Focus Manuals: Programing Guide: Structures in BIEF

links from Programing Guide

A short description

In BIEF 6.0, structures will be composed of integer and real numbers, of pointers to other structures or to integer and real arrays. The structures defined in this way are, for the time being, as follows:

- **BIEF_OBJ** (may be a vector, a matrix or a block)
- **BIEF_MESH** (information on a mesh)
- **SLVCFG** (Solver Configuration)
- **BIEF_FILE** (Description of a data file)

The notions of **VECTOR**, **MATRIX** and **BLOCK** that were pre-programmed in BIEF 6.+ have been gathered in a single structure called **BIEF_OBJ**. This will enable what is called “polymorphism” in Object Oriented Languages, i.e. the fact that arguments of subroutines may be of different types. As a matter of fact, many subroutines in BIEF are able to treat in the same way vectors or blocks of vectors (see for example OS), matrices or blocks of matrices (see e.g. **SOLVE** and **DIRICH**). Polymorphism is possible in Fortran 90 with the use of interfaces, however it requires the writing of one subroutine per combination of types, and thus leads to a lot of duplication. The use of a single structure **BIEF_OBJ** was thus more elegant, the only drawback being that the misuse of a matrix as a vector, for example, cannot be checked by the compiler but only by the subroutines dealing with such structures.

Information on the structures can be simply retrieved by the component selector. We shall also refer to **BIEF_OBJ** structures as **VECTOR**, **MATRIX** or **BLOCK**, depending on their use, as is done below.

VECTOR

This may be any vector (a simple array) or a vector defined on the mesh, with values for every point of the mesh. In the latter case, there is a corresponding discretisation type and numbering system (global or boundary numbering of nodes or numbering of elements). For example, a vector defined on all the mesh with a discretisation P0 will be implicitly given according to the element numbers. In certain conditions, a vector may change discretisation while the calculations are being carried out.

A vector has a first dimension which corresponds to the number of nodes to which it applies. There is also a second dimension (for example, the off-diagonal terms of a matrix).

Any vector is in fact an array with 2 dimensions which the user can process as he wishes.

MATRIX

Matrices are also linked to the mesh. Different storage methods are possible. These matrices can be multiplied by the vectors mentioned above.

BLOCK

A block is a set of structures. This notion has proved of particular importance for:

- Writing general solvers for linear systems, with the possibility of the matrix being a block of several matrices.
- Using simple orders to group together and process sets of vectors or matrices, for example the arrays of variables which are advected by the method of characteristics.

- Eliminate the need for certain arrays to follow one another in the memory.

BIEF_MESH structure

This structure includes all information concerning the mesh (connectivity tables, boundary points, point coordinates, etc.). It replaces a large number of arrays used in releases of BIEF prior to 3.2

SLVCFG

It stands for “SoLVer ConFiGuration). This is a simple structure to store all the information needed by the subroutine **SOLVE** for solving linear systems (choice of the method, accuracy, preconditioning, etc).

Reference description of the structures

Module **BIEF_DEF** of the library is given hereafter, with the list of components for every structure and a short description.

++++ POINTER TO BIEF OBJ

```
!
!=====================================================================
!
! STRUCTURE OF POINTER TO A BIEF_OBJ, TO HAVE ARRAYS OF POINTERS
! IN THE BIEF_OBJ STRUCTURE FOR BLOCKS
!
! BIEF VERSION 6.0
!
!=====================================================================
!
! THIS IS NECESSARY IN FORTRAN 90 TO HAVE ARRAYS OF POINTERS
! LIKE THE COMPONENT ADR BELOW, WHICH ENABLES TO BUILD BLOCKS
! WHICH ARE ARRAYS OF POINTERS TO BIEF_OBJ STRUCTURES
!
      TYPE POINTER_TO_BIEF_OBJ
         SEQUENCE
         TYPE(BIEF_OBJ), POINTER :: P
      END TYPE POINTER_TO_BIEF_OBJ
!
```

