ZDL MANUAL

«Un transfert réussi est celui qui fait grimper la famille aux arbres»
Zoom Design Language

DESCRIPTION DES COMMANDES.

Les FM (familles) et les CL (cellules) sont designées par un nom (maximum 4 caractères)et un code (code qui doit être un entier). Si le code est omis, il prend la valeur 0(name => name–0). Pour le moment les transferts n'ont pas de nom–code.

Deux objets peuvent avoir le même nom. C'est leur emplacement dans l'arbre qui les différentiera.

Ces noms ne sont accessibles qu'à partir de la famille de travail (Working Family : WF).

Les noms ZEUS et UNIVERSE sont des mots réservés.

Il existe deux types de familles :

. Les familles de structure à base de cellules (les transferts connectent alors les cellules entre elles). (TYPE T)

. Les familles de structure à base de transferts (les cellules connectent alors les transferts entre eux). (TYPE S)

Commande :

```
SET FM TYPE: T ;    (or SFT: T ;)

SET FM TYPE: S ;    (or SFT: S ;)
```

[not yet implemented]

CREATION DES FM.

Il faut tout d'abord créer la FM ZEUS par la commande :

```
CREATE ZEUS ;
```

L'adresse de la ZEUS bank est : LZ.

Puis il faut créer UNIVERSE (pour le moment un seul UNIVERSE est permis) :

```
CREATE UNIVERSE ;
```

L'adresse de la UNIVERSE bank est : LU .

A la création de U, on met :

. le type de famille à T,

. la Working Family (WF) à LU (LZWF=LU) .

Dans le corps du texte la commande de création est:

```
CREATE FM name[–code] ;
```

Ceci crée une famille FM), de nom name[–code] (par défaut : name–0).

Cette famille est créée sous la WF; l'adresse de la FM bank est : LZFNEW . (Pour changer de WF, il faut utiliser la commande **UNDER**).

CREATION DE CL.

Commande :

```
CREATE CL name[–code],TYPE = proc [nsv];
```

Une CL de nom : name–code (name–0 si code est omis) est créée sous la WF. Cette CL fera appel au processeur de nom proc (chaîne de 5 caractères au maximum) possédant nsv variables d'état (Si nsv est omis , nsv=1)

Dans de nombreux cas, le processeur fixe lui même le nombre de variables d'état.

Apres la création, le nombre de variables d'état est dans JZSIG.

- L'adresse de la CL bank est LZE.
- Le pointeur LZCD donne l'adresse de la zone des mots de description.
- LZETA est l'adresse du ETA bank. (Les variables d'états devront être placées dans Q(LZETA+1), ... ,Q(LZETA+JZSIG)).

Par défaut les variables d'état sont initialisées à zéro.

CREATION DE TR.

Commande :

```
CREATE[–F | –Z | –U] TR,TYPE = proc [ntv] ;
```

Lorsque [–F] est omis, le sous–programme OFRAME placera le transfert sous la FM adéquate.

[–Z] créera le transfert sous ZEUS.

[–U] le créera sous l'UNIVERS

Ce TR fait appel au processeur proc (chaîne de 5 caractères) et possède ntv variables de transfert. (par défaut, ntv=1)

Souvent le processeur fixe lui–même le nombre de variable de transfert. Noter qu'il n'y a pas lieu d'initialiser ces variables.

Avec l'option –F le transfert est créé sous la WF. (Transfert force)

- LZTR est l'adresse du TR bank.
- LZTD est un entier pointant sur la zone de description de ce transfert.

VARIABLES ASSOCIEES AUX NOMS DE FM OU DE CL

Commande :

```
SET var[(int)] = name[–code];
```

Stockage dans une variable scalaire (var) ou dans un tableau (var(int)) de l'adresse de l'objet ayant pour nom : name[–code]. Cet objet doit se trouver sous la WF. Si ce n'est pas le cas on fait alors la description de l'arbre à partir de la WF pour atteindre cet objet:

```
SET var(int) = nam1[–code]. –––––––– .name[–code] ;
```

nam1[–code] est un objet de la WF; name[–code] est l'objet dont on veut stocker l'adresse.

