vi=vi, vi4=vi4, vr=vr, vc=vc, vk8=vk8, vk16=v16, vk24=vk24, vk32=vk32, vk80=vk80)

in	size	I	length of the vector to be allocated
inout	vx	L, I, I4, C, Kx	vector in the desired type

This macro makes it possible to allocate a vector, who must be declared below under the model. The various arguments all are optional, it is of course necessary to provide the length of the vector.

```
integer, to point:: (:) => will tab_para no one ()
AS_ALLOCATE (will vi=tab_para, size=1958)
```

AS_DEALLOCATE (vl=vl, vi=vi, vi4=vi4, vr=vr, vc=vc, vk8=vk8, vk16=v16, vk24=vk24, vk32=vk32, vk80=vk80)

in	vx	L, I, I4, C, Kx	vector in the desired type

This macro makes it possible to release the vector whose name passed in argument and to restore the memory while informing manager JEVEUX.

4.10 Surcouche of enlarging of an object of vector kind

SUBROUTINE JUVECA (nom_os, length)

in	nom_os	K24	simple name of object
in	length	I	new value associated with the attribute LONMAX, length of the vector.

This routine makes it possible to modify the size of a simple object of vector kind. The new vector is affected with the same class as the old one.

Note:

The object must be in memory (call to JEVEUO precondition),
The attribute LONUTI is affected with the same value as LONMAX ,
The values are recopied,

4.11 Recopies of objects JEVEUX

4.11.1 Recopy of an object JEVEUX (simple object, collection or object of collection):

SUBROUTINE JEDUPO (nom_in, nom_clo, nom_ou, dup_co)

in	nom_in	K24	name of the object JEVEUX to recopy
in	nom_clo	K1	name of the class of the receptacle object (≠ ' ')
in	nom_ou	K24	
in	dup_co	L	used only for the collections if the classes are different, if =.TRUE. : one recopies the external pointers with a collection, if =.FALSE. : the external pointers are preserved

Note:

The receptacle object is destroyed if there exists before.

4.11.2 Recopy of a set of objects JEVEUX :

Instead of working on only one object, it is possible to provide to certain routines a character string being used to select the objects JEVEUX whose name contains this chain.

SUBROUTINE JEDUPC (nom_cli, souchi, ipos, nom_clo, soucho, dup_co)

in	nom_cli	K1	name of the class of the objects to be recopied
in	souchi	K*	character string to be identified in the names of objects
in	ipos	I	position in the names of the chain to be identified
in	nom_clo	K1	name of the class of the receptacles objects
in	soucho	K*	chain to be substituted in names origins to obtain the name of the object JEVEUX receptacle
in	dup_co	L	used only for the collections if the classes are different, if =.TRUE. : one recopies the external pointers with a collection, if =.FALSE. : the external pointers are preserved

4.12 Requests of release

These requests lead to a writing on disc (differed or not) according to the access term chosen at the time of the request of allowance and the rules of release [§2.3]:

One calls release the stop of the request in progress on an object JEVEUX :

- end of a writing, with writing differed on disc,
- end of reading, which does not imply any writing on disc.

The releases are carried out at the time of the call to JEDEMA. Calls to the routine JELIBE are in general prohibited.

Note:

The access in reading makes it possible to avoid the writings on disc at the end of the use (time-saver).

Management of the access:

Access authorizations affected at the time of the first call to JEVEUO in reading or writing are managed using the following routine:

SUBROUTINE JEMARQ ()

JEMARQ increment the value of the current mark. Its call is obligatory in any unit of program charging in memory with the objects JEVEUX.

Implicit release of all the objects in charge in a unit of programming

SUBROUTINE JEDEMA ()

JEDEMA release all the affected objects of the current mark and décrémente the value of the current mark. Its call is obligatory in any unit of program charging in memory with the objects JEVEUX.

Release of an object JEVEUX or of an object of dispersed collection:

SUBROUTINE JELIBE (nom_o)

in	nom_o	K24	name of object JEVEUX (simple object, collection or object of dispersed collection)

Its call is tolerated under certain conditions.

Release of the whole of objects JEVEUX :

SUBROUTINE JELIBZ ()

The call to this routine causes the release of the whole of the objects JEVEUX who were charged in memory by a call to JEVEUT (they are affected of a mark being worth -1).