++++

BIEF_OBJ

```
        TYPE BIEF_OBJ
  !

!-----------------------------------------------------------------
------
  !
  ! HEADER COMMON TO ALL OBJECTS
```

```
  !
  ! KEY : ALWAYS 123456 TO CHECK MEMORY OVERWRITING
        INTEGER KEY
  !
  ! TYPE: 2: VECTOR, 3: MATRIX, 4: BLOCK

        INTEGER TYPE
  !
  ! NAME: FORTRAN NAME OF OBJECT IN 6 CHARACTERS
        CHARACTER(LEN=6) NAME
  !

!-----------------------------------------------------------------------
------
  !
  ! FOR VECTORS
  !
  !
  ! NAT: NATURE (1:DOUBLE PRECISION 2:INTEGER)
        INTEGER NAT
  !
  ! ELM: TYPE OF ELEMENT
        INTEGER ELM
  !
  ! DIM1: FIRST DIMENSION OF VECTOR
        INTEGER DIM1
  !
  ! MAXDIM1: MAXIMUM SIZE PER DIMENSION
        INTEGER MAXDIM1
  !
  ! DIM2: SECOND DIMENSION OF VECTOR
        INTEGER DIM2
  !
  ! MAXDIM2: MAXIMUM SECOND DIMENSION OF VECTOR
        INTEGER MAXDIM2
  !
  ! DIMDISC: TYPE OF ELEMENT IF VECTOR IS DISCONTINUOUS AT
  ! THE BORDER BETWEEN ELEMENTS, OR 0 IF NOT
        INTEGER DIMDISC
  !
  ! STATUS:
  ! 0: ANY ARRAY
  ! 1: VECTOR DEFINED ON A MESH, NO CHANGE OF DISCRETISATION
  ! 2: VECTOR DEFINED ON A MESH, CHANGE OF DISCRETISATION ALLOWED
        INTEGER STATUS
  !
  ! TYPR: TYPE OF VECTOR OF REALS
  ! '0' : NIL '1' : EQUAL TO 1 'Q' : NO SPECIFIC PROPERTY
        CHARACTER*1 TYPR
  !
  ! TYPR: TYPE OF VECTOR OF REALS
```

```
  ! '0' : NIL '1' : EQUAL TO 1 'Q' : NO SPECIFIC PROPERTY
        CHARACTER*1 TYPI
  !
  ! POINTER TO DOUBLE PRECISION 1-DIMENSION ARRAY
  ! DATA ARE STORED HERE FOR A DOUBLE PRECISION VECTOR
        DOUBLE PRECISION, POINTER,DIMENSION(:)::R
  !
  ! POINTER TO INTEGER 1-DIMENSION ARRAY
  ! DATA ARE STORED HERE FOR AN INTEGER VECTOR
        INTEGER, POINTER,DIMENSION(:)::I
  !

!-----------------------------------------------------------------
------
  !
  ! FOR MATRICES
  !
  ! STO: TYPE OF STORAGE 1: CLASSICAL EBE 3: EDGE-BASED STORAGE
        INTEGER STO
  !
  ! ELMLIN: TYPE OF ELEMENT OF LINE
        INTEGER ELMLIN
  !
  ! ELMCOL: TYPE OF ELEMENT OF COLON
        INTEGER ELMCOL
  !
  ! TYPDIA: TYPE OF DIAGONAL
  ! '0' : NIL 'I' : IDENTITY 'Q' : NO SPECIFIC PROPERTY

        CHARACTER*1 TYPDIA
  !
  ! TYPEXT: TYPE OF EXTRA-DIAGONAL TERMS
  ! '0' : NIL 'S' : SYMMETRY 'Q' : NO SPECIFIC PROPERTY
        CHARACTER*1 TYPEXT
  !
  ! POINTER TO A BIEF_OBJ FOR DIAGONAL
        TYPE(BIEF_OBJ), POINTER :: D
  !
  ! POINTER TO A BIEF_OBJ FOR EXTRA-DIAGONAL TERMS
        TYPE(BIEF_OBJ), POINTER :: X
  !
  ! PRO: TYPE OF MATRIX-VECTOR PRODUCT
        INTEGER PRO
  !

!-----------------------------------------------------------------
------
  !
  ! FOR BLOCKS
  !
```

```
  ! BLOCKS ARE IN FACT ARRAYS OF POINTERS TO BIEF_OBJ STRUCTURES
  ! ADR(I)%P WILL BE THE I-TH BIEF_OBJ OBJECT
  !
  ! N: NUMBER OF OBJECTS IN THE BLOCK
        INTEGER N
  ! MAXBLOCK: MAXIMUM NUMBER OF OBJECTS IN THE BLOCK
        INTEGER MAXBLOCK
  ! ADR: ARRAY OF POINTERS TO OBJECTS (WILL BE OF SIZE MAXBLOCK)
        TYPE(POINTER_TO_BIEF_OBJ), POINTER, DIMENSION(:) :: ADR
  !

!-----------------------------------------------------------------
------
  !
      END TYPE BIEF_OBJ
  !
  !

!=================================================================
======
  !
```

