INTER

Guide de l'utilisateur

24 octobre 2012

Table des matières

Chapitre 1

INTRODUCTION

Le travail de réduction de données avec un logiciel spécialisé peut se décomposer en deux phases. L'une définit la procédure de traitement et les paramètres appropriés à celle-ci, tandis que l'autre est la réduction de données de masse proprement dite. Ces deux phases se distinguent selon le travail qu'elles demandent. L'un est typiquement interactif, l'utilisateur accède les différentes fonctions du logiciel avec une vision immédiate des résultats. L'autre peut être interactif pour les cas spéciaux mais plus généralement elle est réalisée grâce à l'emploi de procédures soumises au logiciel d'analyse.

Le logiciel INTER–MOAN a été crée pour résoudre les problèmes liés a ce type de travail et pour posséder un environnement bien adapté à sa spécialité : la réduction des images stellaires. Il se compose de deux produits distincts. L'interpréteur de commande INTER, d'utilité générale, avec son ensemble de fonctions de base et son interfaçage sur un système d'acquisition CCD, sur un système interactif de visualisation d'image, sur un système de génération de cartes stellaires, sur l'utilitaire graphique sm et sur un monochromateur (mesure de laboratoire) et MOAN, l'ensemble des programmes ou sous-programmes rattachés à l'analyse stellaire. Chacun des programmes ou sous-programmes de MOAN est une commande de INTER.

Le travail sous INTER se caractérise par l'emploi d'un bloc de données. C'est un fichier éditable qui contient toutes les variables reconnues par INTER (de type numérique réel et caractère) pouvant être passées aux commandes MOAN de manière automatique. Il peut exister plus d'un bloc. Cela permet d'exécuter plusieurs analyses de front sans mélange de variables ou de sauvegarder les paramètres liés à un travail particulier.
Un bloc est lu à l'entrée dans INTER, les variables qu'il contient peuvent être modifiées par la commande tacite "SET" ou par les commandes INTER–MOAN (résultats de calcul, mises à jour, lecture de table, ...). A la fin du travail, le bloc peut soit être mis à jour avec les modifications apportées durant la session soit rester dans son état original.

Une autre caractéristique d'INTER est la mise en commun entre INTER et les commandes MOAN de zones appelées matrices de travail. Ce sont des matrices de nombres (simple précision) pouvant être réparties en couche et dont le nombre est défini pour l'application (actuellement 22 matrices sur 5 couches).
Ces matrices sont allouées des qu'on les accède ; soit en fonction de leur taille (NX et NY), soit en fonction des données qu'elles vont accepter (image, fichier).

Chapitre 2

UTILISATION ET ENVIRONNEMENT

2.1 Généralités

Les lignes de commandes peuvent écrites en majuscule ou en minuscule sans distinction.
Une commande peut être prolongée sur plusieurs lignes en plaçant un signe moins en fin de ligne et peut contenir jusqu'à 512 caractères. Chaque éléments composant la commande (paramètre, ...) peut avoir au maximun 256 caractères.
On a accès à une centaine d'opérateurs mathématiques, fonctions numériques, astronomiques, lexiques et logiques.
Les lignes de commandes les plus récentes sont sauvegardées et peuvent être facilement rappelées ou éditées selon les standards Unix et Emacs.
Les commandes peuvent être écrites dans des fichiers (extension ".prc"), formant ainsi des procédures. Il existe 2 types de procédures :
– les macros (lancées par "@")
– les subroutines (lancées par "CALL")
On peut dès lors considérer le niveau 0 (niveau interactif) et les niveaux supérieurs. (le niveau est incrémenté lors de chaque entrée dans une macro ou une subroutine).
Les chaînes de caractères s'écrivent entre doubles guillemets. S'il n'y a pas de conflit avec une variable, s'il n'y a pas de blanc dans le texte, si le texte ne commence pas par un chiffre et enfin si le texte ne contient pas d'opérateurs mathématiques ($+, /, ...$) alors, on peut omettre les doubles guillemets.
Exemple :

```
/TITLE=NGC242
/TITLE="NGC242 filtre B"
/FMT=I4,"4I2"
```

2.2 Variables et blocs de données

Les variables existent sous 2 types :

Réel simple précision

Caractères de longueur maximum égal à 256.

Elles sont **globales** lorsqu'elles sont définies au niveau interactif et sont alors reconnues dans tout les niveaux de procédures.
Elles sont **locales** à une procédure lorsqu'elles sont définies dans celle-ci.
Il faut toutefois remarquer que les macros reconnaissent aussi les variables définis ou reconnus au niveau directement inférieur.

L'ordre de recherche d'une variable est le suivant :

1. niveau courant (variables définis dans la procédure courante)
2. niveau zéro (block de donnée et variables définis en interactifs)
3. niveaux inférieurs (si l'on se trouve dans une macro)

On défini des variables par la lecture d'un bloc de données ou avec les commandes 'LOCAL' et 'GLOBAL'. Les variables homonymes de même type sont remplacés et ceux de types différents sont interdits.

Certaines commandes d'Inter utilisent les variables de manière implicite comme arguments d'entrée ou de sortie (paramètres, résultats, ...). Ces variables sont globales et sont définies dans un bloc de données standard. Ce type de bloc est un fichier éditable, dans lequel sont rangés, par ordre alphabétique et selon un format fixe, les variables de type numérique et caractère. Un soin particulier doit être porté aux blocs dans le but de leur conserver leur intégrité. Il est absolument interdit de supprimer ou d'ajouter une ligne dans un bloc. Si malgré tout, un doute survient sur le contenu d'un bloc, il suffit de modifier la date contenue dans la variable nommée AAAAAA(1). De cette manière, Inter sait que ce bloc n'est pas à jour et le modifie si besoin dès sa première utilisation.
Il est absolument nécessaire de lire un bloc standard en entrant dans Inter.

Il existe plusieurs types de blocs de données. Il sont toujours écrit selon le même format. On les différencie selon leur nom :

'block.ref' Le bloc de référence. Il est définit à l'installation d'Inter. Il contient la liste des variables. que reconnaît Inter.

'*.blk' Bloc standard contenant les mêmes variables que 'block.ref', l'extension "blk" est l'extension ajoutée par défaut.

'*.ptl' Bloc partiel contenant une partie des variables d'un bloc standard.

'*.cte' Bloc de variables supplémentaires qui ne peuvent pas être modifiés dans Inter (read only).

'*.*' Bloc de variables supplémentaires permettant de compléter la liste actuelle ou de modifier le contenu de certains variables du bloc (initialisation personnalisée). Ces blocs servent aussi à faire des sauvetages partiels de variables.

Remarque importante : Les blocs sont reconnus uniquement par leur extension. Ceux contenant des variables inexistantes dans le bloc standard sont lus à la compilation afin que soient définies ces nouvelles variables. C'est une limitation importante, mais il faut que le nom de ce type de bloc soit déjà fixé à la compilation, par exemple, on ne peut fabriquer son nom dans une procédure.

Si un bloc non standard est lu dans une procédure, les variables qu'il définit sont locales à la procédure. Les blocs standards (block.ref et *.blk) redéfinissent toujours les variables globales, quelle que soit le niveau de procédure où il sont lu.

En entrant sous Inter, vous êtes questionné sur le nom du bloc de données standard à utiliser. Inter vous propose par défaut le bloc ayant servi à son installation "block.ref". Vous pouvez accepter ce choix si le fichier existe dans votre directory ou donner le nom d'un bloc standard existant.
Si vous n'en possédez pas il faut en récupérer un dans le directory oú se trouve Inter (actuellement /weber/src/inter).
L'extension pour le nom d'un bloc est par défaut ".blk".

Les opérations sur les blocs sont :

READ	lit un bloc (ne sauve pas le bloc courant)
SAVE	sauve le bloc courant ou une partie du bloc
SEARCH	recherche lexicale dans le bloc

Les opérations sur les variables sont :

SHOW	visualise le contenu des variables
SET	met à jour les variables (commande tacite)
GLOBAL	définition de variables globales
LOCAL	définition de variables locales
WRITE /KEYW	écriture formattée dans une variable

Un bloc standard peut être modifié en dehors d'Inter avec l'éditeur, mais les modifications ne doivent porter que sur le contenu des variables. Si l'on souhaite posséder de nouveaux variables globales en standard, il est absolument nécessaire de mettre à jour Inter (voir annexe).

Remarque : Les variables sont stockés dans une "hash table". Afin d'optimaliser leur temps d'accès, il est préférable :

1. d'utiliser des variables locales dans les procédure.
2. d'initialiser les variables les plus accédées en premier.

