INTER

Fonctions

24 octobre 2012

Table des matières

Chapitre 1

FONCTIONS

OPERATIONS

+	addition.
−	soustraction.
∗	multiplication.
/	division.
∗∗	exponentiation.

OPERATIONS DOUBLE PRÉCISION

DPLUS(x,y).....	Addition. Les arguments sont numériques ou caractères, le résultat est numérique en simple précision.
DMINUS(x,y).....	Soustraction. Même remarque que pour DPLUS.
	Ex: `dminus(jd(),"2400000") = 52534.500`
DMUL(x,y)	Multiplication. Même remarque que pour DPLUS.
DDIV(x,y)	Division. Même remarque que pour DPLUS.

OPERATIONS SUR NOMBRES DONNES SOUS FORME DE CHAINE DE CARACTERES

DCPLUS(x,y)	Addition. Les arguments sont caractères, le résultat est caractère.
DCMINUS(x,y)	Soustraction. Même remarque que pour DCPLUS.
DCMUL(x,y).....	Multiplication. Même remarque que pour DCPLUS.
DCDIV(x,y).....	Division. Même remarque que pour DCPLUS.

FONCTIONS MATHS

ABS(x)	retourne la valeur absolue de *x*.
EXP(x)	retourne l'exponentiel de *x*.
INT(x)	retourne la valeur entière de *x*.
LOG(x)	retourne le logarithme naturel de *x*.
LOG10(x)	retourne le logarithme base 10 de *x*.
MAX(x1,x2,...xn)	retourne le maximum parmis les argument *x1...xn*. Si un des arguments est un vecteur, la maximisation est faite pour chacun de ses élements et le résultat est un vecteur **Ex:** `[1](:,:)=max([1](:,:),maxval)`
MIN(x1,x2,...xn)	retourne le minimum parmis les argument *x1...xn*. Si un des arguments est un vecteur, la minimisation est faite pour chacun de ses élements et le résultat est un vecteur
MOD(x,y)	retourne le modulo de *x* par *y*.
NINT(x)	retourne la plus proche valeur entière de *x*.
RAND(flag)	retourne un nombre aléatoire. Pour $flag = 0$ on a le prochain nombre, pour $flag = 1$ on redémarre le générateur et pour $flag = n$ on initialise le générateur en fonction de "n".
SQRT(x)	retourne la racine carrée de *x*.

FONCTIONS VECTEURS

VEC(arg1,arg2,...argn) . .	fabrique un vecteur en joignant tous les arguments. **Ex:** `ny(1:5)=vec(1,2,3,nx(1:2))`
SETV(start,stop[,step]) . .	remplis un vecteur selon les paramètres de boucle. **Ex:** `setv(1,5) = 1,2,3,4,5`
EXPAND(val|vec, mul [,size])	expand chaque position d'un vecteur ou d'une matrice. Le facteur multiplicatif est donné par "mul". Si "mul" est donné négatif, cette fonction fait une compression des pixels (moyenne). Si l'on travaille en 2D (matrice), il faut préciser la nombre de colonnes avec "size". **Ex:** `[2](:,:)=expand([1](:,:),2,nx)`

INMAXV(vec) . . . . . retourne l'index du premier plus grand élément de *vec.*

Ex: `inmaxv(vec(2,4,6,8,4,2)) = 4.0000000`

INMINV(vec) . . . . . retourne l'index du premier plus petit élément de *vec.*

Ex: `inminv(vec(2,4,6,8,4,2)) = 1.0000000`

MINV(vec) . . . . . retourne le plus petit élément de *vec.*

Ex: `minv(vec(2,4,6,8,4,2)) = 2.0000000`

MAXV(vec). . . . . retourne le plus grand élément de *vec.*

Ex: `maxv(vec(2,4,6,8,4,2)) = 8.0000000`

SUM(vec). . . . . retourne la somme des valeur contenus dans *vec.*

Ex: `sum(vec(2,4,6,8,4,2)) = 26.000000`

MEDIAN(vec) . . . . . retourne le median (nag :g07daf)

MEDDEV(vec). . . . . retourne le sigma median (nag :g07daf)

STDDEV(vec). . . . . retourne le sigma (nag :g07daf)

XBARY([mat]) . . . . . retourne le x barycentre

YBARY() . . . . . retourne le y barycentre (il faut utilise xbary(mat) avant)

CLEAN([mat],seuil,size,valrep) pour une matrice centree de cote=2*size+1 contenant valrep pour chaque poinr plus eleve que seuil

Ex: `pour les cosmiques : clean([1],60000,1,0)`
`defaut : 60000,1,0`

FONCTIONS TRIGOS

COS(alpha) . . . . . retourne le cosinus de *alpha* donné en radian.