BIEF_MESH

```
  !

!=================================================================
======
  !
  ! STRUCTURE OF MESH : BIEF_MESH
  !

!=================================================================
======
  !
      TYPE BIEF_MESH
  !
  ! 1) A HEADER
  !
  ! NAME: NAME OF MESH IN 6 CHARACTERS
        CHARACTER(LEN=6) NAME
  !
  ! 2) A SERIES OF INTEGER VALUES (DECLARED AS POINTERS TO ENABLE
  ! ALIASES)
  !
  ! NELEM: NUMBER OF ELEMENTS IN MESH
        INTEGER, POINTER :: NELEM
  !
  ! NELMAX: MAXIMUM NUMBER OF ELEMENTS ENVISAGED
        INTEGER, POINTER :: NELMAX
```

```
!
! NPTFR: NUMBER OF 1D BOUNDARY NODES, EVEN IN 3D
      INTEGER, POINTER :: NPTFR
!
! NPTFRX: NUMBER OF 1D BOUNDARY NODES, EVEN IN 3D
      INTEGER, POINTER :: NPTFRX
!
! NELEB: NUMBER OF BOUNDARY ELEMENTS (SEGMENTS IN 2D)
! IN 3D WITH PRISMS:
! number of LATERAL boundary elements for sigma mesh
      INTEGER, POINTER :: NELEB
!
! NELEBX: MAXIMUM NELEB
      INTEGER, POINTER :: NELEBX
!
! NSEG: NUMBER OF SEGMENTS IN THE MESH
      INTEGER, POINTER :: NSEG
!
! DIM: DIMENSION OF DOMAIN (2 OR 3)
      INTEGER, POINTER :: DIM
!
! TYPELM: TYPE OF ELEMENT (10 FOR TRIANGLES, 40 FOR PRISMS)
      INTEGER, POINTER :: TYPELM
!
! NPOIN: NUMBER OF VERTICES (OR LINEAR NODES) IN THE MESH
      INTEGER, POINTER :: NPOIN
!
! NPMAX: MAXIMUM NUMBER OF VERTICES IN THE MESH
      INTEGER, POINTER :: NPMAX
!
! MXPTVS: MAXIMUM NUMBER OF POINTS ADJACENT TO 1 POINT
      INTEGER, POINTER :: MXPTVS
!
! MXELVS: MAXIMUM NUMBER OF ELEMENTS ADJACENT TO 1 POINT
      INTEGER, POINTER :: MXELVS
!
! LV: MAXIMUM VECTOR LENGTH ALLOWED ON VECTOR COMPUTERS,
! DUE TO ELEMENT NUMBERING
      INTEGER, POINTER :: LV
!
!
! 3) A SERIES OF BIEF_OBJ FOR STORING INTEGER ARRAYS
!
! IKLE: CONNECTIVITY TABLE IKLE(NELMAX,NDP) AND KLEI(NDP,NELMAX)
      TYPE(BIEF_OBJ), POINTER :: IKLE,KLEI
!
! IFABOR: TABLE GIVING ELEMENTS BEHIND FACES OF A TRIANGLE
      TYPE(BIEF_OBJ), POINTER :: IFABOR
!
! NELBOR: ELEMENTS OF THE BORDER
```

```
          TYPE(BIEF_OBJ), POINTER :: NELBOR
  !
  ! NULONE: LOCAL NUMBER OF BOUNDARY POINTS FOR BORDER ELEMENTS
          TYPE(BIEF_OBJ), POINTER :: NULONE
  !
  ! KP1BOR: POINTS FOLLOWING AND PRECEDING A BOUNDARY POINT
          TYPE(BIEF_OBJ), POINTER :: KP1BOR
  !
  ! NBOR: GLOBAL NUMBER OF BOUNDARY POINTS
          TYPE(BIEF_OBJ), POINTER :: NBOR
  !
  ! IKLBOR: CONNECTIVITY TABLE FOR BOUNDARY POINTS
          TYPE(BIEF_OBJ), POINTER :: IKLBOR
  !
  ! IFANUM: FOR STORAGE 2, NUMBER OF SEGMENT IN ADJACENT ELEMENT
  ! OF A TRIANGLE
          TYPE(BIEF_OBJ), POINTER :: IFANUM
  !
  ! IKLEM1: ADRESSES OF NEIGHBOURS OF POINTS FOR FRONTAL
  ! MATRIX-VECTOR PRODUCT
          TYPE(BIEF_OBJ), POINTER :: IKLEM1
  !
  ! LIMVOI: FOR FRONTAL MATRIX-VECTOR PRODUCT. ADDRESSES OF POINTS
  ! WITH A GIVEN NUMBER OF NEIGHBOURS.
          TYPE(BIEF_OBJ), POINTER :: LIMVOI
  !