2.2.1 Constantes

Les variables données avec un signe égal ("=") dans le bloc de données définissent des constantes que l'on ne peut modifier interactivement. Exemple : (extrait de "bloc.ref")

```
@PI____001=     3.141592        ; constante
@VAR___001:       0.            ; variable
```

2.3 Matrices de travail

Les matrices de travail sont des tableaux à 2 dimensions de type réel simple précision. On y stocke des images, parties d'images, fichiers en colonnes, filtres, etc. C'est sur celles-ci que travaillent la majorité des commandes. On peut y appliquer toutes les fonctions numériques à dispositions.
On peut en utiliser jusqu'à un nombre maximum prédéfini, sur un nombre de couches prédéfini lui-aussi (paramètres d'installation, actuellement 22 matrices sur 5 couches).
La notion de couches existe pour faciliter l'emploi des matrices. En effet, toutes les couches attribuées à une matrice ont leurs tailles et leurs paramètres identiques.
Les matrices sont allouées ou réallouées dynamiquement lors de leur premier accès ou lorsque leur taille à changé.
La taille d'une matrice est définie par son nombre de lignes et de colonnes (NX(i) et NY(i)).
La configuration courante est affichée avec la commande MATRIX /LIST. On peut récupérer divers paramètres concernant les matrices avec les fonctions : NBPIX(), NBMAT(), NBCOU(No_mat) et MAXSIZ(No_mat).
La plupart des fonctions travaillent sur les matrices de travail. Presque toutes demandent le numéro de la matrice sur laquelle va porter le traitement (la première est prise par défaut). On reconnaît dès lors les syntaxes suivantes :

n	matrice n couche 1
[n]	matrice n couche 1
[n,m]	matrice n couche m

Exemple :

```
Inter> SUM 1     /list
Inter> SUM [1,2] /list
```

Les matrices sont mises à zéro en début de session et sont détruites en fin de session. Il est toutefois possible des les sauver sous diverses formes (voir PATCH et MAT /SAVE). Plusieurs variables les décrivent.

NX(i) NY(i)		taille d'une matrice
XSTART(i)	**YSTART(i)**	coordonnée du pixel bas gauche d'une matrice
XSTEP(i)	**YSTEP(i)**	pas selon les deux axes.
XCORI(i)		coordonnées de départ de l'image dont a été extraite la matrice.
YCORI(i)		(Mis à jour par EXTRACT)

Les opérations de base sur les matrices sont :

EXTRACT	pour le transfert depuis une image MIDAS
PATCH	pour le stockage dans une image ou sur l'afficheur
MATRIX	pour une visualisation sommaire ou sauvetage dans fichier sous divers formats
SET	pour effectuer des opérations mathématiques

La commande SET (tacite), permet d'effectuer des opérations numériques sur ou entre matrices. Elle travaille selon deux modes :

– Travail selon la coordonnée des pixels :
Dans ce mode, la trame des matrices mis en jeu doit être superposable, le pas identique et les matrices être au moins partiellement superposable.
La syntaxe est la suivante :
[$< N_1 >$**]=[**$< N_2 >$**]oper[**$< N_3 >$**]**
ou
[$< N_1 >$**]=[**$< N_2 >$**]**
De manière interne, cette commande balaye l'espace de la matrice [$< N_1 >$] et effectue l'opération "oper" entre chaque pixel des matrices [$< N_2 >$] et [$< N_3 >$] ayant la même coordonnée. Les pixels hors matrices sont posés à zéro.
[$< N_2 >$] et [$< N_3 >$] peuvent être remplacés par des constantes (variables ou nombres) mais pas par des expressions.
Les opérations à dispositions sont :

```
+, -, *, / et &
```

L'opération "&" remplace les points éliminés de la matrice [$< N_2 >$] par les valeurs de [$< N_3 >$].

exemple :

```
[1,2]=[1,2]+100
[2]=bg
[2]=[1]&[1,2]
```

– Travail sur pixel :
Dans ce cas, on ne tient pas compte des coordonnées. La syntaxe est plus complexe mais permet l'utilisation des toutes les fonctions d'Inter.
Le principe est de transformer les matrices en vecteurs avant des les traiter.

Exemple :

```
[1](:,:)=log([1,2](:,:))
```

Il faut toujours se souvenir que si plusieurs matrices (vecteurs) entrent dans une opération, ils doivent impérativement avoir le même nombre de pixels (positions).
Par exemple : échantillonnage d'une matrice.
La matrice [2] est 4 fois plus petite que la matrice [1]. On pose

```
[2](:,:)=[1](::2,::2)
```

2.4 Modes de travail

Inter travail selon 3 modes :

1. le mode **interactif** lors duquel l'utilisateur tape ses commandes au clavier. (Voir INTERPRÈTE)
2. le mode **procédure** où les commandes sont lues dans un fichier. (Voir "@" ou CALL)
3. le mode **compilation**, qui permet de d'entrer une procédure de manière interactive puis de l'exécuter. (voir COMPILE)

Ces trois modes permettent l'utilisation d'ALIAS tandis que seul les modes procédure et compilation acceptent les ordres de contrôle type IF, DO, CASE, etc .

2.4.1 Rappel d'une procédure

On peut rappeler la dernière procédure sans la recompiler au moyen de la commande EXECUTE.

2.5 Elimination des points

Les points d'une matrice de travail peuvent être éliminés selon plusieurs critères de rejet (voir la commande ELIMINATION). Les points éliminés sont généralement ignorés par certaines commandes ou peuvent être recalculés ou remplacés (voir ELIMINATION).

Chapitre 3

SYNTAXE ET RÈGLES DE GRAMMAIRE

Ce chapitre décrit les règles de syntaxe du langage de commande de l'interpréteur.

Inter reconnaît plusieurs types d'entrée selon le mode de travail :

1. en mode interactif
   - Les commandes
   - Les commandes spéciales (@, $, ?, ...)
   - L'édition et le rappel des précédentes commandes
2. en mode procédure et compilation
   - Les commandes
   - Les commandes spéciales (@, $, ?, ...)
   - Les choix d'un groupe CASE...ENDCASE
   - Les labels (points d'arrivées d'un GOTO)

3.1 Syntaxe

3.1.1 Syntaxe d'une ligne de commande

Cette syntaxe est très proche de celle de VMS sur VAX. Elle se représente sous cette forme :

```
<commande> [/<qualificateur>...] [<parametre>...] [/<qualificateur>...]
```

Le séparateur unique est l'espace. On reconnaît :

<**commande**> : c'est le nom de la commande. Il peut être abrégé.

/<**qualificateur**> : c'est un terme facultatif qui sert à modifier ou préciser le déroulement d'une commande. Un qualificateur peut posséder des arguments nommés ou non selon les syntaxes suivantes :
– /<qualificateur>[=argument[,...]]
– /<qualificateur>[=[(][option :]argument[,...][)]]
Les qualificateurs et les options peuvent être abrégés.

<**paramètre**> : c'est le paramètre de la commande. Pour certaines commandes internes on reconnaît une syntaxe plus riche :
– <paramètre>[=argument[,...]]

3.1.2 Syntaxe pour le choix d'un groupe case...ENDCASE

Ces choix s'utilisent pour les branchements dans un groupe type CASE...ENDCASE. Ils sont représentés par une valeur numérique entière pour les choix de type numérique et par une chaîne de caractères pour les choix de type caractères. Ils s'écrivent :

```
<choix>::
```

Le choix par défaut (optionnel) s'écrit :

```
DEFAULT::
```

Les commandes qui le suivent sont exécutées lorsqu'aucun autre choix n'a pu être pris.

3.1.3 Syntaxe d'un label

Les labels s'utilisent pour les branchements de type GOTO. Ils représentent des adresses symboliques et s'écrivent :

```
<label>:
```

3.2 Règles de grammaire

Ces règles décrivent le terme <paramètre> ou <argument> utilisé ci-dessus. Rapellons que l'interpréteur a accès à des variables de type caractère et des variables de type numérique.
Une commande peut demander des arguments (pour les qualificateurs ou pour les paramètres) de type caractère ou numérique. Ces arguments peuvent revêtir les formes suivantes :
– Pour les arguments de type numériques :
  – un nombre
  – une variable numérique
  – une expression numérique
– Pour les arguments de type caractères :

- une chaîne de caractères entre doubles guillemets
- une variable caractères
- une chaîne de caractères
- le résultat d'une fonction lexique

3.3 Syntaxes des objets accédés

variable numérique :

- <nom>
- <nom>(<position>)

exemples : COUNT, I(20)

variable caractère :

- <nom>
- <nom>(<position>)
- <nom>(<position>)([<début>] :[<fin>] :[<pas>])

exemples : GSTA, GSTA(1), GSTA(1)(: :2)

vecteur numérique :

- <nom>([<début>] :[<fin>])

exemples : NW(1 :N), NX(:N), NY(6 :), NZ(:)

pixel :

- <NO de matrice>(<x>,<y>)

exemple : [1,2](X,Y)

matrice :

- <NO de matrice>([<xdébut>] :[<xfin>] :[<xpas>],[<ydébut>] :[<yfin>] :[<ypas>])

exemple : [1,2](:, :) = toute la matrice
exemple : [1,2](:,2) = la ligne 2
exemple : [1,2](3, :) = la colonne 3
Les positions sont données en lignes et colonnes (origine en <1,1>)

3.4 Priorité dans le décodage

Lors de l'analyse de syntaxe, un argument est considéré selon les priorités suivantes :

1. comme chaîne de caractères s'il est entouré de guillemets.
2. comme variable numérique
3. comme variable caractères
4. comme chaîne de caractères

Chapitre 4

TRAVAIL INTERACTIF

Le mode de travail de base est interactif. C'est dans celui-ci que l'on se trouve en entrant sous Inter. Les commandes sont acceptées dès l'apparition du prompt, puis interprétées et enfin sont exécutées.