La description du parcours peut se faire à partir de ZEUS (Z) ou de UNIVERSE (U) :

```
SET var[(int)] = Z.U.name[–code]. –––––––– .name[–code] ;

SET var[(int)] = U.name[–code]. –––––––– .name[–code] ;
```

Les variables ainsi définies permettent d'accéder directement aux adresses des CL ou des FM, sans suivre l'arbre. La WF ne change pas.

Note 1 :

Three name forms are possible :name ; name(num) ; name–code
The last one is reserved for object names :

	variable name	object name
name	X	X
name(num)	X	X
name–code		X

The two first forms may apply either to object or to variables names. In this case, the variable interpretation takes precedence. Thus, if there exist an object and a variable having the same name, the first two forms will yield the variable and the last one the object (note that the –0 suffix might be necessary).

Note 2 :

When dealing with two or three dimensional networks, the one dimensional index system used so far may become awkward. A multi–index facility is therefore provided, based on two commands:

```
DIMENSION : n1,n2 ... ;
```

sets maximum values for indices 1, 2, 3, ...

`(i1:i2: ... )` is converted to the corresponding one–dimensional index, using the dimensions set by the `DIMENSION : ...` command. This is mainly intended for cell or family naming : `CREATE CL MESH(I:J),TYPE = WALL 2;` .However notice that this expression may be used anywhere ; for instance the instruction `A(I:J)=1.` will convert to `A(I+n1*(J–1))=1.`

DEPLACEMENT DANS L'ARBRE

Commande :

```
UNDER name[–code] <       ––––––––––––––––––––       >

UNDER name(num)   <       ––––––––––––––––––––       >
```

a l'ouverture **(<)** la WF devient celle du name.

a la fermeture **(>)** retour à l'ancienne WF.

CREATION DES SV ET DES TV

Commande :

```
C–PLUG number: iv[,iv[,iv[---]]] ;

T–PLUG number: iv[,iv[,iv[---]]] ;
```

. C = création d'un Cell Plug.

. T = création d'un Transfert Plug.

. number est le numéro du Plug; il servira à les designer lors des interconnections.

. iv est le numéro de la variable d'état ou de transfert.

Cette commande doit être placée après la création d'un CL ou d'un TR. Si aucune variable de transfert ne doit être connectée, on utilise la commande :

```
T–PROBE number ;
```

CONNEXION ENTRE OBJETS (CL,TR)

Il faut auparavant avoir créé les Plugs des cellules et des transferts.

Commande :

```
JOIN [itp] TO (name [–code],[icp])[( , )[( , )]] ;

JOIN [itp] TO (name [(num)],[icp])[( , )[( , )]] ;
```

. itp = numéro du Transfert Plug (par défaut itp=1)

. name[–code]; nom de la CL.

. name[(num)]; nom de la CL ou variable associée à la CL.

. icp = numéro du Plug (par défaut icp=1) de la CL.

Si les connections CL–TR ont lieu par l'intermédiaire d'un seul Transfert Plug, on peut utiliser la commande:

```
CONNECT itv[,itv[,---]] TO (name,[icp])[(name,[icp])[(---)]] ;
```

Cette commande est équivalente à :

```
T–PLUG 1:itv[,itv---] ;

JOIN TO (name,icp)[(name,icp)[---]] ;
```

INITIALISATION DES VARIABLES D'ETAT ET DE TRANSFERT.

1) En spécifiant chaque numero de composantes du vecteur (_Eta ou _Fi):

FILL_ETA < [1] = T; [3] = 273. >;" Initialise Eta(1) avec la "
" variable T et Eta(3) avec 273."
;
fill_Fi< [1 thru 12] = Fi_zero; >;" Fi(1 à = Fi_zero"

remarque: error ou Warning si on tente de déborder des dimensions déclarées dans le processeur .

2) Initialisation de tout le vecteur

fill_Eta< [.] = H; >;" Initialise toutes les composantes à H"

3) Initialisation implicite par liste:

FILL_ETA< [.] = T_ref, Vx_surface, Vy_surface, Sel_conc, Ro_eau; >
a pour consequences:
Eta(1) = T_ref;
Eta(2) = Vx_surface;
Eta(3) = Vy_surface;
Eta(4) = Sel_conc;
Eta(5) = Ro_eau;
Vérifications de débordement et Warning si le vecteur n ' est pas entièrement initialisé.