  ! NUBO: FOR FINITE VOLUMES, GLOBAL NUMBERS OF VERTICES OF
SEGMENTS
          TYPE(BIEF_OBJ), POINTER :: NUBO
  !
  ! FOR SEGMENT-BASED STORAGE

  !
  ! GLOSEG: GLOBAL NUMBERS OF VERTICES OF SEGMENTS
          TYPE(BIEF_OBJ), POINTER :: GLOSEG
  ! ELTSEG: SEGMENTS FORMING AN ELEMENT
          TYPE(BIEF_OBJ), POINTER :: ELTSEG
  ! ORISEG: ORIENTATION OF SEGMENTS FORMING AN ELEMENT 1:TRIGO
2:CLOCKWISE
          TYPE(BIEF_OBJ), POINTER :: ORISEG
  !
  !
  ! SERIES OF ARRAYS FOR PARALLELISM
  ! HERE GLOBAL MEANS NUMBER IN THE WHOLE DOMAIN
  ! LOCAL MEANS NUMBER IN THE SUB-DOMAIN
  !
  ! KNOLG: GIVES THE INITIAL GLOBAL NUMBER OF A LOCAL POINT
          TYPE(BIEF_OBJ), POINTER :: KNOLG
  ! NACHB: NUMBERS OF PROCESSORS CONTAINING A GIVEN POINT
          TYPE(BIEF_OBJ), POINTER :: NACHB
  ! ISEG: GLOBAL NUMBER OF FOLLOWING OR PRECEDING POINT IN THE
```

```
BOUNDARY
  ! IF IT IS IN ANOTHER SUB-DOMAIN.
        TYPE(BIEF_OBJ), POINTER :: ISEG
  ! KNOGL: INVERSE OF KNOLG, KNOGL(KNOLG(I))=I. LOCAL NUMBER OF A
  ! POINT WITH GIVEN GLOBAL NUMBER
        TYPE(BIEF_OBJ), POINTER :: KNOGL
  ! ADDRESSES IN ARRAYS SENT BETWEEN PROCESSORS
        TYPE(BIEF_OBJ), POINTER :: INDPU
  !
  ! DIMENSION NHP(NBMAXNSHARE,NPTIR). NHP(IZH,IR) IS THE GLOBAL
NUMBER
  ! IN THE SUB-DOMAIN OF A POINT WHOSE NUMBER IS IR IN THE
INTERFACE
  ! WITH THE IZ-TH HIGHER RANK PROCESSOR
        TYPE(BIEF_OBJ), POINTER :: NHP
  ! NHM IS LIKE NHP, BUT WITH LOWER RANK PROCESSORS
        TYPE(BIEF_OBJ), POINTER :: NHM
  !
  ! FOR FINITE VOLUMES AND KINETIC SCHEMES
        TYPE(BIEF_OBJ), POINTER :: JMI
  ! ELEMENTAL HALO NEIGHBOURHOOD DESCRIPTION IN PARALLEL
  ! IFAPAR(6,NELEM2)
  ! IFAPAR(1:3,IELEM): PROCESSOR NUMBERS BEHIND THE 3 ELEMENT
EDGES
  ! NUMBER FROM 0 TO NCSIZE-1
  ! IFAPAR(4:6,IELEM): -LOCAL- ELEMENT NUMBERS BEHIND THE 3 EDGES
  ! IN THE NUMBERING OF PARTITIONS THEY BELONG TO
        TYPE(BIEF_OBJ), POINTER :: IFAPAR
  !
  ! 4) A SERIES OF BIEF_OBJ FOR STORING REAL ARRAYS
  !
  ! XEL: COORDONNEES X PAR ELEMENTS
        TYPE(BIEF_OBJ), POINTER :: XEL
  !
  ! YEL: COORDONNEES Y PAR ELEMENTS
        TYPE(BIEF_OBJ), POINTER :: YEL
  !
  ! ZEL: COORDONNEES Z PAR ELEMENTS
        TYPE(BIEF_OBJ), POINTER :: ZEL
  !
  ! SURFAC: AREAS OF ELEMENTS
        TYPE(BIEF_OBJ), POINTER :: SURFAC
  !
  ! SURDET: 1/DET OF ISOPARAMETRIC TRANSFORMATION
        TYPE(BIEF_OBJ), POINTER :: SURDET
  !
  ! LGSEG: LENGTH OF 2D BOUNDARY SEGMENTS
        TYPE(BIEF_OBJ), POINTER :: LGSEG
  !
  ! XSGBOR: NORMAL X TO 1D BOUNDARY SEGMENTS
```

```
        TYPE(BIEF_OBJ), POINTER :: XSGBOR
!
! YSGBOR: NORMAL Y TO 1D BOUNDARY SEGMENTS
        TYPE(BIEF_OBJ), POINTER :: YSGBOR
!
! ZSGBOR: NORMAL Z TO 1D BOUNDARY SEGMENTS
        TYPE(BIEF_OBJ), POINTER :: ZSGBOR
!
! XNEBOR: NORMAL X TO 1D BOUNDARY POINTS
        TYPE(BIEF_OBJ), POINTER :: XNEBOR
!
! YNEBOR: NORMAL Y TO 1D BOUNDARY POINTS