COSD(alpha). . . . . retourne le cosinus de *alpha* donné en degré.

SIN(alpha) . . . . . retourne le sinus de *alpha* donné en radian.

SIND(alpha) . . . . . retourne le sinus de *alpha* donné en degré.

TAN(alpha)	retourne la tangente de *alpha* donné en radian.
TAND(alpha)	retourne la tangente de *alpha* donné en degré.
ACOS(x)	retourne l'arc cosinus de *x* en radian.
ACOSD(x)	retourne l'arc cosinus de *x* en degré.
ASIN(x)	retourne l'arc sinus de *x* en radian.
ASIND(x)	retourne l'arc sinus de *x* en degré.
ATAN(x)	retourne l'arc tangente de *x* en radian.
ATAND(x)	retourne l'arc tangente de *x* en degré.
ATAN2(y,x)	retourne l'arc tangente selon *x,y* en radian.
ATAN2D(y,x)	retourne l'arc tangente selon *x,y* en degré.
DEG2RAD(x)	conversion degrés radians.
RAD2DEG(x)	conversion radians degrés.

FONCTIONS LEXIQUES

ATOR(str)	retourne le nombre écrit dans *str*. **Ex:** `ator("12.2") = 12.200000`
CHAR(code)	retourne le caractère ASCII donné par *code*. **Ex:** `char(48) = "0"`
FORMAT(val|vec|str, fmt)	retourne *val*|*vec*|*str* formatté le format *fmt*. Le format suit la syntaxe fortran. **Ex:** `format(nx(1:3),"3i4") = " 100 100 100"`
ICHAR(str)	retourne le code ASCII du premier caractère de la chaîne *str*. **Ex:** `char("abc") = 97.000000`
COMPAC(str)	retourne *str* sans espace. **Ex:** `compac(" a bb ccc ") = "abbccc"`
STRIP(str)	supprime les blancs, les 0 non significatifs et le point non significatif sur une chaine numérique.

Ex: `strip(" 123.00000 ") = "123"`

INDEX(str1,str2) . . . . . retourne la position de *str2* dans *str1.*

Ex: `index("file.ext",".") = 5.0000000`

FIND(str1,str2) . . . . . retourne 1 si *str2* existe dans *str1.*

Ex: `find("file.ext","ext") = 1.0000000`

ITOA(val) . . . . . retourne le nombre *val* formatté en entier sans blanc.

Ex: `itoa(1234.56) = "1234"`

LCAT(str1, ..., strN) . . . . . retourne la concaténation des tous les arguments (type caratère) .

Ex: `lcat("file",".","ext") = "file.ext"`

LSCAT(str1,str2,...strN) . retourne la concaténation des tous les arguments (type caratère) en plaçant un espace entre chaque argument.

Ex: `lscat("filea","fileb","filec") = "filea fileb filec"`

LOWER(str) . . . . . retourne *str* mis en minuscule.

Ex: `lower("AbCdEF-1234") = "abcdef-1234"`

UPPER(str) . . . . . retourne *str* mis en majuscule.

Ex: `upper("AbCdEF-1234") = "ABCDEF-1234"`

PAD(str) . . . . . retourne *str* sans les espaces en début et fin de chaîne et avec une seule barre de soulignement ("_") pour chaque suites d'espaces entres les mots.

Ex: `pad(" a bb ccc ") = "a_bb_ccc"`

PARSE(str,No) . . . . . retourne le champ *No* de *str*, avec pour séparateur le premier caractère de *str*.

Ex: `parse("/ccd2/t4/beta",3) = "beta"`

SPARSE(str,No) . . . . . retourne le champ *No* de *str*, avec pour séparateur l'espace.