4.1 Commandes spéciales

Il y a plusieurs caractères spéciaux signifiant à Inter d'exécuter des opérations spécifiques :

@<nom du fichier>	lance une procédure
$<commande système>	lance une commande système
? ou **? ?**	help général
?<commande Inter>	help sur une commande
.<commande Inter>	affiche les keywords utilisés par une commande
? ? ?	affiche le résultat de l'analyse grammaticale de la dernière commande

Remarques :

1. SHOW /COMMAND affiche toutes les commandes à disposition.
2. les commandes ?, ? ?, ? ? ? et . ne sont pas compilables, elles s'exécutent immédiatement quel que soit le mode de travail.
3. un ou plusieurs espaces sont autorisés après un caractère spécial.

4.2 Rappel des commandes précédentes

Selon le standard Unix (! !, !n, ...), et en plus, utilisation des flèches et commandes d'édition d'EMACS.

4.3 Alias

Inter permet l'utilisation d'alias. Ce sont des chaînes de caractères synonymes de commandes ou parties de commandes, comprenant au maximum 4 caractères. (voir la commande ALIAS)

Chapitre 5

UTILISATION DES PROCÉDURES, LANGAGE DE COMMANDE

Le travail de routine nécessite l'emploi de procédures (fichiers) dans lesquelles sont écrites les commandes et les instructions de contrôle (IF, CASE, DO, GOTO, ...). Deux types de procédures existent dont l'extension est la même : ".prc"

– les procédures type **subroutine** (usage équivalent au fortran) que l'on appelle avec CALL peuvent avoir jusqu'à 19 arguments. Exemple :

```
CALL <nom de la procedure> [<argument>] ...
```

– les procédures type **macro** que l'on appelle avec "@" peuvent avoir jusqu'à 9 arguments. Exemple :

```
@<nom de la procedure> [<argument>] ...
```

Ces arguments sont considérés uniquement comme chaîne de caractères et remplacent littéralement les termes `"{n}"` (n=1 à 9) se trouvant dans le fichier. Les arguments manquants peuvent être remplacés grâce à la commande DEFAULT. Si un argument est noté "?", c'est sa valeur par défaut qui est prise en compte.

Une procédure, une fois exécutée, peut être à nouveau exécutée avec la commande EXECUTE.

Une procédure peut en appeler une autre et ainsi de suite jusqu'a une profondeur maximum prédéfinie. (Cette valeur est paramétrisable à l'installation, actuellement 6). L'exécution de la procédure se déroule en deux phases. Dans un premier temps tout les fichiers sont lus et stockés de manière séquentielle dans un seul fichier temporaire d'accès direct. En même temps les lignes sont décodées et la syntaxe vérifiée. Les commandes ne sont pas exécutées, les ordres de compilation et de branchements sont compilés. Lorsque tout les fichiers sont lus et qu'aucune erreur n'a été détectée la procédure est exécutée en fonction des ordres compilés et des commandes contenues dans le fichier temporaire.

L'allure d'un fichier de procédure doit être la suivante si on veut que ce dernier fasse partie de la documentation et soit traité de manière efficace par la commande "PROCEDURE" :

```
! Commentaire d’une ligne, description rapide de la commande
! Commentaires plus generaux
! ...
!
  Default 1="out.dat"
  Default 2=0.001
! Commentaires suplementaires facultatifs
!
! Fin des commentaires, debut des instructions

LOCAL name={1}
...
```

ou

```
! Commentaire d’une ligne, description rapide de la commande
! Commentaires plus generaux
! ...
!
  SUBROUTINE TEST  AA  BB  CC=C
! Commentaires suplementaires facultatifs
!
! fin des commentaires, debut des instructions

LOCAL name=cc
...
```

5.1 Instructions de contrôle

Inter reconnaît les structures suivantes :

5.1.1 structure IF...ENDIF

```
*  IF (test) <commande>

*  IF (test) THEN
      <commande>
      ...
   ENDIF

*  IF (test) THEN
      <commande>
      ...
   ELSE
      <commande>
      ...
   ENDIF
```

5.1.2 structure DO...ENDDO

```
*  DO <index>=<start>,<stop>[,<step>]
      <commande>
      ...
   ENDDO
```

5.1.3 structure CASE...ENDCASE

```
*  CASE <expression numerique entiere> OF
   <choix>::
      <commande>
      ...
   <choix>::
      <commande>
      ...
   ...
   [ DEFAULT::
      <commande>
      ... ]
   ENDCASE

*  CASE <chaine de caractere> OF
   <choix>::
```

```
   <commande>
   ...
<choix>::
   <commande>
   ...
...
[ DEFAULT::
   <commande>
   ... ]
ENDCASE
```

5.1.4 Branchement GOTO

```
GOTO <label>
...
<label>:
```

5.1.5 SUBROUTINE

Les subroutines sont d'un usage similaire au Fortran. Leur déclaration peut se faire a tout moment mais elle ne peut être faite qu'en début de fichier ou qu'après une fin de procédure (voir ENDPROC). Elle permettent le passage de variables **non dimensionnées** de type numérique ou caractère.

```
...
<commande>
...
CALL <nom de la subroutine> [arg [arg ...]]
...
ENDPROC


SUBROUTINE <nom de la subroutine> [arg [arg ...]]
...
RETURN
ENDPROC
```

Lors de l'appel si un argument de type numérique est écrit <arg>=V, il est passé par valeur et ne pourra pas être modifié par la subroutine.
Les arguments déclarés dans la sous-routine sont considérés par défaut comme des variables numériques. Les arguments de type caractères devront être notés : <arg>=C.
Exemple :

```
...
```

```
CALL TEST I J "/image/demo.bdf"
...

...
SUBROUTINE TEST A=V B NAME=C
...
```

5.1.6 GLOBAL, LOCAL

Ces commandes permettent de créer des variables dimensionnés de type numérique ou caractère temporaires et locales à une procédure ou à une subroutine avec **LOCAL** ou globales et accessibles à tout niveau avec **GLOBAL**. Les valeurs d'initialisations données aux variables dimensionnées sont stockées dans toutes les positions de la variable.
Si on définit une variable au niveau interactif avec LOCAL ou GLOBAL, elle est globale, au même titre que les variables du bloc.
Lorsque l'on définit une variable globale, il ne faut pas avoir défini des variables locales au préalable.
Remarque : Les variables, locales ou non, sont stockées dans une "hash table" prévue par défaut pour une centaines de variables (tailles et types confondus par niveau). Afin d'optimaliser le temps d'initialisation, si le nombre de variables locales à une procédure dépasse 100, il devient intéressant de donner le qualificateur "/N=<n>" avec n le nombre maximum de variables utilisés dans la procédure ou subroutine.
Les variables sont initialisées dans l'ordre où ils apparaissent et peuvent ainsi être utilisées pour l'initialisation des autres.

```
...
LOCAL I=4 J(I) K(I)=I  L(K(1)+1)=I+1
LOCAL USER NAME="Texte"
...
```

5.2 Contrôle du déroulement d'une procédure

La commande ECHO (ECHO /ON ou ECHO /OFF) permet l'écho des lignes de commandes lors de la compilation et de l'exécution. Par défaut il n'y a pas d'écho.

5.3 Contrôle des erreurs

La commande ERREUR (ERREUR /ON ou ERREUR /OFF) permet le contrôle de l'arrêt de la procédure en cas d'erreur dans une commande. Par défaut la procédure est stoppée à la première erreur.

5.4 Affichage de messages de test

La commande DEBUG (DEBUG /ON ou DEBUG /OFF) permet d'afficher les messages de test fournis de manière autonome par les commandes d'Inter.

La commande VERBOSE (VERBOSE /ON ou VERBOSE /OFF) permet l'affichage conditionnel des messages adressés à l'écran par la commande WRITE /VERBOSE.

5.5 Mode compilation

Ce mode permet de taper des procédures de manière interactive. On y rentre en tapant la commande COMPILATION qui sollicite l'utilisateur pour l'entrée d'une procédure avec le message "COMP >". Les commandes peuvent alors être tapées. Il y a deux commandes permettant de quitter ce mode :

1. INTERACTIF : remet le mode interactif en oubliant toutes traces de la procédure.
2. EXECUTE : termine le mode compilation et lance la procédure.