        TYPE(BIEF_OBJ), POINTER :: YNEBOR
!
! ZNEBOR: NORMAL Z TO 1D BOUNDARY POINTS
        TYPE(BIEF_OBJ), POINTER :: ZNEBOR
!
! X: COORDINATES OF POINTS
        TYPE(BIEF_OBJ), POINTER :: X
!
! Y: COORDINATES OF POINTS
        TYPE(BIEF_OBJ), POINTER :: Y
!
! Z: COORDINATES OF POINTS
        TYPE(BIEF_OBJ), POINTER :: Z
!
! COSLAT: LATITUDE COSINE
        TYPE(BIEF_OBJ), POINTER :: COSLAT
!
! SINLAT: LATITUDE SINE
        TYPE(BIEF_OBJ), POINTER :: SINLAT
!
! DISBOR: DISTANCE TO 1D BOUNDARIES
        TYPE(BIEF_OBJ), POINTER :: DISBOR
!
! M: WORKING MATRIX
        TYPE(BIEF_OBJ), POINTER :: M
!
! MSEG: WORKING MATRIX FOR SEGMENT-BASED STORAGE
        TYPE(BIEF_OBJ), POINTER :: MSEG
!
! W: WORKING ARRAY FOR A NON-ASSEMBLED VECTOR
        TYPE(BIEF_OBJ), POINTER :: W
!
! T: WORKING ARRAY FOR AN ASSEMBLED VECTOR
        TYPE(BIEF_OBJ), POINTER :: T
!
! VNOIN: FOR FINITE VOLUMES
        TYPE(BIEF_OBJ), POINTER :: VNOIN
!
! XSEG: X COORDINATE OF FOLLOWING OR PRECEDING POINT IN THE
```

```
BOUNDARY
  ! IF IT IS IN ANOTHER SUB-DOMAIN.
        TYPE(BIEF_OBJ), POINTER :: XSEG
  !
  ! YSEG: Y COORDINATE OF FOLLOWING OR PRECEDING POINT IN THE
BOUNDARY
  ! IF IT IS IN ANOTHER SUB-DOMAIN.
        TYPE(BIEF_OBJ), POINTER :: YSEG
  !
  ! FAC: MULTIPLICATION FACTOR FOR POINTS IN THE BOUNDARY FOR
  ! DOT PRODUCT.
        TYPE(BIEF_OBJ), POINTER :: FAC
  !
  ! FOR PARALLELISM AND NON BLOCKING COMMUNICATION (SEE PARINI.F)
  !
  ! NUMBER OF NEIGHBOURING PROCESSORS (SEEN BY POINTS)
        INTEGER , POINTER :: NB_NEIGHB
  ! FOR ANY NEIGHBOURING PROCESSOR, NUMBER OF POINTS
  ! SHARED WITH IT
        TYPE(BIEF_OBJ), POINTER :: NB_NEIGHB_PT
  ! RANK OF PROCESSORS WITH WHICH TO COMMUNICATE FOR POINTS
        TYPE(BIEF_OBJ), POINTER :: LIST_SEND
  ! NH_COM(DIM1NHCOM,NB_NEIGHB)
  ! WITH DIM1NHCOM IS THE MAXIMUM NUMBER OF POINTS SHARED
  ! WITH ANOTHER PROCESSOR (OR SLIGHTLY MORE FOR 16 BYTES
ALIGNMENT)

  ! NH_COM(I,J) IS THE GLOBAL NUMBER IN THE SUB-DOMAIN OF I-TH
  ! POINT SHARED WITH J-TH NEIGHBOURING PROCESSOR
        TYPE(BIEF_OBJ), POINTER :: NH_COM
  !
  ! NUMBER OF NEIGHBOURING PROCESSORS (SEEN BY EDGES)
        INTEGER , POINTER :: NB_NEIGHB_SEG
  ! FOR ANY NEIGHBOURING PROCESSOR, NUMBER OF EDGES
  ! SHARED WITH IT
        TYPE(BIEF_OBJ), POINTER :: NB_NEIGHB_PT_SEG
  ! RANK OF PROCESSORS WITH WHICH TO COMMUNICATE FOR EDGES
        TYPE(BIEF_OBJ), POINTER :: LIST_SEND_SEG
  ! LIKE NH_COM BUT FOR EDGES
        TYPE(BIEF_OBJ), POINTER :: NH_COM_SEG
  !
  ! WILL BE USED AS BUFFER BY MPI IN PARALLEL
  !
        TYPE(BIEF_OBJ), POINTER :: BUF_SEND
        TYPE(BIEF_OBJ), POINTER :: BUF_RECVC
  ! FOR FINITE VOLUMES AND KINETIC SCHEMES
  !
        TYPE(BIEF_OBJ), POINTER :: CMI,DPX,DPY
        TYPE(BIEF_OBJ), POINTER :: DTHAUT,AIRST
  !
```

```
        **END TYPE BIEF_MESH**
  !
!=====================================================================
======
  !
```