4.13 Requests of existence

The requests of existence make it possible to check the existence of the descriptor of an object JEVEUX or of an object of collection, but also the presence of a name in a repertoire. They also make it possible to recover the sequence number of a name in a repertoire of names.

Existence of an object JEVEUX or of an object of collection:

SUBROUTINE JEEXIN (nom_o, iret)

in	nom_o	K24	name of object JEVEUX
out	iret	I	code return of the routine $iret=0$ the descriptor of the object does not exist, $iret\neq 0$ the descriptor exists

4.14 The passage of the sequence number to the name and vice versa

Obtaining the sequence number starting from the name:

S**UBROUTINE JENONU (JEXNOM (nom_o, name), num)**

in	nom_o	K24	name of collection or simple object of repertoire kind
in	name	K*	name of object of collection or name
out	num	I	num = number of the object corresponding to the name name

Note:
If the sought name does not appear in the repertoire, the returned number is 0.
This call is useless in the case of a numbered collection.

Obtaining the name starting from the sequence number:

SUBROUTINE JENUNO (JEXNUM (nom_o, num), name)

in	nom_o	K24	name of collection or simple object of repertoire kind
in	num	I	sequence number of insertion
out	name	K	name = name of the object corresponding to the number num

Note:
If the sought number does not appear in the repertoire, the returned name is white.
This call is impossible in the case of a numbered collection.
If the sought number is higher than the stored number of numbered objects, the routine stops in error.

4.15 Destruction of the descriptors

It is possible to destroy the descriptor of an object JEVEUX and by extension the name and attributes of an object of collection. This destruction is accompanied by the destruction of the segments of values associated present in memory and returns inaccessible those present on disc.

SUBROUTINE JEDETR (nom_o)

in	nom_o	K24	name of object JEVEUX (simple object, collection or object of named collection)

The place released in the catalogue or the repertoire of names becomes immediately reusable for another descriptor. The place, possibly used on disc, will be recoverable only by one operation of “retassage” of the file.

Note:
It is not possible to destroy an object of numbered collection,
for a named collection, the destruction of an object brings up to date the attribute NUTIOC .
In the case of a simple object of standard repertoire of names, it is not possible to destroy one not of entry in the repertoire, the call to this function destroys the object completely JEVEUX.

One can also use the following routine to destroy a set of descriptors.

SUBROUTINE JEDETC (nom_cl, souch, ipos)

in	nom_cl	K1	name of the class or “to treat all the open classes.
in	souch	K*	character string to be identified in the whole of the names contained in the repertoire of one or more classes.
in	ipos	I	position in the name of the chain to be identified.

One searches all the descriptors whose name contains under chain souch with the position ipos in the classes defined by the parameter nom_cl and one destroys the descriptors thus located.

It is preferable to use the routine JEDETR when one knows the name of the objects explicitly to be destroyed, the call in a loop or a routine of low level can appear very expensive.

4.16 The recovery of the size of the storage areas available

SUBROUTINE JEDISP (nbp, lplace)

in	nbp	I	many sought positions.
in	lplace	V_I	size in unit of addressing of the various zones available.

JEDISP return, by decreasing order in the vector of entireties $lplace$, size of nbp greater segments of liquid assets for an allowance (JEVEUO), at the moment of the call. The values obtained remain valid under the condition of the only use of the requests of allowance concerning of the segments of values lower or equal lengths.

Thus $lplace(1)$ return the size of the largest object than one could allocate, $lplace(2)$ that of the largest object than one could allocate after having used the zone corresponding to $lplace(1)$.

Note:

The allowance of a collection can involve the loading in memory of the attributes objects and make thus null and void the values obtained by JEDISP .

4.17 Consultations

Reading of an attribute of an object JEVEUX or of an object of collection

SUBROUTINE JELIRA (nom_o, nom_at, ival=ival, cval=cval)

in	nom_o	K24	name of object JEVEUX
in	nom_at	K*	name of attribute (cf appendix 2: list of the accessible attributes)
out	ival	I	whole value for an attribute of the whole type.
out	cval	K*	text for an attribute of type character.

Note:

In addition to the attributes described with [§2], it is possible to recover with the value XOUS the type of object JEVEUX associated with the descriptor, "S" simple object, "X" collection.
Two arguments ival and cval are optional. The two arguments can be provided at the time of the call to the routine with or without the respective identifier "ival=" and "cval=".