Ex: `sparse(" a bb ccc ",2) = "bb"`

TPARSE(str,No) . . . . . retourne le champ *No* de *str*, avec pour séparateur le tabulateur.

TRIM(str) . . . . . retourne *str* sans les espaces avant et apres le texte utile.

Ex: `trim(" a bb ccc ") = "a bb ccc"`

LTRIM(str) . . . . . retourne *str* sans les espaces à gauche du texte utile.

Ex: `ltrim(" a bb ccc ") = "a bb ccc "`

RTRIM(str) . . . . . retourne *str* sans les espaces à droite du texte utile.

Ex: `rtrim(" a bb ccc ") = " a bb ccc"`

ASSIGN(str[,prefix]) . . . . assignation de variables selon une chaine formatée contenant une suite variable, contenu. On prefixe le nom de la variable si prefix est donnée

Ex: `assign("/texp/100/sn/20","e.")`

LEN(str) . . . . . retourne le nombre de caractères dans *str*.

Ex: `len("file.ext") = 8.0000000`

OPERATEURS LOGIQUES

.AND. . . . . ET logique.

Ex: `if (flag.and.test) then`

.NOT. . . . . négation de la condition.

Ex: `if (.not.flag) then`

.OR.	OU logique.
.EQ.	égalité numérique.
.NE.	non égalité numérique.
.GE.	plus grand ou égal.
.GT.	strictement plus grand.
.LE.	plus petit ou égal.
.LT.	strictement plus petit.

FONCTIONS LOGIQUES

AND(a,b) . . . . . retourne le résultat du 'and' binaire

Ex: `and(7,2) = 2.0000000`

OR(a,b) . . . . . retourne le résultat du 'or' binaire

Ex: `or(7,2) = 7.0000000`

XOR(a,b) . . . . . retourne le résultat du 'xor' binaire

Ex: `xor(7,2) = 5.0000000`

NOT(a) . . . . . retourne le résultat du 'not' binaire

Ex: `not(7) = -8.0000000`

LEQ(str1,str2)	retourne 1 si *str1* $=$ *str2*.
LCEQ(str1,str2)	retourne 1 si *str1* $=$ *str2*. Cette fonction est insensible aux minuscules et aux majuscules
LNE(str1,str2)	retourne 1 si *str1* $\neq$ *str2*.
LGE(str1,str2)	retourne 1 si *str1* $\geq$ *str2*.
LGT(str1,str2)	retourne 1 si *str1* $>$ *str2*.
LLE(str1,str2)	retourne 1 si *str1* $\leq$ *str2*.
LLT(str1,str2)	retourne 1 si *str1* $<$ *str2*.
LNO(str)	retourne 1 si *str* vaut "n", "no" ou "non".

LYES(str) . . . . . retourne 1 si *str* vaut "y", "yes", "o" ou "oui" .

ISNUM(xxx) . . . . . retourne 1 si "xxx" est un nombre (lexicalement parlant).

Ex: `isnum("1.2") = 1.0000000`

ISVNUM(xxx) . . . . . retourne 1 si "xxx" est une variable numérique.

Ex: `isvnum(nx(2)) = 1.0000000`

EXIST(fichier) . . . . . retourne 1 si le *fichier* existe.

Ex: `exist(lcat(getenv("HOME"),"/.login")) = 1.0`

FONCTIONS ASTRO

TIME() . . . . . donne l'heure de la machine (heures décimales).

Ex: `time() = 11.657222`

TS([tcl][,[j][,[m][,[a]]]]) . donne l'heure sidérale. Les arguments optionnels (heure, jour, mois, année) sont pris par défaut à l'instant courant, l'heure est donnée en heure décimale. Les variables LONGITU et FUSEAU sont utilisées pour le calcul, elles donnent la longitude du lieu et le décalage horaire par rapport a Greenwich (compté positif vers l'est).

Ex: `ts() = 9.7042542`

JD([j][,[m][,[a]]]) . . . . . retourne le jour julien formatté (1x,f11.3,1x). Il est alculé à partir des arguments optionnels (jour,mois,année) pris par défaut à l'instant courant. Remarque : le jour peut être fractionnaire.