SLVCFG

```
  !
!=====================================================================
======
  !
  ! STRUCTURE OF SOLVER CONFIGURATION
  !
!=====================================================================
======
  !
      TYPE SLVCFG
  !
  ! SLV: CHOICE OF SOLVER
        INTEGER SLV
  !
  ! NITMAX: MAXIMUM NUMBER OF ITERATIONS
        INTEGER NITMAX
  !
  ! PRECON: TYPE OF PRECONDITIONING
        INTEGER PRECON
  !
  ! KRYLOV: DIMENSION OF KRYLOV SPACE FOR GMRES SOLVER
        INTEGER KRYLOV
  !
  ! EPS: ACCURACY
        DOUBLE PRECISION EPS
  !
  ! ZERO: FOR CHECKING DIVISIONS BY ZERO
        DOUBLE PRECISION ZERO
  !
  ! OK: IF PRECISION EPS HAS BEEN REACHED
        LOGICAL OK
  !
  ! NIT: NUMBER OF ITERATIONS IF PRECISION REACHED
        INTEGER NIT
  !
      END TYPE SLVCFG
  !

!=====================================================================
```

```
=======
  !
```

BIEF_FILE

```
  !

!=====================================================================
=======
  !
  ! STRUCTURE OF FILE
  !

!=====================================================================
======
  !
       TYPE BIEF_FILE
  !
  ! LU: LOGICAL UNIT TO OPEN THE FILE
         INTEGER LU
  !
  ! NAME: NAME OF FILE
         CHARACTER(LEN=144) NAME
  !
  ! TELNAME: NAME OF FILE IN TEMPORARY DIRECTORY
         CHARACTER(LEN=6) TELNAME
  !
  ! FMT: FORMAT (SERAFIN, MED, ETC.)
         CHARACTER(LEN=8) FMT
  !
  ! ACTION: READ, WRITE OR READWRITE
         CHARACTER(LEN=9) ACTION
  !
  ! BINASC: ASC FOR ASCII OR BIN FOR BINARY
         CHARACTER(LEN=3) BINASC
  !
  ! TYPE: KIND OF FILE
         CHARACTER(LEN=12) TYPE
  !
       END TYPE BIEF_FILE
```