Reading of the value of the current mark

SUBROUTINE JEVEMA (mark)

out	mark	I	value of the current mark

Research of the list of the names of descriptors present in a class

SUBROUTINE JELSTC (nom_cl, souch, ipos, nbmax, l_nom, nbval)

in	nom_cl	K1	name of the class or "to treat all the open classes.
in	souch	K*	character string to be identified in the whole of the names contained in the repertoire of a class.
in	ipos	I	position in the name of the chain to be identified.

in	nbmax	I	dimension of the vector of K24 provided below
VAr	l_nom	V_K24	vector containing the list of the identifiers answering the search criterion
out	nbval	I	maximum number of identifiers answering the search criterion $nbval=-nbval$ if $nbval>nbmax$

4.18 Impressions

Several routines make it possible to print the contents of an object JEVEUX or of an object of collection, to print the attributes, to print out of the memory managed by JEVEUX (objects present, position, size, etc), to print the catalogue of a class, or to consult the state of a database.

SUBROUTINE JEIMPO (links, nom_o, param, cmess)

Impression of the contents of an object JEVEUX or of an object of collection

in	links	I	Logical number of unit associated with the print file, within Code_Aster one will use 6 ('MESSAGE') or 8 ('RESULT')
in	nom_o	K24	name of object JEVEUX
in	param	K*	not currently used (white ''obligatory)
in	cmess	K*	text appearing in comment with the impression

SUBROUTINE JEIMPA (links, nom_o, cmess)

Impression of all the attributes of an object JEVEUX or of an object of collection

in	links	I	Logical number of unit associated with the print file, within Code_Aster one will use 6 ('MESSAGE') or 8 ('RESULT')
in	nom_o	K24	name of object JEVEUX
in	cmess	K*	text appearing in comment with the impression

SUBROUTINE JEIMPR (links, nom_cl, cmess)

Impression of the catalogue

in	links	I	Logical number of unit associated with the print file, within Code_Aster one will use 6 ('MESSAGE') or 8 ('RESULT')
in	nom_cl	K1	name of class or ''to treat all the open classes
in	cmess	K*	text appearing in comment with the impression

SUBROUTINE JEIMPM (links)

Impression of the state of the zone memory managed by JEVEUX

in	links	I	Logical number of unit associated with the print file, within Code_Aster one will use 6 ('MESSAGE') or 8 ('RESULT')

The routine JEIMPM allows to publish the contents of zone memory managed at the time of the call. Here in the order significance of the values appearing with the impression:

- name in summary of the class associated with the object,

- G base TOTAL ,
- V base BIRD ,

- identifier of collection, if this last is worth 0 , it is a simple object,
- identifier of object simple or number of object of collection,
- level of call in the pile **JEMARQ/JEDEMA**
- address memory of the segment of value,
- use of the segment of value, is worth U or X ,
- length in unit of addressing (whole 8 bytes in 64 bits) segment of values,
- statute of the segment of value, is worth D or With ,
- name of the object, follow-up possibly of the number of object of collection.

Possible combinations of the statute and the use of a segment of values, as their significance are the following ones:

- U D : segment of values used, in access in writing and possibly in reading. It will have to be discharged on disc.
- U WITH : segment of values used, in access in reading. It will not be discharged on disc.
- X D : segment of values unutilised, déchargeable: it will have to be discharged on disc. A request in writing or reading on the associated object directly returns its position without movement report and access disc. Its position can be recovered constantly to place a new segment of values at the cost of an access disc (origin: U D).
- X WITH : segment of values unutilised, removable. A request in writing or reading on the associated object directly returns its position without movement report and access disc. Its position can be recovered constantly to place a new segment of values without access disc (origin: U WITH).