Ex: `jd() = " 2452534.500 "`

JD0IN([j][,[m][,[a]]]) . . . initialise le jour julien de référence (jd0) pour les calculs de précéssion et de nutation. Ce calcul est fait à partir des arguments optionnels (jour,mois,année) pris par défaut à l'instant courant. Remarque : le jour peut être fractionnaire. Cette fonction retourne toujours 0.

PRECIN([j][,[m][,[a]]]) . initialise les variables internes pour le calcul de la précession. La précéssion se fera à la date donnée calculée à partir des arguments optionnels (jour,mois,année) pris par défaut à l'instant courant. Remarque : le jour peut être fractionnaire. Cette fonction retourne toujours 0.

APREC(alpha,delta)	précessionne la coordonnée donnée en alpha. Les coordonnées sont données en degré décimaux. JDOIN() et PRECIN() doivent être utilisés au préalable. On utilise DPREC pour la précéssion en delta.
DPREC(alpha,delta)	précessionne la coordonnée donnée en delta. Les coordonnées sont données en degré décimaux. JDOIN() et PRECIN() doivent être utilisés au préalable. On utilise APREC pour la précéssion en alpha.
NUTIN([j][,[m][,[a]]]) . .	initialise les variables internes pour le calcul de la nutation. La nutation se fera à la date donnée calculée à partir des arguments optionnels (jour,mois,année) pris par défaut à l'instant courant.Remarque : le jour peut être fractionnaire. Cette fonction retourne toujours 0.
ANUT(alpha,delta)	nute la coordonnée donnée en alpha. Les coordonnées sont données en degré décimaux. JDOIN() et NUTIN() doivent être utilisés au préalable. On utilise DNUT pour la nutation en delta.
DNUT(alpha,delta)	nute la coordonnée donnée en delta Les coordonnées sont données en degré décimaux. JDOIN() et NUTIN() doivent être utilisés au préalable. On utilise ANUT pour la nutation en alpha.
AHDE2AZ(angle_horaire, delta)	calcul l'azimut. Les coordonnées sont données en degré décimaux. La variable LATITU est utilisée, elle donne la latitude du lieu. Convention : azimut=0 au Nord, croissant vers l'Est
AHDE2EL(angle_horaire, delta)	calcul l'élévation. Les coordonnées sont données en degré décimaux. La variable LATITU est utilisée, elle donne la latitude du lieu.
AZEL2DE(azimut, élévation)	calcul delta. Les coordonnées sont données en degré décimaux. La variable LATITU est utilisée, elle donne la latitude du lieu. Convention : azimut=0 au Nord, croissant vers l'Est
AZEL2AH(azimut, élévation)	calcul angle horaire. Les argument sont donnés en degré décimaux. La variable LATITU est utilisée, elle donne la latitude du lieu. Convention : azimut=0 au Nord, croissant vers l'Est
FZ(dist_zénitale)	calcul de la masse d'air (voir slalib).
REFCO(tdk, pmb, rh, [wl], [eps], [tlr], [hm], [phi])	calcul des parametres AREFRA et AREFRA du modèle de réfraction atmospherique. AREFRA et AREFRA sont des variables du bloc de données.
REFRAC(dist_zénitale) .	calcul de la réfraction atmosphérique pour La Silla.
ROTPOS(alpha,delta) . . .	angle du derotateur moment courant. Remarque : utilise longit et latitu.
ROTVEL(alpha,delta) . . .	vitesse du derotateur moment courant [deg/s]. Remarque : utilise longit et latitu.
ROTHPOS(ah,delta)	angle du derotateur. Remarque : utilise longit et latitu.
ROTHVEL(ah,delta)	vitesse du derotateur [deg/s]. Remarque : utilise longit et latitu.

ANGLES ET HEURES

ANGLE("angle") . . . . . retourne la valeur numérique d'un angle donné sous forme de chaîne de caractères. L'angle peut être donné sous n'importe quelle forme. Si aucune unité n'est donnée, cette fonction comprend des degrés.

Ex: `"3D12", "15 degre", "12h 15.4", "123sec", "12:23:22", "12h23m12.2"`

DDTOD(angle) . . . . . formatte *angle* en : "SDDDd MMm SSs ".

DDTOD2(angle) . . . . . formatte *angle* en : "SDDD:MM:SS ".