Allocation of structures

Once declared, **BIEF_OBJ** structures must be defined and memory for their arrays of data must be dynamically allocated. This is done by specific subroutines, depending of their type, i.e. whether they are vectors, matrices or blocks. **BIEF_MESH** structure must also be allocated.

The allocations of structures are grouped in a subroutine called **POINT_NAME** (**NAME** is the name of a TELEMAC module, for example **ARTEMIS**).

The mesh structure must be allocated first. Vectors and matrices will then be allocated with respect to that mesh.

Mesh : SUBROUTINE ALMESH

A mesh must be declared previously as a BIEF_MESH structure.

Syntax:

```
CALL ALMESH( MESH, NOM, IELM, SPHERI,CFG,NFIC,
             EQUA,NPLAN,NPMAX,NPTFRX,NELMAX,I3,I4)
```

ALMESH prepares the **BIEF_MESH** structures and fills some of them, for example it will allocate the memory for storing the component **IKLE** and will read it in the geometry file.

However not all the data structure is ready after exiting **ALMESH**. This task is carried out by the subroutine **INBIEF** which must be called later, when all the necessary data have been logged.

Arguments:

- **MESH** : The 'BIEF_MESH' structure to allocate.
- **NOM** : Fortran name of this structure in 6 characters.
- **IELM** : Element with the highest number of degrees of freedom in the mesh. **11** : only linear interpolation in 2D**12** : quasi-bubble in 2D**41** : linear in 3D with prisms
- **SPHERI** : Logical. If true, coordinates will be spherical, if not, Cartesian.
- **CFG** : Configuration. So far 2 integer values: **CFG(1)** is the storage of matrices (1: classical EBE, 3: edge-based)**CFG(2)** is the matrix-vector product (1: classical EBE, 2: frontal)These data will be used to build specific data structures relevant to every option.
- **NFIC** : Logical unit where the geometry file has been opened.
- **EQUA** : Equations to solve or calling programme in 20 characters. Up to now is only used to allocate specific arrays for Finite volumes if **EQUA="SAINT-VENANT VF"**. is used to optimise memory requirements.

Next 6 arguments are optional:

- **NPLAN** : Number of horizontal planes in 3D meshes of prisms.
- **NPMAX** : Maximum number of vertices in the mesh, in case of adaptive meshing (not implemented yet).
- **NPTFRX** : Maximum number of boundary points in the mesh, in case of adaptive meshing (not implemented yet).
- **NELMAX** : Maximum number of elements in the mesh, in case of adaptive meshing (not implemented yet).
- **I3, I4** : When present, it means that the **X** and **Y** coordinates of the mesh are in reality **X+I3** and **Y+I4**, **I3** and **I4** (integers representing a number of metres, have been removed to minimize truncation errors (see also the SELAFIN format where these two numbers are included for a geo-referenced post-processing.