Example of impression obtained:

```
 CL  --NUM-- - MY  ---------IADY--------- - U - LON UA -  -S- ------------- NAME --------------

|G|   0|       1|  -2|           118073808|U|         11| D| ________GLOBALE ________$$CARA
|G|   0|       2|  -2|           120752752|U|       4000| D| ________GLOBALE ________$$IADD
|G|   0|       3|  -2|           118553440|U|        251| D| ________GLOBALE ________$$GENR
|G|   0|       4|  -2|           118533472|U|        251| D| ________GLOBALE ________$$TYPE
|G|   0|       5|  -2|           118559216|U|       1001| D| ________GLOBALE ________$$DOCU
|G|   0|       6|  -2|           120784832|U|       2000| D| ________GLOBALE ________$$ORIG
|G|   0|       7|  -2|           120800912|U|       8004| D| ________GLOBALE ________$$RNOM
|G|   0|       8|  -2|           120865024|U|       2000| D| ________GLOBALE ________$$LTYP
|G|   0|       9|  -2|           120881104|U|       2000| D| ________GLOBALE ________$$LONG
|G|   0|      10|  -2|           120897184|U|       2000| D| ________GLOBALE ________$$LONO
|G|   0|      11|  -2|           120913264|U|       2000| D| ________GLOBALE ________$$DATE
|G|   0|      12|  -2|           120929344|U|       2000| D| ________GLOBALE ________$$LUTI
|G|   0|      13|  -2|           120945424|U|       3203| D| ________GLOBALE ________$$HCOD
|G|   0|      14|  -2|      46912496128016|U|     188742| D| ________GLOBALE ________$$USADI
|G|   0|      15|  -2|           118578720|U|      62914| D| ________GLOBALE ________$$ACCE
|G|   0|      16|  -2|           118002208|U|       4000| D| ________GLOBALE ________$$MARQ
|G|   0|      17|  -2|           118034288|U|       2000| D| ________GLOBALE ________$$INDI
|G|   0|      18|  -2|           119082112|U|     102400| D| ________GLOBALE ________$$TLEC
|G|   0|      19|  -2|           119901392|U|     102400| D| ________GLOBALE ________$$TECR
|G|   0|      20|  -2|           120720672|U|       4000| D| ________GLOBALE ________$$IADM
|G|   0|      21|   0|            93362960|X|         21| D| &FOZERO             .PROL
|G|   0|      22|   0|            98181600|X|          2| D| &FOZERO             .VALE
|G|   0|      23|   0|           108074656|X|          1| D| &&_NUM_CONCEPT_UNIQUE
|G|   0|      24|   0|           123411680|X|       5010| D| &&SYS.KRESU
|G|   0|      25|   0|            88929024|X|         11| D| &CATASTROPHES.ACOUSTIQUE
…
```

SUBROUTINE JEIMPD (links, nom_cl, cmess)

Impression of the list of the objects JEVEUX present in a database:

in	links	I	Logical number of unit associated with the print file, within Code_Aster one

			will use 6 ('MESSAGE') or 8 ('RESULT')
in	nom_cl	K1	name of class or "to treat all the open classes
in	cmess	K*	text appearing in comment with the impression

SUBROUTINE JEPRAT (links, name, nom_at, param, cmess)

This routine publishes the catalogue of the objects JEVEUX present on disc.

in	links	I	Logical number of unit associated with the print file, within Code_Aster one will use 6 ('MESSAGE') or 8 ('RESULT')
in	name	K24	name of object JEVEUX or name of class preceded by the character $
in	nom_at	K8	name of the attribute or suffix of the system object to be printed. nom_at takes its values among the following list: for a collection $$DESO, $$IADD, $$IADM, $$PEPL, $$NOM, $$REEL, $$LONG, $$LONO, $$LUTI, $$NUM. for a whole class $$CARA, $$IADD, $$GENR, $$TYPE, $$DOCU, $$ORIG, $$RNOM, $$LTYP, $$LONG, $$LONO, $$DATE, $$LUTI, $$HCOD, $$USADI, $$ACCE, $$MARQ, $$TLEC, $$TECR, $$IADM
in	param	K*	not currently used (white "obligatory)
in	cmess	K*	text appearing in comment with the impression

Note:
This routine is a utility reserved for the assistance.

4.19 Utilities

The following utilities were written to copy or reset parts of objects simple or objects of collection.

White or restoring following the type of the object JEVEUX.

SUBROUTINE JERAZO (nom_o, N, i1)

in	nom_o	K24	name of object JEVEUX
in	N	I	many values to be reset
in	I1	I	index in the vector of the first value to be reset

4.20 A utility of debugging

A bad use of the address returned by the routine JEVEUO can lead to a crushing report, and in particular to the destruction of the chaining of the segments of values. The programmer can in this case try to locate the routine aggressive crushing by instrumenting his source code with calls to JXVERI. This routine controls the entireties present on both sides of each segment of values: at the time of a overflow these values are in general crushed. Crushing upstream corresponds to the entirety located right in front of the first value of the segment of values. Crushing downstream corresponds to the entirety located just behind the last value of the segment of values.