HTOHD("heure") . . . . . retourne la valeur numérique d'une heure donnée sous forme de chaîne de caractères. L'heure peut être donnée sous n'importe quelle forme. Si aucune unité n'est donnée, cette fonction comprend des heures. Exemple de format : "3D12", "15 degre", "12h 15.4", "123sec", "12:23:22", "12h23m12.2"

Ex: `angle("12 12") = 12.200000`

HDTOH(heure) . . . . . formatte *heure* en : "HHh MMm SS.Ss ".

Ex: `hdtoh(1.234567) = " 1h 14m 04.4s "`

HDTOH2(heure) . . . . . formatte *heure* en : "HH:MM:SS.S ".

Ex: `hdtoh2(1.234567) = " 1:14:04.4 "`

MATRICES

MAXSIZ(No_mat) . . . . . retourne la taille actuelle ou maximum (dans le cas d'une matrice en mémoire partagée) de la matrice No_mat.

NBCOU(No_mat) . . . . . retourne le nombre de couche pour la matrice *No_mat.*

Ex: `nbcou(1) = 20.000000`

NBMAT() . . . . . retourne le nombre de matrices accessibles.

Ex: `nbmat() = 22.000000`

NBPIX() . . . . . retourne le nombre de pixels alloués par l'ensemble des matrices en mémoire partagée.

NOCOU(str|num) . . . . . retourne le No de couche d'un No de matrice.

Ex: `nocou([1,3]) = 3.0000000`

NOMAT(str|num) . . . . . retourne le No de matrice d'un No de matrice.

Ex: `nomat([1,3]) = 1.0000000`

FIT

FITGAU(vec) . . . . . Fit une gaussienne sur les données de *vec*. retourne 4 paramètres plus le résidu, dans un vecteur 5 positions.

GENGAU(vec, nb_points) . . . . . génère une gaussienne selon les 4 premières valeurs du vecteur *vec*. (1=max, 2=centre, 3=forme, 4=background

Ex: `gengau(fitgau([1](34:46,1)),13)`

LSFIT(No_de_matrice) . . résoud m équations à n inconnues. Travaille sur une matrice dont chaque ligne représente une équation (résultat en dernière colonne). Retourne un vecteur de taille (NX-1).

POLYF(xvec, yvec, degre [,pvec]) . . . . . fit polynômial à une inconnue de degré *degré*. Les points peuvent être pondéré par le vecteur pvec (attention pvec(i)>0). Retourne un vecteur de taille (DEGRE+1). Remarque : si *degré* est donné négatif, on donne à l'écran des messages concernant les résidus.

GENF(xvec, polyvec) . . . fabrique f(xvec) avec polyvec comme coefficient du polynôme. Le calcul est effectué en double précision. Polyvec est vecteur de coefficents donné dans l'ordre des puissance à partir du degré zero.

DIVERS

APPNAME() . . . . . retourne le nom de l'application.

Ex: `appname() = "inter"`

ENVDEF(evar) . . . . . retourne 1 si la variable d'environnement est definie, 0 sinon.

Ex: `envdef("HOME") = 1`

GETENV(evar) . . . . . retourne le contenu de la variable d'environnement *evar.*

Ex: `getenv("HOME") = "/home/albert"`

SETTENV("evar=val") . . assigne une variable d'environnement *evar.*

Ex: `stat=setenv("CAT=file.rdb")`

GETLU() . . . . . retourne une unité logique inutilisée.

Ex: `local unit=getlu()`

SLEEP(sec) . . . . . Suspend le process durant un temps donné en seconde

Ex: `stat=sleep(1.5)`

SYSTEM(cmd) . . . . . retourne le résultat fournis par la commande system *cmd.*

Ex: `system("hostname") = "ouranos"`

VERS() . . . . . retourne la date de la dernière compilation d'Inter.

Ex: `vers() = "inter_SunOS_5.8 - Fri Sep 13 07 :38 :08 CLT 2002"`

USER() . . . . . retourne le No d'utilisateur.

Ex: `user() = 4730.0000`

GROUP() . . . . . retourne le numéro de groupe.