INITIALISATION DES VARIABLES INTERNES AUX PROCESSEURS.

fill_CD «ou **fill_TD»**
< toute intruction FORTRAN;
[Var_Name {(i {,j}) }] {:Var_type} = expression;
"{ } : veut dire optionel"
>

avec : Var_Name : * Nom de la variable dans le bank CD (ou TD) (Case insensitif)
Var_Type : * I ou R (optionel) * Type de la variable. Si le type n'est pas precise, on suit la convention FORTRAN (entier de I–N, reel sinon) de la variable Var_Name.* Si ce type ne correspond pas a celui de Var_Name lors de sa declaration dans le CD ou TD bank, on envoie une bordee d'insultes et on n'ecrit pas dans la variable
i et j : * Valeur des indices. On verifie qu'il n'y a pas de depassement. Même supplice que precedemment en cas d'heresie ou de tentative quelconque de ne pas suivre La Norme.

Il est interdit d'ecrire dans une link_var ou dans une variable utilisee pour dimensionner des tableaux.

Remarque générale pour le remplissage Eta, Fi TD_bank et CD_bank:
==========

Le block < .... > est un block standard MORTRAN: toute opération y est donc possible. Par exemple, fill_Fi< [1 thru 12] = Fi_zero; >; pourrait

s'écrire:
```
     fill_Fi< <I=1,12; [I] = Fi_zero; > > ;
```

PILOTAGE DES SORTIES.

Sont implementees des facilites pour sortir l information par objet:
soit liees aux print out dans les processeurs
soit liees a OPTAU OPETA et aux matrices de FM

Options par defaut: en tete de ZINIT :

```
!   Print_out options ****************************************
[  mxdeep=1;]
PRINT_OUT TWICE EVERY 10 STEPS;
PAW_OUT ONCE EVERY 5 STEPS;
;
Zprint_options:
    Default=none Cl,none Tr,none Fm;
[ equivalent a Zprint_options: Default=none; ]
Print_options:
    Default=none;
[ Paw_options:  Default=all; ]sans consequence pour l instant.
! ****************************************************************
```

l absence totale de ces options equivaut a :
OPETA et OPTAU complets (comme avant)
mxdeep est toujours pris en compte pour sortir les matrices de toutes les Fm de profondeur au plus egale a mxdeep.

ACTIVATION DES SORTIES : deux options
1) Par objet:

```
  DO I=1,N
  <  CREATE CL P–I, TYPE = RNOD;
     .....
      SET P(I)=P–I;
      Zprint_CL;
  >
```

ou de maniere equivalente :

```
 GET P(i); Zprint_CL;

   CREATE–F TR, TYPE = CIRC 2*N;
   Zprint_TR;
```