SUBROUTINE JXVERI (cfic, cmess)

in	cfic	K*	text defining the local name of the print file, within Code_Aster one will use 'MESSAGE' or 'RESULT'

in	cmess	K*	text appearing in comment with the impression

This routine transmits a message indicating the integrity or not chaining at the time which there exists, or during the crushing of the identifiers located on both sides of values.

4.21 Routines of initialization used by the supervisor

The use of objects JEVEUX is not possible that after the initialization which requires to define:

- LE number maximum of databases, which one will be able to manage simultaneously,
- size of the zone memory managed by JEVEUX, which will be allocated dynamically.

This initialization is obligatorily carried out by the supervisor within the orders BEGINNING or CONTINUATION using the routine JEDEBU, which creates, automatically, all the system objects necessary:

SUBROUTINE JEDEBU (nbases, lzone, cmess, cvig, idebug)

in	nbases	I	maximum numbers of simultaneous databases (≤5)
in	lzone	I	memory size allocated in unit of addressing
in	cmess	K*	local name of the print file of the error messages
in	cvig	K*	not used
in	idebug	I	used for an operation in mode debug $idebug=0$ normal operating process $idebug=1$ debug JEVEUX engaged

When the application JEVEUX is initialized, it is important to open the classes of objects on which one wishes to work. To open a class, it is necessary to specify its characteristics: presence or not of a file associated, name of the class, name of the associated database, characteristic of the file, etc… In mode DEBUG the operation of the manager of memory is modified, unloadings on disc are not differed any more and memory occupied by a segment of values places it is positioned with an indefinite value. This operating process is used for déverminer code: the use of an address corresponding to a segment of values released causes a brutal stop.

SUBROUTINE JEINIF (stin, stout, nom_bas, nom_cl, nmax, nbloc, lbloc)

in	stin	K*	text defining the statute at the beginning of work: 'DUMMY' not of associated file 'BEGINNING'D initialization or restoring' an existing class 'CONTINUES'recovery of the contents D' an existing class
in	stout	K*	text defining the statute at the end of the work: 'SAVE'safeguard on file in fine D' application 'DESTROYED'destruction of the class in fine D' application
in	nom_bas	K*	local name of the database (example: 'TOTAL', 'BIRD')
in	nom_cl	K1	name of the associated class (example: 'G', 'V')
in	nmax	I	maximum number of names of objects JEVEUX in the class
in	nbloc	I	maximum number of recordings of the file of associated direct access
in	lbloc	I	length of the recordings of the file of associated direct access

Note:

A class can be open or closed constantly, up to the maximum number of manageable classes simultaneously.

To close a class, in the course of application:

SUBROUTINE JELIBF (cond, nom_cl)

in	cond	K*	text allowing to overload the value of stout defined in JEINIF 'SAVE' with immediate safeguard on file 'DESTROYED' with immediate destruction
in	nom_cl	K*	name of the class to close (example: 'G', 'V')

Lastly, it is obligatory to call on the routine of closure the application which carries out the closing of all the still open classes after safeguard of the objects JEVEUX and of the catalogues present in memory and the application by instruction FORTRAN stops STOP. This routine is called within the order END by the supervisor.

SUBROUTINE JEFINI (cond)

in	cond	K8	condition of fence: 'NORMAL' safeguard according to the value of stout defined in JEINIF 'ERROR' safeguard of the opened classes, for possible later analysis

Note:

It is only STOP usable in all the application to be able to re-use the databases.
a user must call this routine to stop his application after detection of an error with cond = 'ERROR',
Within Code_Aster a stop with the condition cond = 'ERROR' do not authorize the update of the base TOTAL in the repertoire of the user, the concepts created not having been validated.

SUBROUTINE JETASS (nom_cl)

in	nom_cl	K1	name of the associated class (example: 'G', 'V')

This routine is intended to recover the recordings become unutilised following the destruction of the objects JEVEUX associated. That relates to only the large objects, i.e. those which require at least a recording of the file of direct access. The only effect is sequentially to reorder the recordings, the effective recovery of the place must then be carried out by recopying the beginning of the file of direct access, only the supervisor of Aster can carry out this action.

4.22 Routines of safeguard and second reading used by the supervisor

The base TOTAL allows to save and read again on the same platform the Jeveux objects obtained during an execution. This binary file, composed of one or more subfiles is adherent with the platform of execution (operating system 32 or 64 bits). It is possible, using the two following routines, to record the complete contents with format HDF and to read again it indépendemment type of platform.