Ex: `group() = 4000.0000`

$(string) . . . . . retourne le contenu de la variable nommée dans string. Attention, en phase de compilation on ne sait pas si on fait une indirection sur une variable numérique ou caractère. En mode compilation et pour en tout cas autoriser les doubles indirections, Inter pose le résultat comme type caratère. Donc si l'on veut assigner une variable numérique il faut le faire en 2 temps : local x ; x=$(var) ; x=x+n

Ex: `$("nx") = 100.00000`

RM ?(string)..... Cette fonction supprime les points d'interrogation en fin de ligne. (utile pour les commandes @@ et @@@ ...).

Ex: `rm?("@qq?? 1 2???") = "@qq?? 1 2"`

COMMUNICATION

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.

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

SELECT(server)..... Séléctionne le serveur sur lequel vont travailler toutes les fonctions de communication. Rem : le mot–clé "myself" selectionne son propre bloc de communication

Ex: `i=select("ccd")`

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

SHMGET(key)..... lit un paramètre référencié par *key* dans le bloc de communication.

Ex: `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.

Ex: `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.

SHMPCOD()..... met le code d'erreur, l'erreur et le message d'erreur en shared memory.

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

SHMGACK() . . . . . retourne la valeur du flag ackno.

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

SHMCMD(cmd [,to]) . . . 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

Ex: `i=shmcmd("@qq")`

SHMCMDW(cmd [,to_wait [,to_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

SHOWSEL() . . . . . affiche le nom de tous les serveurs possible ainsi que le serveur actuellement séléctionné.

Ex: `i=showsel()`

SRVEXIS() . . . . . retourne 1 si le serveur existe.

SRVWAIT() . . . . . indique si le client a la main sur le serveur courant. Permet de traiter les erreurs dues aux time-out.

Ex: `if srvwait() i=shmfree()`

SRVWORK()	indique si le serveur est en train de travailler. retourne 0 s'il est sur le prompt (independement du fait qu'il soit ou non réservé. Retourne -1 s'il est occupé par un autre client, retourne +1 s'il est occupé par nous même.
SIGNAL(signal)	Envoie un signal au serveur. retourne le status de la fonction kill ou -1 si le serveur est inexistantex : 2=CTRL–C (voir "man -s5 signal")
SHMNCNT()	retourne le nombre de client en attente sur le serveur courant.
SHMZERO()	met le semaphore zéro à 1 (reset).
SHMZCNT()	retourne le nombre de process en attente de zero sur sem No.
SHMSRV()	retourne 1 si Inter a été lancé en mode serveur.
SHMCLI([name])	retourne 1 si Inter a été lancé en mode client. Si name est précisé, alors retourne 1 si Inter est client de *name*
FETCH()	assigne les variable Inter selon les couples donnés dans le bloc de communication.
SHMGCOD()	retourne le code d'erreur (ascii).
SHMGERR()	retourne le code d'erreur (numérique).
SHMGID()	retourne l'identificateur du bloc de mémoire partagée.
SHMGMES()	retourne le texte du message d'erreur.
SHMGSRV()	retourne le nom du serveur courant.
SHMGSTA()	retourne le status d'erreur.
SEMAGET(No)	retourne la valeur du sémaphore *No.*
SEMASET(No, val)	Met le sémaphore *No* à *val.*

FICHIERS FITS

RFITS(file, key)	Lit un keyword *key* du fichier *file.*

LOGBOOK

INILOG(host)	Initialise une connection sur le logbook de la machine *host.* Retourne 0 en cas de succès ou -1 si la connection est impossible. Dans ce dernier cas, Inter ne voit pas d'erreur. L'initialisation peut se faire de manière externe à Inter en envoyant le signal SIGUP à Inter.

Ex: `unix> kill -1 <pid_inter>`

DEBUG

ADD(variable) . . . . . retourne l'adresse de la *variable* numérique dans les tableaux d'Inter.

DIM(variable) . . . . . retourne la dimension d'une *variable*.

TEMPÉRATURES CCD

TEMP(flag) . . . . . retourne certaines températures du CCD. Cette fonction demande un argument (entre 1 et 4) et retourne une des température mesurée sur le sytème CCD en degré Kelvin. Pour $flag = 1$ on a le réservoir d'azote, pour $flag = 2$ on a ccd1, pour $flag = 3$ on a ccd2 et pour $flag = 4$ on a la paroi extérieure.