2) Par Famille:
apres :

```
!_____________________________________
CALL OGTV;
CALL OFRAME;
PRINT *,'OFRAME–>: FREE WORDS= ',NQRESV ;
!_____________________________________

 Print all Cl of Fm: Z;
 Print all Cl of Fm: PIEC;
 Print all Tr of Fm: U;
 Print top Mx of Fm : U;
 Print all Mx of Fm : TROM;
 Print all Tr of FM : Z;
```

```
************************** Diagnostic de structure *
CALL DOPLUG(LZ) ;
  diagnostique LE bug de ZOOM


************************** RESUME FROM
  Il s agit de la reprise d un run par initialisation a partir d un
fichier
de sorties d un run precedant (qui aura evntuellement pu etre tue).
  L' initialisation ne sera faite qu a partir des etats des Cellules non
gelees.

--  on commence par changer le nom de OZDAT.CL en y ajoutant un seul
caractere:
     ex: OZDAT.CL1
-- Dans ZINIT, vers les print out every step, on ajoute l' instruction:

   RESUME FROM OZDAT.CL1, at IT;ou RESUME FROM OZDAT.CL1;
   L' initialisation sera alors faite a partir, soit du dernier vecteur d
etat,soit a partir de la ITieme sortie.

Remarques:
**********
*     Il est clair que la structure ZINIT doit etre la meme pour ce qui
concerne les cellules (sinon diagnostic et STOP 'RESUME')
      Il est cependant possible de geler ou degeler des cellules par
rapport au run de reference, d' enlever des D_Mx flag a certains
transferts ...
*     On peut utiliser une procedure analogue pour sortir la matrices
sous Zeus en faisant varier le pas de temps. L exemple suivant, de JLD,
le fait en un seul run, en arretant la progression dans le temps a
NbItStop (Inc Nb), et calcule NbItMx matrices en incrementant DTIME
(dtMx).
      Les flags ZPAW et ZENDRN(end_run) declenchent une sortie pour Visu
et la fin de run.


!======================== Zsteer possibility ====================
ENTRY ZSTEER;
!
if NbItMx.eq.0
< if (istep.ge.NbItStop)
    < ZENDRN =.true.;  >     "si dernier passage"
>else "si analyse matrice de couplage"
< if (istep.ge.NbItStop)
  < if (istep.ge.(NbItStop+NbItMx))
    < ZENDRN =.true.; >     "si dernier passage"
! pour faire l'equivalent de  resume from OZDAT.CL4;
    call ORESUME(LZ);  " !! time modifier "
    if (istep.eq.NbItStop)
    < IstepMx = 1;
      dtime = dtMx;
    > else
    < IstepMx = IstepMx+1;
      dtime = dtime + dtMx;
    >
    time = dtime ;   " Ceci permet d avoir dans visu la variation des
coefs(dtime)"
    ZPAW = .true. ;
  >else< ZPAW=.false.;>
>
RETURN ;
```

```
*   On pourrait tout aussi bien partir d un precedant run et construire
la matrice(dTMx) sans l' option NbItStop qui serait remplacee par RESUME
From OZDAT.CL1, at NbItStop
```

REMPLISSAGE D UN TABLEAU

Commande :

```
FILL   var(l)[,num]:[NO.]a1[=b1][,[NO.]a2[=b2][,---]];
```

Remplissage d un tableau sans décompte:

```
FILL q(l) : a,b,c,a+c*b ;
```

Cette instruction est remplacée par :

```
==>     q(l+1) = a
        q(l+2) = b
        q(l+3) = c
        q(l+4) = a+c*b
```

Remplissage d'un tableau avec utilisation de variables intermédiaires, mais toujours sans décompte :

```
FILL q(l) : a = 4,b = a**2,c,a+b*c ;

==>     a = 4
        q(l+1) = a
        b = a**2
        q(l+2) = b
        q(l+3) = c
        q(l+4) = a+b*c
```

Pour ne pas stocker l'une des variables intermédiaires dans le tableau, on lui accole le préfixe NO. :

```
FILL q(l) : a = 4,NO.b = a**2,c,a+b*c  ;

==>     a = 4
        q(l+1) = a
        b = a**2
        q(l+2) = c
        q(l+3) = a+b*c
```

Remplissage avec décompte :

```
FILL q(l),n : a,b,c,NO.d = a*b*c,e,d ;

==>     n = 5
        q(l+1) = a
        q(l+2) = b
        ........
        ........
```

CREATION DE BUFFER

Commande :

```
BUFFER   [I]Q(l),num:[NO.]a1[=b1][,[NO.]a2[=b2][,---]];
```

crée un bloc scratch et le remplit avec la liste qui suit les : ;

`l` et `num` sont des outputs: `l` est l'adresse du bloc, `num` est le nombre de mots.

Exemple d' utilisation:

On veut créer un bloc description pour la CL MESH–(I:J) , tout en envisageant que le nombre de mots puisse changer d'une section à l' autre :

```
BUFFER Q(LBUF),N : T,NO.P=RHO*R*T,RHO,P ;

CALL UCOPY(Q(LBUF+1),Q(LZCD+1),N) ;
```

VARIABLES UTILISABLES DANS ZINIT

```
LZ      :  address of ZEUS bank
LU      :  address of UNIVERSE bank
LZWF    :  Working Family.
LZE     :  address of last CL bank created
LZTR    :  address of last TR bank created
LZETA   :  address of ETA bank
LZFNEW  :  address of last FM bank created
LZCD    :  address of CD bank
LZTD    :  address of TD bank
```

COMMAND LIST

I)——————————————————CREATE——————————————————

```
CREATE ZEUS ;
CREATE  UNIVERSE ;
CREATE FM name[-code]  ;
CREATE[-F] TR,TYPE=proc [ntv] ;
CREATE CL name[-code],TYPE=proc [nsv]  ;
```

name: 4 character string

code: integer

Full name of an object is : name–code (code=0 if omitted).

proc: 5 character string;it is the processor name.

ntv(or nsv): integer; it is the transfer (or state) variable number to the created object.