5.6 Accès aux procédures, directories

Les procédures ont une extension par défaut ".prc". Une autre extension est autorisée mais alors elle doit être précisée pour chaque appel.
Les procédures sont recherchées automatiquement et dans l'ordre dans les directories nommés dans la variable "DIRPRC".
La commande PROCEDURE permet de lister les procédures à disposition selon un wildcard et différents paramètres. Elles accède les directories nommés dans DIRPRC se trouvant avant un " :" facultatif.
Ex: DIRPRC=". prc : call"

5.7 Fichier d'initialisation

En début de programme, lors de l'initialisation, si Inter trouve un fichier nommé "login.prc", celui-ci est exécuté automatiquement. Il permet l'initialisation des matrices, alias, etc... .

5.8 HELP

Les fichiers de help sont stockés dans un directory spécifique avec une extension ".hlp". Ces fichiers sont affichés au moyen du "more" Unix. Un directory nommé dans la variable DIRHLP permet d'accéder tout autre fichier de help (ex : mémo, procédure, ...).

5.9 Mesure du temps CPU

La variable "CHRONO" est mise à jour entre chaque commande tapée en mode interactif. Elle contient la valeur de CHRONO plus le temps CPU utilisé depuis sa dernière mise à jour.

5.10 Interruption d'une procédure

Le déroulement d'une procédure peut être modifié en tapant <CTRL – C>. A ce moment là et dans tout les cas la commande en cours se termine de façon normale et seulement après cela le traitement du <CTRL – C> intervient.
Les spécifications du comportement de la procédure après un <CTRL – C> se définissent avec la commande ACTION. On peut décider en priorité que la procédure effectue un ou plusieurs RETURN successifs dans le cas d'utilisation de SUBROUTINES, ou alors que l'on termine des boucles DO...ENDDO sur un ou plusieurs niveaux d'imbrication. Dans le cas où aucune action n'est prévue, la procédure est stoppée et le contrôle rendu à l'opérateur.
Il est intéressant de noter que les variables locales définis dans la procédure courante restent accessibles (en lecture et écriture), permettant ainsi une certaine souplesse dans le "debugging". (On peut interdire leur accès avec la commande INTERPRETE).
Une procédure peut être reprise depuis l'endroit où elle a été interrompue par la commande "EXECUTE /CONTINUE".

5.11 Modification du prompt

Tout simplement, selon le contenu de la variable PROMPT.

5.12 Utilisation du Log-Book

Le Log-Book permet de receuilir des messages venant de process connectés sur cet utilitaire depuis n'importe quelle machine du réseau. Les messages sont préfixé de la date. Inter possède deux fonctions pour l'utilisation du Log-Book, inilog() pour la connection et logbk() pour l'envoi de message.
Si pour une raison quelconque le log-book est détruit et donc la connection perdue, il suffit d'envoyer le signal SIGUP à Inter pour qu'il se reconnecte, même en cours de d'exécution. Ex : `kill -1 <pid_inter>`.
La fonction inilog() peut être envoyée plusieurs fois, Inter ne se connecte que s'il se sait deconnecté.
La fonction logbk() peut être utilisée même si le log-book n'est pas lancé. Les message sont simplement perdu, sans messages d'erreur.
Exemple :

```
i=inilog("obssd7")
i=logbk("Inter: debut du travail")
```

Chapitre 6

ENTREE SORTIE

Voici les divers possibilités utilisées sous Inter pour importer des données.
Elles sont classées selon le type de données accédées.

6.1 Image Midas

La lecture est réalisé avec la commande **EXTRACT**. On peut lire des parties d'images en utilisant les qualificateurs appropriés. L'extension **"bdf"** est rajoutée par défaut.
Exemple :

```
! Lecture de l'image flat.bdf
extract flat
```

L'écriture est réalisée avec la commande **PATCH** qui écrit la matrice choisie dans l'image en tenant compte des coordonnées. On peut ainsi modifier partiellement ou complètement une image Midas. L'extension **"bdf"** est rajoutée par défaut.
Exemple :

```
! ecriture sur l'image flat.bdf
patch 1 /imag=flat
```

6.2 Table Midas

Les tables Midas sont accessibles en lecture et en écriture au moyen de la commande **TBL**. Cette commande permet de réaliser les principaux appels à l'interface des tables Midas décrit plus en détail dans le volume Midas-Environnement. L'extension **"tbl"** est rajoutée par défaut.

6.3 Descripteur Midas

Les descripteurs des images et des tables Midas sont accessibles en lecture et en écriture au moyen de la commande **DESCR**. Cette commande permet de réaliser les principaux appels à l'interface des descripteurs Midas décrit plus en détail dans le volume Midas-Environnement. L'extension **"bdf"** est rajoutée par défaut.

6.4 Image FITS

La lecture est réalisé avec la commande **EXTRACT**. On peut lire des parties d'images en utilisant les qualificateurs appropriés. L'image peut être complètement lue par la commande **FIMAGE**. Dans tout les cas, il faut préciser l'extension.
Exemple :

```
! Lecture de l'image flat.fits
extract flat.fits
! ou
fimage /read flat.fits
```

L'écriture est réalisée avec la commande **FIMAGE** qui sauve l'image en totalité sous forme réel simple précision.
Exemple :

```
! ecriture sur l'image flat.bdf
fimage /write flat.fits
```

6.5 Keyword FITS

On peut les lire et les écrire au moyen de la commande **FIMAGE**. Dans tout les cas, il faut préciser l'extension.
Exemple :

```
local content="" comment=""
! lecture
fimage image.fits /rkey="INSTRUM",content,comment
! ecriture
fimage image.fits /wkey="OBSERVER",getenv("LOGNAME"),"observateur"
```

6.6 Image Binaire

La lecture est réalisé avec la commande **MATRIX**. En utilisant les qualificateurs appropriés, il est possible de lire des réels (32 bits), des entiers (32, 16 et 8 bits) signés ou non.
Exemple :

```
! Lecture de l'image flat.float contenant du binaire reel
matrix /read=flat.float /direct /float /recl=n*4
```

L'écriture est réalisée avec la commande **MATRIX** qui sauve la matrice en totalité sous la forme désirée.
Exemple :

```
! Ecriture de la matrice 1 dans de l'image flat.byte
! sous la forme de bytes non-signés
matrix /save=flat.byte /direct /byte /unsigned
```

6.7 Image ou colonnes de nombres en ASCII

La lecture est réalisé avec la commande **MATRIX /FMT**. En utilisant le qualificateur **/COLUMNS**, on peut lire que certaines colonnes (maximum 20 noms de colonnes ou numéro de colonnes).
Exemple :

```
! Lecture de l'image flat.lis contenant des colonnes écrites en ASCII
matrix /read=flat.lis /fmt
! ou
matrix /read=flat.lis /columns=1,4
```

L'écriture est réalisée avec la commande **MATRIX /FMT** qui sauve la matrice en totalité sous forme ASCII.
Exemple :

```
! Ecriture de la matrice 1 dans l'image flat.lis
! sous forme de colonnes écrites en ASCII
matrix /save=flat.lis /fmt
```

6.8 fichier ASCII

Les fichiers ASCII (formattés) peuvent être accédés d'une manière identique au Fortran.

– On les ouvre et on leur attribue un numéro d'unité au moyen de **FILE /OPEN /UNIT**. Exemple :
```
file result.dat /open /unit=50
```
– On lit son contenu au moyen de **FREAD**. Exemple : `! pour lire une ligne complete local str="" du`
```
fread /unit=50 /line str ! pour lire la troisième valeur fread /unit=50 dummy du
! pour lire la ligne et la troisième valeur fread /unit=50 /line str dummy dummy
```
– On se déplace avec **FILE /REWIND /BACKSPACE**. Exemple :
```
file /unit=50 /rewind
```
– On écrit avec WRITE /UNIT. Exemple :
```
write i j k /unit=50
```
– On ferme le fichier au moyen de **FILE /CLOSE**. Exemple :
```
file /unit=50 /close
```

Le numéro d'unité peut être fixé au moyen de la fonction **GETLU** qui fournis une unité libre
Exemple :

```
local unit=getlu()
```

6.9 Fichier RDB

La lecture est réalisé avec la commande **MATRIX /RDB**. En utilisant le qualificateur **/COLUMNS**, on peut lire que certaines colonnes (maximum 20 noms de colonnes ou numéro de colonnes). **Attention,** la table doit contenir que des colonnes numériques. Si des colonnes contenant des chaînes de caractères sont présentes, il faut fabriquer un fichier temporaire sans celle-ci (voir rdb : column) et utiliser ce dernier.
Exemple :

```
! Lecture du fichier RDB flat.rdb
matrix /read=flat.rdb /rdb
! ou
matrix /read=flat.rdb /rdb /columns=1,4
```

L'écriture est réalisée avec la commande **MATRIX /RDB** qui sauve la matrice en totalité sous forme ASCII.
Exemple :

```
! Ecriture de la matrice 1 dans le fichier RDB flat.rdb
matrix /save=flat.lis /rdb=skel.rdb
```

6.10 Variable d'environnement

On peut récupérer les variables d'environnement déclarées avant le lancement d'Inter avec la fonction **GETENV**.
Exemple :

```
local path=getenv("HOME")
```

6.11 La command INQUIRE

Cette commande permet d'entrer des données en questionnant l'utilisateur.