SUBROUTINE JEIMHD (fichdf, nom_cl)

in	fichdf	K*	local name of file HDF to be created
in	nom_cl	K1	name of the associated class (example: 'G', 'V')

SUBROUTINE JELIHD (nom_bas, fichdf, nom_cl)

in	nom_bas	K*	local name of the database (example: 'TOTAL', 'VOLATILE')
in	fichdf	K*	local name of file HDF to be read again
in	nom_cl	K1	name of the associated class (example: 'G', 'V')

4.23 Routines of interrogation for the supervisor

SUBROUTINE JELIAD (nom_cl, numr, nboct)

in	nom_cl	K1	name of the associated class (example: 'G', 'V')
out	numr	I	number of the recording
out	nboct	I	many bytes before the recording

This routine returns the number of the recording containing the system object $$RNOM in the base considered. This repertoire of names being a characteristic of the base, the supervisor uses this property "to identify" the base and to make checks of coherence in continuation.

5 Examples of use

One will suppose, in this part, to be themselves placed in an environment of application (for example that of ASTER) and to have opened the databases associated with the classes G, V and L. One will mention only the calls to the routines JEVEUX, the common declarations and FORTRAN are omitted.

5.1 Creation and re-use of a vector of reality

That is to say the following problem: one wants to create (in the routine SUBA) a vector of real containing the coordinates X and Y of nno nodes. One will call this vector 'COORDO_XY' and one will re-use it in routine SUBB.

```
                SUBROUTINE SUBA (...)
                  …
                  #include "jeveux.h"
                  real (kind=8), to point:: COOR (:) => no one ()

                  !  -  Beginning of the instructions:
(A)                  CAL JEMARQ ()
                  !  -  Allowance of the vector on the 'TOTAL' basis:
(b)                  CAL WKVECT ('COORDO_XY', 'G V R', 2*nno, vr=COOR)
                  !  -  "filling of the vector"
                     C ino = 1, No
                           COOR (2* (ino-1) +1) = X
                           COOR (2* (ino-1) +2) = Y
                     ENDDO
(c)                  CAL JEDEMA ()
                  END


                  SUBROUTINE SUBB (...)
                  …
                  real (kind=8), to point:: COOR (:) => no one ()
                  !  - Beginning of the instructions:
(d)                  CAL JEMARQ ()
                  !  - Recovery of the address of the vector (in reading):
(E)                  CAL JEVEUO ('COORDO_XY', 'It, vr=COOR)
                  !  - Recovery of the coordinates of the 27th node:
                     X27 =   COOR (2 * 26 + 1)
                     Y27 =   COOR (2 * 26 + 2)
                     …
(F)                  CAL JEDEMA ()
                  END
```

Comments:

- lines (A), (c), (d), (E): routines JEMARQ/JEDEMA allow to release automatically the objects at the end of the routines [§4.11],
- line (b):
- 'G V R':
  - 'G' : base "Total" (database of the user cf [§3.3])
  - 'V': vector,
  - 'R': reality,
- 2*nno : length of the vector
- COOR : pointer on the segment of values
- line (E)
- 'L ' : access in "reading" of the object (cf [§4.5]).

5.2 Creation of a repertoire of names and insertion of two names

```
      CAL JECREO ('MES_NOMS', 'G NR K8')
      CAL JEECRA ('MES_NOMS', 'NOMMAX', ival=25)
C
      CAL JECROC (JEXNOM ('MES_NOMS', 'NOM_1'))
      CAL JECROC (JEXNOM ('MES_NOMS', 'NOM_5'))
C
      CAL JELIRA ('MES_NOMS', 'NUTIOC', ival=IVAL)
      CAL JENONU (JEXNOM ('MES_NOMS', 'NOM_5'), NUM)
      CAL JENUNO (JEXNUM ('MES_NOMS', 1), NAME)
```

The call to JENONU return value 2 in the variable NUM, The call to JENUNO return the value 'NOM_1' in the variable NAME.