II)——————————————————SET——————————————————

```
SET FM TYPE: fmt    ;
```

fmt : T or S (FM type)

```
SFT: FMT   ;
```

III)——————————————————SET——————————————————

```
SET   var[(int)]=name-code[.name-code[._______.]]]name-code;
```

if var : scalar variable

if var(int) : array variable; int ne 0;

name–code : name of object;code=0 if omitted.

IV)——————————————————UNDER——————————————————

```
UNDER name <  ------------------------------>
UNDER  name(num)  <  -------------------------->
UNDER  name-code  <  -------------------------->
UNDER  name-code=n1,n2< --------------------->
```

	variable name	object name
name	X	X

name(num)	X	X
name–code		X

name–int=n1,n2 : loop on the FM name–code;
code: first value n1
last value n2

V)——————————————— PLUG ———————————————

```
C–PLUG  num:isv[,isv[,....]];
T–PLUG  num:itv[,itv[,......]] ;
T–PROBE num ;
```

C: Create SV;

T: Create TV;

num: SV or TV number

isv(itv): connected state(or transfer) variable number.

VI)——————————————— JOIN ———————————————

```
JOIN [ntv] TO (namobj,[nsv])[(nameobj,[nsv])[(.......)]]   ;
```

ntv : TV number; ntv=1 if omitted.

namobj : object name or object variable

nsv : SV number; nsv=1 if omitted.

VII)——————————————— CONNECT ———————————————

```
CONNECT [itv][,itv[,..]] TO (namobj,[nsv])[(namobj,[nsv])[(..)]] ;

==>  T–PLUG 1 : itv;
     JOIN 1 TO (namobj,[nsv])[( ..................)]  ;
```

VIII)——————————————— FILL ———————————————

```
FILL q(l) : a,b,c,a+c*b  ;

==>     q(l+1)=a
        q(l+2)=b
        q(l+3)=c
        q(l+4)=a+c*b

    FILL q(l) :a=4,b=a**2,c,a+b*c  ;

==>     a=4
        q(l+1)=a
        b=a**2
        q(l+2)=b
        q(l+3)=c
        q(l+4)=a+b*c

    FILL q(l) :a=4,NO.b=a**2,c,a+b*c  ;

==>     a=4
```

```
          q(l+1)=a
          b=a**2
          q(l+2)=c
          q(l+3)=a+b*c

      FILL q(l),n : a,b,c,NO.d=a*b*c,e,d;
==>       n=5
          q(l+1)=a
          q(l+2)=b
         ........
         ........
```

IX) MISCELLANEOUS

```
GET namobj  ;
```

namobj : cell name.

```
FREEZE namobj  ;
```

namobj : cell name. ——> N–CL (no cell)

```
MOVE nameobj TO fname  ;
```

nameobj : cell name or family name

fname : family name

X) BUFFER

```
BUFFER q(l),n :a,b,c,......    ;
```

q(l) : array of ZEBRA

XI) DIMENSION

```
DIMENSION:n1[,n2[,---]];
```

n1 : valeur maximale du premier indice.

n2 : valeur maximale du deuxième indice.

Attention :

L'instruction ainsi définie est très différente de l'instruction DIMENSION Fortran.

Alors qu'en Fortran, seules les variables définies dans l'instruction DIMENSION sont concernées, ici toutes les variables le sont.

De plus, contrairement à l'instruction Fortran, l'instruction DIMENSION est ici locale: elle s'applique de l'instruction DIMENSION considérée jusqu'a la suivante.

```
(i1:i2[:--])
```

Cette expression est remplacée par : `i1+n1*(i2–1[+n2*(i3–1 ...)]`

Exemple :

```
DIMENSION:IMAX,JMAX;
CREATE CL MESH–(I:J:K), TYPE=CCV;
```

défini une maille de type CCV et de code `(I+IMAX*(J–1+JMAX*(K–1)))`