Chapitre 7

MOYENS GRAPHIQUES

Inter est interfacé avec les librairies graphiques PGPLOT. Il possède différent drivers graphiques dont le nom doit être écrit dans la variable GDRIVE.

Les drivers principalement utilisés sont :

/xwin	fenêtre X
<**file**>**/ps**	sortie postscript horizontale, noir et blanc
<**file**>**/vps**	sortie postscript verticale, noir et blanc
<**file**>**/cps**	sortie postscript horizontale, couleur
<**file**>**/vcps**	sortie postscript verticale, couleur

Si la variable "GDRIVE" contient " ?", l'utilisateur reçoit une invite de PGPLOT et peut à ce moment lister le nom des drivers disponibles. Les cycles d'effacement de l'écran ou d'impression sont géré automatiquement pour ces drivers lorsque la surface graphique est entièrement remplie.

L'utilisateur a accès à deux commandes graphiques :

CONTOUR	permet de représenter une matrice en 2D ou 3D.
ONED	affiche le profil radial d'un objet.

et deux utilitaires graphiques

WRITE	pour afficher du texte.
CLEAR	pour effacer le dessin.
NEXTWINDOW	gestion automatique de la zone accessible en mode window.

7.1 Marge

La variable **GMARGE** définit la marge pour les fichiers PostScript non encapsulés. Si elle vaut zéro, il n'y a pas de marge, si elle est plus grande que zéro on pose une marge standart de 10[mm] et si elle est négative sa valeur absolue donne la marge en millimètre.

7.2 Taille de la zone graphique

Elle est retournée après un dessin (Contour, Oned) ou un effacement (Clear) dans les variables **XSIZE** et **YSIZE**, en millimètre.
On peut changer la taille de la zone graphique avec les variables **XSSIZE** et **YSSIZE**. On exprime la taille désirée en millimètre. Si au moins une des deux variables vaut zéro, la taille de la zone graphique est définie par défaut.
L'origine des coordonnée est en bas à gauche de l'écran.

7.3 Modes d'Affichages

Chaque commande graphique possède un set d'options commun qui est décrit ci-dessous.
Tout d'abord il faut préciser que les dessins peuvent être tracés selon deux modes.
– Le mode standard où toute la zone graphique est accédée (un dessin par page).
– le mode window où le dessin est placé dans une partie de la zone accessible (window).
Le mode window est activé par la présence du qualificateur **/WINDOW**. La taille et la position de la zone où viendra se loger le dessin est calculée selon le contenu des variables suivantes :

NWX, NWY	définissent un maillage dans la zone graphique sur lequel viendra se caler la window.
WDX, WDY	donnent la coordonnée dans le maillage du coin bas gauche de la window (Attention, la window <1 ;1> est en haut a gauche)
LWX, LWY	donnent la taille de la window selon le maillage.

Une fois ces variables définies, la commande NEXTWINDOW permet de passer à la prochaine fenêtre accessible horizontalement, si c'est possible, ou verticalement contre la marge sinon.
Dans le cas ou l'on redéfinis la tailles des windows (LWX, LWY), il est préférable de donner la position de la prochaine window (WDX, WDY).
NEXTWINDOW gère le passage à la page suivante dès que la dernière fenêtre accessible a été remplie.

7.4 Qualificateurs Graphiques Standards

Chaque dessin est composé d'un graphique, bordé optionellement de ticks, de graduations, de labels en X, en Y et d'un titre.

Voici la liste des qualificateurs communs aux commandes graphiques :

/WINDOW

Dessine dans la window courante.

/OVER

Dessine sans effacement préalable.

/LEFT

Dessine l'image sur la moitié gauche de la fenêtre.

/RIGHT

Dessine l'image sur la moitié droite de la fenêtre.

/NOTICK

Ne dessine pas les ticks.

/NOGRAD

N'écrit pas les graduations sur aucun des deux axes.

/NOGRAD=<**axe**>

N'écrit pas les graduations sur l'axe donné ("X" ou "Y").

/LABEL

Ecrit les labels des axes fournis par défaut.

/XLABEL

Ecrit le label par défaut pour l'axe "X".

/YLABEL

Ecrit le label par défaut pour l'axe "Y".

/ZLABEL

Ecrit le label par défaut pour l'axe "Z".

/XLABEL=<**label**>

Affiche le label désiré sur l'axe "X".

/YLABEL=<**label**>

Affiche le label désiré sur l'axe "Y".

/ZLABEL=<**label**>

Affiche le label désiré sur l'axe "Z".

/TITLE

Affiche le titre donné par défaut par la commande, centré sur le dessin.

/TITLE=<**titre**>

Affiche le titre désiré, centré sur le dessin.

/MM=$< X\_mm >$**[,**$< Y\_mm >$**]**

Donne la taille des axes en millimètres.

La taille du graphique peut être modifiée avec le qualificateur /mm=x[,y] qui donne les dimensions en x et en y. Si les axes sont proportionnels (cas de CONTOUR) la valeur en y n'est pas pris en compte. Il est à noter que les dimensions du graphique peuvent dépasser la taille de la window.

7.5 Texte

Il y a quatre types de texte sur une page graphique :
– Textes écrits par la commande WRITE
– Les titres des graphiques

– Les labels des graphiques
– Les graduations des graphiques
Chaque type de texte est caractérisé par sa taille, sa fonte, son épaisseur de trait, sa couleur et sa couleur d'arrière plan dans les variables suivantes (dans l'ordre des quatre type énuméré ci-dessus :
– la taille du caractère par GSIZ, GTSIZ, GLSIZ et GGSIZ
– la fonte par GFONT, GTFONT, GLFONT et GGFONT
– la couleur du caractère par GCOL, GTCOL, GLCOL et GGCOL
– la couleur de fond des caractères par GBCOL, GTBCOL, GLBCOL et GGBCOL
– l'épaisseur du trait par GLWIDTH, GTLWIDTH, GLLWIDTH et GGLWIDTH
Il n'y a pas de défaut pour la variable GSIZ, mais les variables GTSIZ, GLSIZ et GGSIZ sont calculées automatiquement si la variable GFLAG à une valeur différente de zéro.

Les fontes à dispositions sont choisie par un nombre :

```
1: (default) a simple single-stroke font ("normal" font)
2: roman font
3: italic font
4: script font
```

7.6 Travail sous X11

Avec le driver "/xwin", la fenêtre disparaît à chaque changement de driver.

7.7 Travail sous PostScript

Chaque dessin sort sur un fichier, dont le nom est donné avec le driver dans la variable "GDRIVE".
Aucune impression n'est effectuée, c'est l'utilisateur qui doit explicitement demander l'impression par une commande système ("lp -d" ou "lpr -P").
Tout ses fichiers peuvent être visualisée avec le "Previewer" PostScript "gs".

7.8 Connection à SuperMongo (sm)

On peut se connecter à **sm** (voir SM /CLIENT) et lui passer des commandes (voir SMONGO).
Inter prend la place de l'entrée interactive de **sm** qui fonctionne de manière habituelle.
Si l'on désire observer l'échange de commandes (Inter–sm), on peut soit utiliser DEBUG /ON ou alors lancer **sm** dans une fenêtre indépendante avant de lancer SM /CLIENT au moyen de :

```
sun> server sm
```

sm survit à une déconnection d'inter et peut être récupéré par la suite. Dans tout les cas, il faut quitter **sm** avec :

```
Inter> sm quit
```

Chapitre 8

MIDAS

Inter est interfacé avec les structures d’image MIDAS et de table MIDAS en vigueur l’ESO. Cet interface permet :

– de créer des images (voir IMAGE)
– d’extraire des images ou des parties d’image et de les placer dans les matrices de travail (voir EXTRACT)
– de stocker une matrice de travail dans une image ou une partie d’image (voir PATCH)
– de créer, lire, écrire, rechercher des informations et effacer des descripteurs d’image (voir DESCR)
– d’effectuer toutes les opérations de bases sur tables MIDAS. Ces opérations sont décrites dans le document "MIDAS Environnement" (voir TBL)

Chapitre 9

AFFICHAGE D’IMAGES

INTER peut se connecter à un logiciel de visualisation d’image indépendant.
Il se nomme "**aff**" sous SunView ou "**xaff**" sous OpenWindow (voir CLIENT /AFF). Cette interconnection permet de :

– visualiser en fausses couleurs les matrices de travail (voir PATCH)
– lire des positions à l’aide d’un curseur (voir GET)
– écrire des symboles sur l’image (voir SYMBOLE)
– placer le curseur sur une position (voir POINTE)
– de lire une portion d’image (voir EXTRACT AFF)
– de lire la table de couleur pour le dessin (voir CONTOUR /CSCALE=aff)
– effectuer divers opérations sur l’afficheur : zoom, clear, resize, ... (voir AFF)
– de télécharger des images (voir LOAD)

Seule la "**xaff**" permet une représentation sur les stations monochromes.