Ex: `write "T boitier = " TEMP(1)-27`

COORDONNÉES

CAL(x,y) . . . . . convertion de coordonnées cartésiennes en coordonnées équatorialles. retourne <alpha>. Cette conversion utilise des variables prefixées "ALI" ou "GSC". Elle permet de retrouver le coordonnée équatoriale alpha d'une position pointée sur l'afficheur après avoir utilisé la commande ALIGN.

Ex: `alpha=CAL(x,y)`

CDE(x,y) . . . . . convertion de coordonnées cartésiennes en coordonnées équatorialles. retourne <delta>. Cette conversion utilise des variables prefixées "ALI" ou "GSC". Elle permet de retrouver la coordonnée équatoriale delta d'une position pointée sur l'afficheur après avoir utilisé la commande ALIGN.

Ex: `delta=CDE(x,y)`

Chapitre 2

QUICKREF

Fonctions	Descriptions
*	multiplication.
**	exponentiation.
+	addition.
−	soustraction.
/	division.
.AND.	ET logique.
.EQ.	égalité numérique.
.GE.	plus grand ou égal.
.GT.	strictement plus grand.
.LE.	plus petit ou égal.
.LT.	strictement plus petit.
.NE.	non égalité numérique.
.NOT.	négation de la condition.
.OR.	OU logique.
ABS(x)	retourne la valeur absolue de *x*.
ACOS(x)	retourne l'arc cosinus de *x* en radian.
ACOSD(x)	retourne l'arc cosinus de *x* en degré.
ADD(variable)	retourne l'adresse de la *variable* numérique dans les tableaux d'Inter.
AHDE2AZ(angle_horaire, delta)	calcul l'azimut.
AHDE2EL(angle_horaire, delta)	calcul l'élévation.
AND(a,b)	retourne le résultat du 'and' binaire
ANGLE("angle")	retourne la valeur numérique d'un angle donné sous forme de chaîne de caractères.
ANUT(alpha,delta)	nute la coordonnée donnée en alpha.
APPNAME()	retourne le nom de l'application.
APREC(alpha,delta)	précessionne la coordonnée donnée en alpha.
ASIN(x)	retourne l'arc sinus de *x* en radian.
ASIND(x)	retourne l'arc sinus de *x* en degré.
ASSIGN(str[,prefix])	assignation de variables selon une chaine formatée contenant une suite variable, contenu.
ATAN(x)	retourne l'arc tangente de *x* en radian.
ATAN2(y,x)	retourne l'arc tangente selon *x,y* en radian.
ATAN2D(y,x)	retourne l'arc tangente selon *x,y* en degré.
ATAND(x)	retourne l'arc tangente de *x* en degré.
ATOR(str)	retourne le nombre écrit dans *str*.
AZEL2AH(azimut, élévation)	calcul angle horaire.
AZEL2DE(azimut, élévation)	calcul delta.
CAL(x,y)	convertion de coordonnées cartésiennes en coordonnées équatorialles.
CDE(x,y)	convertion de coordonnées cartésiennes en coordonnées équatorialles.
CHAR(code)	retourne le caractère ASCII donné par *code*.
CLEAN([mat],seuil,size,valrep)	patch une matrice centree de cote=2*size+1 contenant valrep pour chaque poinr plus eleve que seuil
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.
COMPAC(str)	retourne *str* sans espace.
CONNECT()	Se (re)connecte sur les sémaphores du serveur courant s'ils ont été tués.
COS(alpha)	retourne le cosinus de *alpha* donné en radian.
COSD(alpha)	retourne le cosinus de *alpha* donné en degré.
DCDIV(x,y)	Division. Même remarque que pour DCPLUS.
DCMINUS(x,y)	Soustraction. Même remarque que pour DCPLUS.
DCMUL(x,y)	Multiplication. Même remarque que pour DCPLUS.
DCPLUS(x,y)	Addition. Les arguments sont caractères, le résultat est caractère.
DDIV(x,y)	Division. Même remarque que pour DPLUS.
DDTOD(angle)	formatte *angle* en : "SDDDd MMm SSs ".
DDTOD2(angle)	formatte *angle* en : "SDDD:MM:SS ".
DEG2RAD(x)	conversion degrés radians.
DIM(variable)	retourne la dimension d'une *variable*.
DMINUS(x,y)	Soustraction. Même remarque que pour DPLUS.
DMUL(x,y)	Multiplication. Même remarque que pour DPLUS.
DNUT(alpha,delta)	nute la coordonnée donnée en delta
DPLUS(x,y)	Addition. Les arguments sont numériques ou caractères, le résultat est numérique en simple précision.