5.3 Dispersed collection of vectors of entireties

The objects of this collection are named in the repertoire, managed by the user and created in the preceding example; they are variable length.

```
      CAL JECREC ('MA_COLL', 'G V I', 'NO', 'DISPERSES', 'VARIABLE', 25)
      CAL JECROC (JEXNOM ('MA_COLL', 'NOM_13'))
      CAL JEECRA (JEXNOM ('MA_COLL', 'NOM_13'), 'LONMAX', ival=125)
      CAL JEECRA (JEXNOM ('MA_COLL', 'NOM_1'), 'LONMAX', ival=250)
```

In another routine, the object is used 'NOM_13' who has just been created:

```
      CAL JEVEUO (JEXNOM ('MA_COLL', 'NOM_13'), 'E', JTAB)
      CAL JELIRA (JEXNOM ('MA_COLL', 'NOM_13'), 'LONMAX', ival=LNOM13)
   C
      C 10 K = 1, LNOM13
        ZI (JTAB - 1 + K) = K
   10 CONTINUOUS
   .
      CAL JEDETR (JEXNOM ('MA_COLL', 'NOM_13'))
```

The attribute NUTIOC collection is updated at the same time as that of the simple object 'MES_NOMS'. The last instruction destroys the object of collection 'NOM_13' and the name in the repertoire 'MES_NOMS'.

5.4 Contiguous collection of vectors of entireties

The objects of this collection are named in the repertoire of names 'MES_NOMS', already used for the two preceding examples. They have a variable length defined by a vector lengths 'LENGTHS' managed by the user.

In a first routine, one creates the collection and one adds an object 'NOM_24'

```
CAL JECREC ('MA_COLL', 'G V I', 'NO MES_NOMS', 'CONTIG', 'VARIABLE', 25)
CAL JECROC (JEXNOM ('MA_COLL', 'NOM_24'))
```

In another routine, one defines the length of objects 2 to 25, equal to the number of insertion of the object:

```
     CAL JEVEUO ('LENGTHS', 'E', JTAB)
 C
     C 10 I = 2.25
         CAL JEECRA (JEXNUM ('MA_COLL', I), 'LONMAX', ival=I)
 10  CONTINUOUS
```

In another routine, one defines the length of object 1 (equalizes to 50), and one reaches all the collection to define the 13th component of the first object:

```
        CAL JEECRA (JEXNUM ('MA_COLL', 1), 'LONMAX', ival=50)
   C
        CAL JEVEUO ('MA_COLL', 'E', vi=TABC)
        K = 13
        TABC (K) = .....
```

In another routine, the 2 are allocated^{ième} object to define all its components:

```
    CAL JEVEUO (JEXNUM ('MA_COLL', 2), 'E', vi=TABOC)
    CAL JELIRA (JEXNUM ('MA_COLL', 2), 'LONMAX', ival=L2)
    C 20 K = 1, L2
       TABOC (K) = .....
```

```
20  CONTINUOUS
```

In another routine, one allocates the whole collection to define the first component of the 3ième object, which one reaches by the vector cumulated lengths:

```
       CAL JEVEUO ('MA_COLL', 'E', JTABC)
C
       CAL JEVEUO (JEXATR ('MA_COLL', 'LONCUM'), 'E', JTABCU)
C
       IOBJ = 3
       IAD  = ZI (JTABCU - 1 + IOBJ)
       ZI (JTABC - 1 + IAD) = .....
```

One will notice the two manners which make it possible to define the length of an object of collection:

- by using the call to JEECRA
- by directly affecting the value in the vector lengths.

The first request with JEVEUO (within the last framework) an access to the contiguous collection carries out overall and thus returns the address of the first object, with load for the user to move compared to this address to reach a particular object. The second request gives access directly an object of collection and returns the address of this object.

In the last example, one recovers the cumulated lengths of the objects of the collection, which makes it possible to the user to thus reach any object, without multiplying the requests JEVEUO.

Note:
In the three cases, all the objects of the contiguous collection are present in memory.

6 Appendix 1: list of the subroutines "user"

Functions of the type CHARACTER*32

```
JEXNOM (nom_co, nom_oc)
JEXNUM (nom_co, num_oc)
JEXATR (nom_co, 'LONCUM')
```

Creation of the descriptors

```
CAL JECREO (nom_os, lis_at)
CAL JECREC (nom_co, lis_at, access, stock, modlon, nmaxoc)
```

Assignment of an attribute

```
CAL JEECRA (nom_o, nom_at, ival=ival, cval=cval)
```

Creation of a name in a repertoire of names

```
CAL JECROC      (JEXNOM (nom_os, name))
```

Creation of an object of collection

```
CAL JECROC      (JEXNOM (nom_co, nom_oc))
```

Request of allowance

```
CAL JEVEUO      (nom_o, concealment, jtab, VI, vr,…)
CAL WKVECT      (nom_o, lis_at, length, jtab, VI, vr,…)
```