Chapitre 10

CARTES STELLAIRES

Inter peut se connecter au logiciel indépendant de génération de carte stellaires (voir CLIENT /GSC).
Il se nomme "**gsc**" sous SunView ou "**xgsc**" sous OpenWindow. Ce dernier a accès a une partie des informations de la base de données du Guide Star Catalog qui donne la position et la magnitude de plus de 18 millions d'objets ayant une magnitude maximum de 16.
Cette interconnection permet de donner les coordonnées du centre du champ que l'on veut voir représenté, sa taille et les magnitudes limites des objets s'y trouvant. A l'aide de la souris on peut rechercher les paramètres de chaque objet affiché (voir GSC).

Chapitre 11

COMMANDE DU TELESCOPE

La connection au système de contrôle du téléscope (PC AT) (voir CLIENT /CCD) permet de :
- pointer le téléscope (voir TELESCOPE)
- fabriquer la liste des zones à lire sur le CCD (voir MAKELIST)
- lire le CCD (voir CCD)
- commander de la bonnet (voir BONNET)
- commander de l'obturateur de la caméra (voir SHUTTER)
- vider le ccd (voir FLUSH)
- tester la mémoire du risc (voir TSTM)
- transférer des fichiers entre AT et Sun (voit FGET et FPUT)
- lire diverses températures de la caméra (voir fonction TEMP)

Chapitre 12

MONOCHROMATEUR

Avec la connection sur un monochromateur (voir CLIENT /MONO) permet différentes mesures de labo (voir MONO).

Chapitre 13

INTER ET LA COMMUNICATION INTER–PROCESS

13.1 MODE SERVEUR

Inter peut travailler en mode serveur, c'est à dire que les commandes qu'il exécute ne lui sont plus fournie par le clavier, mais envoyées par des clients (programmes indépendants) communiquant avec lui par l'intermédiaire d'une mémoire partagée, que l'on appellera bloc de communication.

Un nombre indéfini de clients peuvent entrer en communication avec Inter et partager les matrices de travail. Les clients en attente sont mis en queue avant leurs traitements. Il n'y a pas de priorité, chacun est pris dans son ordre d'arrivée. La synchronisation est réalisée au moyens de sémaphores.

On dira qu'un client prend la main lorsque qu'il est autorisé à communiquer, et qu'il rend la main lorsque que son travail de client est terminé et qu'il laisse l'accès au serveur à d'autres clients.

Lorsqu'un client veut prendre la main, il attend que le serveur soit accessible. Durant cette phase d'attente, dont la durée peut être négligeable, le client est stoppé jusqu'au moment où le serveur devient libre.

La main est rendue selon deux schémas :

1) **En mode NOWAIT,** une fois que le client à la main, il envoie une commande au serveur. La commande est immédiatement exécutée et lorsqu'elle est terminée **c'est le serveur qui rend la main.** L'avantage de cette méthode est que le client peut continuer son propre processus sans se soucier de la durée d'exécution de la procédure. Par contre, il ne sais pas si la procédure c'est bien déroulée. C'est le prochain client (ou lui-même dans le cas d'un processus bouclant) qui saura s'il y a eu une erreur durant la précédente procédure.

Dans ce mode, il faut absolument utiliser des procédures si on doit envoyer plusieurs commandes consécutives. En effet, si on envoie des commandes une à une, on prend le risque de voir se glisser les commandes d'un autre client à un moment inaproprié.

2) **En mode WAIT,** le client indique au serveur qu'il ne doit pas rendre la main en fin de travail et que **c'est le client qui rendra la main.** Dans ce mode, le client se met en attente sur le serveur jusqu'au moment où ce dernier termine. Le client à la possibilité d'exécuter du travail localement avant de se mettre en attente. Ce mode permet d'enchaîner plusieurs exécutions de procédures et de tester la réussite de chaque procédures. Ce mode

permet aussi de passer la main de client en client (processus différents). Dans ce dernier cas, c'est le dernier client qui devra **obligatoirement** rendre la main. Le client à la possibilité d'exécuter du travail localement avant d'attendre la fin de la procédure.

Pour travailler dans le mode serveur, on lance Inter de la manière suivante :

```
inter -server [-block <bloc_de_données>] [-echo]
```

sachant que
– Le bloc "block.ref" est lu par défaut si aucun bloc n'est donné.
– L'option "-echo" permet l'écho des commandes arrivant à Inter.
– Les clients et Inter peuvent être lancés dans n'importe quel ordre, les client resteront en attente jusqu'a que le Inter soit prêt à accepter les commandes.

Inter peut savoir s'il a été lancé en mode serveur ou client :

SHMSRV()
Retourne 1 si Inter a été lancé en mode serveur.

SHMCLI()
Retourne 1 si Inter a été lancé en mode client.

Un client envoie donc des commandes et peut envoyer ou recevoir des paramètres. Ces paramètres sont accessibles par un mot-clé et sont écrits dans une zone de mémoire partagée appelée ici "bloc de communication". Un mot-clé est réservé, c'est :

COMMAND doit contenir la commande que le serveur exécutera

13.1.1 Accès au bloc de communication

Inter accède les mot-clés et leurs contenus avec les fonctions suivantes :

SHMINIT()
initialise le bloc de communication en le vidant.

SHMGET(key)
lit un paramètre référencié par *key* dans le bloc de communication.nx=ator(shmget("XSIZE"))

SHMPUT(key,content)
écrit un paramètre référencié par *key* et son contenu *content* dans le bloc de communication et initialise le bloc de communication.Après cette opération, le bloc ne contient qu'un paramètre.
Exemple : `dummy=SHMPUT("XSIZE",itoa(nx))`

SHMADD(key,content)
ajoute un paramètre référencié par *key* et son contenu *content*, dans le bloc de communication .

SHMSHOW()
visualise le bloc de communication à l'écran.

Attention, avant chaque appel à un serveur, il faut impérativement commencer par initialiser le bloc de communication, soit avec SHMINIT(), soit avec SHMPUT(). En effet, le bloc se remplis de façon incrémentale, il n'y a pas de mode *remplacement*.

On remarque, dans les exemples ci-dessus, que toutes les variables sont passées sous forme de chaîne de caractère, les expressions numériques sont à formatter, soit avec la fonction **itoa()** soit avec **format()**, ces expressions peuvent être lues avec la fonction **ator()**.
– Attention il faut respecter les minuscules et majuscules pour nommer le mot-clé.
– Si pour une raison ou une autre on désire utiliser uniquement **shmadd()**, il faut initialiser le bloc avec la fonction **shminit()**.
– A part **shmget()** qui retourne toujours une chaîne de caractère, les autres fonctions retournent toujours la valeur 0.

Par exemple, un client qui veut connaître les tailles des matrices accessibles lancera une procédure contenant les commandes suivantes qui une fois exécutée auront remplis le bloc de communication. Le client pourra lire le résultat dès que Inter finis la procédure. Exemple :

```
set tmp=shmini()
do i=1,nbmat()
   set shmadd(lcat("NX",itoa(i)),nx(i))
   set shmadd(lcat("NY",itoa(i)),ny(i))
enddo
```

13.1.2 Partage des matrices

Une partie des matrices peut être mis en mémoire partagée. Pour réaliser cela, il faut poser la variable SHAMEM à 1, donner la taille désirée aux matrices destinés à la mémoire partagée dans les variables MATSIZ(i) et préciser dans MATCOU(i) le nombre de couches utilisées par chaque matrice destinée à être placée en mémoire partagée.
Ces variables doivent être définies avant d'utiliser Inter en mode server. **Il est interdit de changer la taille de matrice partagée en cours de travail.**

13.2 MODE CLIENT

Ce mode permet de se connecter sur un autre Inter travaillant en mode serveur. Cela permet de faire un client Inter à part entière ou de fabriquer un simulateur de client.

Pour travailler dans ce mode, on lance Inter de la manière suivante :

```
inter -client
```

Dans le cas ou les sémaphores ou le bloc de communication sont détruits, le client peut se reconnecter au serveur avec la fonction :

CONNECT()
Se (re)connecte sur les sémaphores du serveur courant s'ils ont été tués.