Vector : ALLVEC, ALLVEC_IN_BLOCK

A vector must be declared previously as a **BIEF_OBJ** structure

Syntax:

```
CALL ALLVEC(NAT,VEC,NOM,IELM,DIM2,STATUT)
```

Arguments:

- **NAT** : Nature (1=real, 2=integer).
- **VEC** : The **BIEF_OBJ** structure to be allocated as a vector.
- **NOM** : Fortran name of vector in 6 characters.
- **IELM** : Vector discretisation type (or dimension depending on the status, see below) **0** : dimension 1, constant per element.**1** : dimension 1 linear discretisation.**10** : triangles, constant discretisation per element.**11** : triangles, linear discretisation.**12** : triangles, quasi-bubble discretisation.**40** : prism, constant discretisation per element.**41** : prism, linear discretisation.
  - **DIM2** : Second dimension of vector.
  - **STATUT**
- **0**: Any array. **ELM** is then its first dimension.**1** : Vector defined on a mesh, with no possibility of changing discretisation.**2** : Vector defined on a mesh, with possibility of changing discretisation within the limits of the memory space.

Syntax:

```
CALL ALLVEC_IN_BLOCK(BLO,N,NAT,NOM,IELM,DIM2,STATUT)
```

With **ALLVEC_IN_BLOCK**, N vectors with the same characteristics are put directly into the block **BLO**. **NOM** is then only a generic name, for example if **NOM** is **T**, the names of the vectors will be **T1**, **T2**, etc. Only the block **BLO** must be declared. **T2** will be in fact **BLO%ADR(2)%P** but can be named also **T2** if **T2** is declared as a **BIEF_OBJ** pointer and pointed to **BLO%ADR(2)%P**:

```
TYPE(BIEF_OBJ), POINTER :: T2
T2 => BLO%ADR(2)%P
```

Matrix: ALLMAT

A matrix must be declared previously as a **BIEF_OBJ** structure. We only deal with matrices of double precision numbers.

Syntax:

```
CALL ALLMAT(MAT,NOM,IELM1,IELM2,CFG,TYPDIA,TYPEXT)
```

Arguments:

- **MAT** : The **BIEF_OBJ** structure to be allocated as a vector.
- **NOM** : Fortran name of matrix in 6 characters.
- **IELM1** : Type of discretisation for rows (same convention as for the vectors).
- **IELM2** : Type of discretisation for columns.
- **CFG** : Configuration. So far 2 integer values: **CFG(1)** is the storage of matrices (1: EBE, 3: edge based)**CFG(2)** is the matrix-vector product (1: EBE, 2: frontal)
- **TYPDIA** : Diagonal type ('0' : zero, 'Q' : any, 'I' : identity)
- **TYPEXT** : Type of the off-diagonal terms ('0': zero, 'Q': any, 'S': symmetrical)

Block : ALLBLO

A block must be declared previously as a **BIEF_OBJ** structure.

Syntax:

```
CALL ALLBLO(BLO,NOM)
```

Arguments:

- **BLO** : The **BIEF_OBJ** structure to be allocated as a block.
- **NOM**: Fortran name of block in 6 characters.

In this case, we have an empty shell where we do not specify which objects have been placed in the block. A block structure can thus be used again. To fill the block, the subroutine **ADDBLO** must then be called (see paragraph A.I.4.4). The syntax will be:

```
CALL ADDBLO(BLOCK,OBJ)
```

to add a **BIEF_OBJ** structure to the block called **BLOCK**. A block can be emptied by the simple line:

```
BLOCK%N = 0
```

because the component N is the number of objects in the block.

Example

We take here the example of a double precision array called **SAMPLE**, with one dimension, and quasi-bubble discretisation. This vector will be then set to a constant value.

1) Declare the structure:

```
TYPE(BIEF_OBJ) :: SAMPLE
```

in a global declaration through a module, or locally.

2) Allocate the structure:

```
CALL ALLVEC(1,SAMPLE,'SAMPLE',12,1,STATUT)
```

3) To set the value of the vector to 5.D0 for all points of the mesh, you can then do:

```
CALL OS('X=C ',X=SAMPLE,C=5.D0)
```

which is equivalent in this case to (but the following would require declaration of integer 'I'):

```
DO I=1,SAMPLE%DIM1
   SAMPLE%R(I)=5.D0
ENDDO
```

To understand this loop, remember that **R** is the component storing the real data of vectors, and **DIM1** the size of the first dimension. However it is not mandatory to remember this if you use the functions and subroutines designed for operations on structures.

From:
http://wiki.opentelemac.org/ - **open TELEMAC-MASCARET**

Permanent link:
http://wiki.opentelemac.org/doku.php?id=programing_guide:structures_in_bief

Last update: **2014/10/10 16:01**