Enlarging of a vector

```
CAL JUVECA      (nom_os, length)
```

Recopy

```
CAL JEDUPO      (nom_in, nom_clo, nom_ou, dup_co)
CAL JEDUPC      (nom_cli, souchi, ipos, nom_clo, soucho, dup_co)
```

Requests of release and safeguard

```
CAL JEMARQ ()
CAL JEDEMA ()
CAL JELIBE (nom_o)
CAL JELIBZ ()
```

Request of existence

```
CAL JEEXIN (nom_o, iret)
```

To check the existence of a name and to obtain its sequence number

```
CAL JENONU (JEXNOM (nom_o, name), num)
```

To obtain the name associated with a sequence number

```
CAL JENUNO (JEXNUM (nom_o, num), name)
```

Destruction of the descriptors

```
CAL JEDETR (nom_o)
CAL JEDETC (nom_cl, souch, ipos)
```

Place available

```
CAL JEDISP (nbp, lplace)
```

Consultations

```
CAL JELIRA (nom_o, nom_at, ival=ival, cval=cval)
CAL JELSTC (nom_cl, souch, ipos, nbmax, l_nom, nbval)
CAL JEVEMA (mark)
```

Impressions

```
CAL JEIMPO (links, nom_o, param, cmess)
CAL JEIMPA (links, nom_o, cmess)
CAL JEIMPR (links, nom_cl, cmess)
CAL JEIMPM (links)
CAL JEIMPD (links, nom_cl, cmess)
CAL JEPRAT (links, name, nom_at, param, cmess)
```

To reset n values of a vector

```
CAL  JERAZO (nomlu, nor, i1)
```

Debugging

```
CAL JXVERI (cfic, cmess)
```

And environment implementation of exploitation (reserved to the Supervisor)

```
CAL JEDEBU (nbases, lzone, cmess, cvig, idebug)
CAL JEINIF (stin, stout, nom_base, nom_cl, nmax, nbloc, lbloc)
CAL JELIBF (cond, nom_cl)
CAL JEFINI (cond)
CAL JETASS (nom_cl)
CAL JEIMHD (fichdf, nom_cl)
CAL JELIHD (nom_base, fichdf, nom_cl)
CAL JELIAD (nom_cl, numr, nboct)
```

7 Appendix 2: list of the accessible attributes

Following coding is used:

With	easily affected and nonmodifiable attribute then	(by JECREO, JECREC or JEECRA)
M	modifiable attribute	(by JEECRA)
C	consultable attribute only	(by JELIRA or JEIMPA)
bone	simple object	
Co	collection	
oc	object of collection	
disper	dispersed collection	
contig	contiguous collection	
VAr	variable collection length	
const	collection constant length	

When the attribute is only accessible for a given kind, this last is indicated in first column § the generic attributes. The white indicates that the attribute is not accessible.

In the case of the collections and of the objects of collection, one indicates, if it is necessary, the type of collection concerned (contig & VAr for a contiguous collection variable length).

	possible values	**kind**	**bone**	**Co**	**oc**
CLAS			With	With	C
GENR	E , V , or NR		With	With	C
TYPE (1)	I, S, R, C, L, K8, K16, K24, K32, K80		With	With	C
LTYP			C	C	C
LONMAX		V	With	With (const)	With (VAr)
NOMMAX		NR	With		
LONUTI		V	M	M (const)	M (VAr)
NOMUTI		NR	C	C	
DOCU			M	M	
DATE			C	C	
IADM			C	C (contig)	C (disper)
IADD			C	C (contig)	C (disper)
LONO			C	C (contig)	C (disper)
USE			C	C (contig)	C (disper)
ACCESS	NO, NO nom_uti, NAKED (2)			With	
STORAGE	CONTIG			With	

	DISPERSE				
MODELONG	CONSTANT VARIABLE nom_uti (3)			With	
LONT				With (contig & VAr)	
NMAXOC				With	
NUTIOC				C	

Remarks

(1) The routine JELIRA only return K for the value of the attribute TYPE for the type K8, K16, K24, K32 and K80 . It then to consult the attribute LTYP .

(2) The routine JELIRA return is " NO " for an internal pointer, that is to say " NO name " where name is an external name of pointer of names.

(3) The routine JELIRA return is " CONSTANT " , that is to say " VARIABLE " , that is to say " VARIABLE name " where name is a name of external pointer length.