Comme tout client, cet Inter récupère la mémoire partagée comprenant les matrices de travail et le bloc de communication. Comme c'est un client, c'est à l'utilisateur de gérer tout les problèmes de synchronisation pour la communication entre ces deux processus.
Pour réaliser cela on a à disposition une série de fonctions :

D'abord, deux fonctions d'usage simple, permettant de passer des commandes au serveur dans les cas les plus courants. Attention, ces fonctions ne permettent pas de récupérer des paramètres, car le client n'a plus la main au retour de ces fonctions et donc le bloc de communication peut déjà être corrompu par un autre client. Ces fonctions sont toutefois très utiles en mode interactif. Ce sont :

SHMCMD(<cmd>[,<timeout>])
Suspend le client, envoie une commande et libère le client.Retourne un status différent de zéro si il y a eu une erreur sur la commande **précédente**. Dans ce cas (1=erreur sur serveur, 2=serveur déconnecté, 3=<CTRL>–C sur serveur, 101=SIGHUP, 102=SIGINT (<CTRL>–C), 113=SIGPIPE, 114=SIGALRM (timeout)) aucun message n'est affiché et l'erreur interne n'est pas activée
Exemple : `i=shmcmd("@qq")`

SHMCMDW(<cmd>[,<timeout_wait>[,<timeout_cmd>]])
Suspend le client, envoie une commande et attend la fin de la commande pour libérer le client.Retourne un status différent de zéro si il y a eu une erreur sur la commande en cours. En cas d'erreur (1=erreur sur serveur, 2=serveur déconnecté, 3=<CTRL>–C sur serveur, 101=SIGHUP, 102=SIGINT (<CTRL>–C), 113=SIGPIPE, 114=SIGALRM (timeout)) une message est affiché mais l'erreur interne n'est pas activée

Les autres fonctions sont des fonctions de bases qui permettent de construire tout les type de dialogues possible. Ce sont :

SHMWAIT([<timeout>])
suspend le client jusqu'à que le serveur soit prêt.(décrémente le sémaphore #0). Bloque le client si le serveur n'est pas prêt. Retourne un status différent de zéro si il y a eu une erreur sur la commande **précédente**. Dans ce cas (1=erreur sur serveur, 2=serveur déconnecté, 3=<CTRL>–C sur serveur, 101=SIGHUP, 102=SIGINT (<CTRL>–C), 113=SIGPIPE, 114=SIGALRM (timeout)) aucun message n'est affiché et l'erreur interne n'est pas activée

SHMCONT()
indique au serveur d'exécuter la commande placée dans "COMMAND" et lui signale de rendre la main.(incrémente le sémaphore #1)Le serveur se libérera par lui-même à la fin de la commande.

SHMACK()
indique au serveur d'exécuter la commande placée dans "COMMAND" et lui signale de ne pas rendre la main.(pose le sémaphore #2 à 1 et incrémente le sémaphore #1).

SHMWACK([<timeout>])
attend que le serveur ait finis (après un SHMACK()).(Attend que le sémaphore #2 soit égal à zéro). Suspend le client tant que sa tâche n'est pas terminée. Retourne un status différent de zéro si il y a eu une erreur sur la commande en cours. En cas d'erreur (1=erreur sur serveur, 2=serveur déconnecté, 3=<CTRL>–C sur serveur, 101=SIGHUP, 102=SIGINT (<CTRL>–C), 113=SIGPIPE, 114=SIGALRM (timeout)) une message est affiché mais l'erreur interne n'est pas activée

SHMFREE()
Rend la main (après un SHMWACK()).(incrémente le sémaphore #0) Apres cette commande, le serveur est accessible pour un autre client

L'utilisation des ces commandes se fait selon 2 principaux schémas :
1) Le premier ou l'on envoie une commande sans se soucier du résultat :

```
local dummy
! attend que le serveur soit prêt
dummy=shmwait()
! initialise le bloc de communication
dummy=shmini()
! y place une commande
dummy=shmadd("COMMAND","@proc")
! indique au serveur d'exécuter la commande
dummy=shmcont()
......
```

1) Le second ou l'on envoie une commande et on attend une réponse :

```
local dummy status nval
! attend que le serveur soit prêt
dummy=shmwait()
! initialise le bloc de communication
dummy=shmini()
! y place une commande
dummy=shmadd("COMMAND","@proc")
! indique au serveur d'exécuter la commande mais en
! attendant une réponse
dummy=shmack()
! en attendant on peut exécuter localement autre chose
......
! on attend une réponse et on teste sa réussite
status=shmwack()
if (status) goto erreur
! on lit un message en retour (par exemple)
nval=ator(shmget("NVAL"))
! on libere le serveur pour un autre client
dummy=shmfree()
......
```

Il faut bien prêter garde que si l'on passe outre le système de synchronisation, les valeurs que l'on va lire ou écrire dans le bloc de communication ou dans les matrices peuvent interférer avec d'autre clients utilisant eux le système de synchronisation. Ce mode de fonctionnement est à utiliser avec prudence.

De plus la taille de la mémoire partagée dépend des matrices définies en mémoire partagée de l'inter travaillant en mode serveur. Donc lors d'une connection avec un Inter client, utilisant un bloc de données contenant les paramètres de matrices différents, il faut récupérer (par procédure) les valeurs des variables du serveur. Sauver ces valeurs, sortir du client et le relancer pour récupérer les bonnes matrices. Pour réaliser cela on utilise (cote client) la procédure "getmat.prc"

13.3 COMPORTEMENT EN CAS DE PROBLÈME

Voici la liste des propriétés et comportements des clients et serveurs dans des situations extraordinaires :

– Un serveur à la propriété de pouvoir survivre à la perte de ses sémaphores (après un "ipckill" par exemple), il arrive à les refabriquer.
– Si les sémaphores sont tués deux fois de suite sans qu'aucune communication n'a eu lieu, le serveur se "suicide".
– Si un serveur est tué sans qu'il soit en communication, aucun problème.
– Si un client essaie de se connecter sur un serveur inexistant, le client est suspendu et sa commande sera

exécutée lorsque le client sera lancé.
– Si un serveur est tué pendant une communication avec un client en attente. Le client reste en attente jusqu'au moment ou le serveur est relancé.
– Si un serveur est tué pendant une communication sans client en attente. Aucun problème.
– Un client à la propriété de pouvoir se connecter à un serveur à n'importe quel moment. Il peut se déconnecter que s'il n'est pas en communication avec un serveur. Les clients peuvent ainsi être lancés et tués un nombre de fois illimité
– Si un client en attente est tué, le sémaphore ne sera pas libérer. Il faut donc le libérer avec un shmfree() sur un autre au client (voir aussi l'utilitaire IPCSTAT).

13.4 UTILISATION DE PLUSIEURS SERVEURS

Un client peut converser avec plusieurs serveurs, les serveurs de ce client peuvent être simultanément clients d'autres serveurs et de serveurs du client principal. La complexité du système est limitée que par l'imagination de l'utilisateur.
Pour mettre en oeuvre des systèmes clients–serveurs entrelacés, il faut se souvenir qu'un bloc de communication ainsi que les sémaphores qui y sont associés appartiennent au serveur. Ainsi chaque serveur a son propre bloc de communication et ses propres sémaphores. La manière de différencier plusieurs blocs de communication et sémaphores est donnée par l'attribution d'une clé (valeur numérique) différentes à chaque serveur. La syntaxe d'appel à Inter en mode serveur devient :

```
inter -server [<clé>] [autres options]
```

Attention, la clé est un entier positif arrondi à la centaine. La clé par défaut est 1000.

Le client doit donner un nom symbolique aux serveurs avec qui il veut converser ainsi que le clé du serveur. La syntaxe d'appel à Inter en mode client devient :

```
inter -C<nom_du_serveur_1> <clé_1>  [...] -C<nom_du_serveur_n>
                                          <clé_n> [autres options]
```

Attention, il ne faut pas mettre d'espace entre le `-C` et le nom du serveur.

Par exemple, si la configuration est la suivante :
– un serveur du système téléscope (nom :tele)
– un serveur du système guidage (nom :guid), ce système est aussi client de tele.
– un serveur du spectrographe (nom :elodie)

– un client principal ayant comme serveur : tele, guid et elodie

Les appels de lancements seront :

```
inter -server 1200
inter -server 1300 -Ctele 1200
inter -server 1400
inter -Ctele 1200 -Cguid 1300 -Celodie 1400
```

Les fonctions de communication restent les mêmes que pour une communication client–serveur unique. Seul un appel doit être rajouté pour permettre de sélectionner le bon serveur. Cet appel est :

SELECT(<server>)
Sélectionne le serveur sur lequel vont travailler toutes les fonctions de communication.
Exemple : `i=select("ccd")`

On peut également visualiser l'ensemble des serveurs à disposition avec :

SHOWSEL()
Affiche le nom de tous les serveurs possible ainsi que le serveur actuellement sélectionné.
Exemple : `i=showsel()`

Par exemple, pour envoyer la commande "@test" aux serveurs "tele" et "elodie", la procédure sera :

```
i=select("tele")
i=shmcmd("@test")
i=select("elodie")
i=shmcmd("@test")
```

Les utilitaires travaillant avec un serveur (voir plus bas : prompter, xdbox, ipcstat, ...) travaillent par défaut avec la clé 1000. Si un autre serveur doit être sélectionné, cela se fait avec l'option `"-k <clé>"`.

13.5 LE CLIENT EST UN SERVEUR

Dans le cas ou le client d'un serveur est lui–même un serveur, il faut prendre garde qu'en tant de serveur, il selectionne toujours son bloc de communication pour y lire les commandes qui lui sont adressée. S'il recoit une

commande de communication, il va l'envoyer sur son propre bloc (justement a cause de la selection courante !) et devenir son propre client (situation blocante). Dans ce cas il est necessaire de lui envoyer les commandes de communication dans une procédure contenant l'ordre de selection du destinataire. Par exemple :

```
sh select("spectro")
sh shmcmdw("entrysh /open")
```

13.6 UTILISATION SIMULTANÉE DE PLUSIEURS GROUPES CLIENT–SERVEUR

Pour permettre plusieurs groupes client–serveur de cohabiter sur une même machine, il faut que chacun d'eux utilise une zone de mémoire partagée individuelle.
La manière de différencier les zones de mémoire partagée est donnée par l'attribution d'une clé (valeur numérique) différentes à chaque groupe. Attention, la clé est un entier positif. La clé par défaut est 1000.
La syntaxe d'appel à Inter devient :

```
inter -mem <clé> [autres options]
```

13.7 CONTRÔLE DES ERREURS EN MODE CLIENT–SERVEUR

Un client récupère un status après chaque communication avec un serveur. Ce status vaut :

1. si une erreur s'est produite sur le serveur (à la compilation ou à l'exécution).
2. Si le serveur a été détruit durant l'exécution d'une commande
3. Si l'exécution de la commande a été interrompue par un <CTRL>-C.
4. Si une attente sur le client a été interrompue par un time-out ou un <CTRL>-C.

Dans tout les cas aucune erreur n'est générée au niveau du client. C'est donc la procédure qui doit gérer ces cas.

13.7.1 Interruption d'un serveur

Si un client détecte une erreur dans un de ces serveurs, il peut décider d'interrompre le déroulement des opérations sur une partie (ou la totalités) des serveurs. Dans ce cas, il envoie un signal d'interruption avec la fonction

SIGNAL() (<CTRL>-C) aux serveurs choisis. Ceux qui sont en cours d'exécution s'arrêteront à la fin de la commande en cours, les autres restant dans leur états d'attente.
Par exemple, pour interrompre les clients "tele" et "guid" on tapera :

```
i=select("tele")
i=signal(2)
i=select("guid")
i=signal(2)
```

13.7.2 Interruption d'un client

Pour interrompre un client, un serveur doit simplement terminer son exécution et signaler une erreur. Cette situation est gérée dans la majorité des cas par Inter d'une manière standard. Dans le cas où une situation d'urgence nécessite la fabrication d'une erreur, on utilise la commande "ERREUR /SET <message>".
Par exemple, si on considère un serveur A qui a un problème (ex : faute durant la compilation), avec en attente sur lui un client B et que ce client B soit en même temps le serveur d'un client C. L'erreur qui survient sur le serveur A va monter de manière automatique vers le client B qui doit la détecter et la faire remonter ver le client C. Le code dans la procédure tournant sur B ressemble à :

```
i=shmcmdw("@procedure_pour_serveur_A")
if i.gt.0 erreur /set "Probleme dans le serveur A"
```

Si le client C qui est en attente sur B a des serveurs XX et YY, et qu'il détecte une erreur, il va interrompre ses 2 serveurs. Le code tournant sur C ressemble à :

```
i=shmcmdw("@procedure_pour_serveur_B")
if i.gt.0 then
  i=select("XX")
  i=signal(2)
  i=select("YY")
  i=signal(2)
  erreur /set "Probleme dans le serveur B, arret general"
endif
```

13.7.3 Interruption d'un client lors d'une attente, Time–Out

Lors d'une interruption survenant lors d'une attente sur un serveur la synchronisation peut être perturbée. Par exemple, le client peut être interrompu à cause d'une time–out alors que le serveur est valide, mais ralenti ou stoppé.
Pour effectuer une resynchronisation automatique, la technique est la suivante :

```
i=shmwack(timeout)
if (i.eq.4) then
    if srvwait() status=shmfree()
    erreur /set "TIME-OUT"
endif
```

La fonction srvwait() indique si le serveur est toujours croché. Cela evite de libérer le serveur si ce n'est pas nécessaire.

13.7.4 Fonctions utilitaires

Quelques fonctions permettent de connaître l'état du système ou de le modifier. Ce sont :

SHMNCNT()
: Retourne le nombre de client en attente sur le serveur courant..

CLEARSV()
: Efface les flags d'erreur et de status ainsi que les code et message d'erreur ainsi que le texte de la commande courante dans le bloc de communication.Cette fonction permet de ne pas récupérer d'erreur trainant d'une précédente commande. Si on pose que l'on part d'une situation stable.

SHMZERO()

Met le semaphore zéro à 1 (reset)..

13.8 UTILITAIRES

Ces utilitaires tournent sous Unix, et ont une interaction avec Inter.

13.8.1 PROMPTER

On peut a tout moment utiliser le programme **prompter** qui se connecte à un Inter travaillant en mode serveur et permet de lui envoyer des commandes. Il permet le rappel des commandes comme dans Inter lorsqu'il travaille en mode interactif.

On sort du prompter avec la commande "bye". Les commandes "quit" ou "exit" terminent Inter.

La syntaxe d'appel est :

```
prompter [-k <clé>]
```

La clé permet de choisir un serveur pour les cas traités dans les sections "Utilisation de Plusieurs Serveur" et "Utilisation Simultanée de Plusieurs Groupes Client-Serveur".

Les commandes suivantes simulent un Inter travaillant en interactif, alors que le passage des commandes se fait par la mémoire partagée.

```
sun> inter -server &
sun> prompter
```

13.8.2 XDSEND

Cet utilitaire permet de lancer une commande à inter depuis un shell ou un script.
La syntaxe est la suivante :

```
xdsend options ...
```

Avec les options suivantes :

-k <**clé**>	La clé permet de choisir un serveur pour les cas traités dans les sections "Utilisation de Plusieurs Serveur" et "Utilisation Simultanée de Plusieurs Groupes Client-Serveur".
-f <**command**>	donne la commande que l'on passe à Inter
-s <**script**>	donne une commande système exécutée apres la commande passée à Inter s'il n'y a pas eu de problème.
-E <**erreur**>	donne une commande système ou le nom d'un script que l'on exécute si une erreur s'est produite dans la commande Inter. Dans ce cas, xdsend ne rend pas la main, ce rôle est laissé à cette commande.
-c	indique que xdsend est un client à part entière et qu'il attend d'avoir la main avant de se créer
-h	indique que xdsend rend la main. Ne rend pas la main si une erreur s'est produite dans la commande Inter.
-v	mode verbose : indique ce que "xdsend" comprend de la ligne de commande.
-e	mode echo : affiche toutes les opérations liées à la communication.
-b	émet un beep s'il y a eu une erreur dans la procédure Inter

Exemple :

```
xdsend -v -e -f "@inigen" -c -s "fix" -h -k 1400
```

13.8.3 PANEL

Cet utilitaire permet de placer Inter en attente de la réponse d'un utilisateur. C'est l'équivalent du *INQUIRE* d'Inter.
La syntaxe est la suivante :

```
panel options ...
```

Avec les options suivantes :

-t <**texte**>	texte de la question
-h <**fichier**> **:b**	nom d'un fichier qui est lu afin de créer le header du panel. L'option "b" indique que le texte est à afficher en bold.
-f <**fichier**> **:b**	nom d'un fichier qui est lu afin de créer le footer du panel. L'option "b" indique que le texte est à afficher en bold.
-label <**label**>	label de la fenêtre
-a	ajuste la taille de la fenêtre à son contenu
-geometry <**width**>**x**<**height**>**+**<**x**>**+**<**y**>	largeur de la fenêtre et position
-x <**x offset**>	position de la question selon X
-y <**y offset**>	position de la question selon Y
-s <**y ncar**>	nombre de caractères pour la réponse
-font <**font**>	nom de la font OpenWindows
...	Options standards OpenWindows (voir man xview)

L'utilisation standard est la suivante :

```
local ans=""
ans=system("panel -t""Voulez vous garder le résultat [o/n]"" -s 80 -a")
```

13.8.4 IPCSTAT

Cet utilitaire permet de monitorer l'état des sémaphores de communications

La syntaxe est la suivante :

```
ipcstat [options ...]
```

avec l'option :

-k <**nom_symbolique**> <**clé**> La clé permet de choisir un serveur pour les cas traités dans les sections "Utilisation de Plusieurs Serveur" et "Utilisation Simultanée de Plusieurs Groupes Client-Serveur". On met autant de fois l'option -k qu'il y a de clés.