Fonctions	Descriptions
DPREC(alpha,delta)	précessionne la coordonnée donnée en delta.
ENVDEF(evar)	retourne 1 si la variable d'environnement est definie, 0 sinon.
EXIST(fichier)	retourne 1 si le *fichier* existe.
EXP(x)	retourne l'exponentiel de *x*.
EXPAND(val|vec, mul [,size])	expand chaque position d'un vecteur ou d'une matrice.
FETCH()	assigne les variable Inter selon les couples donnés dans le bloc de communication.
FIND(str1,str2)	retourne 1 si *str2* existe dans *str1*.
FITGAU(vec)	Fit une gaussienne sur les données de *vec*.
FORMAT(val|vec|str, fmt)	retourne *val*|*vec*|*str* formatté le format *fmt*.
FZ(dist_zénitale)	calcul de la masse d'air (voir slalib).
GENF(xvec, polyvec)	fabrique f(xvec) avec polyvec comme coefficient du polynôme.
GENGAU(vec, nb_points)	génère une gaussienne selon les 4 premières valeurs du vecteur *vec*. (1=max, 2=centre, 3=forme, 4=background
GETENV(evar)	retourne le contenu de la variable d'environnement *evar*.
GETLU()	retourne une unité logique inutilisée.
GROUP()	retourne le numéro de groupe.
HDTOH(heure)	formatte *heure* en : "HHh MMm SS.Ss ".
HDTOH2(heure)	formatte *heure* en : "HH:MM:SS.S ".
HTOHD("heure")	retourne la valeur numérique d'une heure donnée sous forme de chaîne de caractères.
ICHAR(str)	retourne le code ASCII du premier caractère de la chaîne *str*.
INDEX(str1,str2)	retourne la position de *str2* dans *str1*.
INILOG(host)	Initialise une connection sur le logbook de la machine *host*.
INMAXV(vec)	retourne l'index du premier plus grand élément de *vec*.
INMINV(vec)	retourne l'index du premier plus petit élément de *vec*.
INT(x)	retourne la valeur entière de *x*.
ISNUM(xxx)	retourne 1 si "xxx" est un nombre (lexicalement parlant).
ISVNUM(xxx)	retourne 1 si "xxx" est une variable numérique.
ITOA(val)	retourne le nombre *val* formatté en entier sans blanc.
JD([j][,[m][,[a]]])	retourne le jour julien formatté (1x,f11.3,1x).
JD0IN([j][,[m][,[a]]])	initialise le jour julien de référence (jd0) pour les calculs de précéssion et de nutation.
LCAT(str1, ..., strN)	retourne la concaténation des tous les arguments (type caratère) .
LCEQ(str1,str2)	retourne 1 si *str1* = *str2*. Cette fonction est insensible aux minuscules et aux majuscules
LEN(str)	retourne le nombre de caractères dans *str*.
LEQ(str1,str2)	retourne 1 si *str1* $=$ *str2*.
LGE(str1,str2)	retourne 1 si *str1* $\geq$ *str2*.
LGT(str1,str2)	retourne 1 si *str1* $>$ *str2*.
LLE(str1,str2)	retourne 1 si *str1* $\leq$ *str2*.
LLT(str1,str2)	retourne 1 si *str1* $<$ *str2*.
LNE(str1,str2)	retourne 1 si *str1* $\neq$ *str2*.
LNO(str)	retourne 1 si *str* vaut "n", "no" ou "non".
LOG(x)	retourne le logarithme naturel de *x*.
LOG10(x)	retourne le logarithme base 10 de *x*.
LOWER(str)	retourne *str* mis en minuscule.
LSCAT(str1,str2,...strN)	retourne la concaténation des tous les arguments (type caratère) en placant un espace entre chaque argument.
LSFIT(No_de_matrice)	résoud m équations à n inconnues.
LTRIM(str)	retourne *str* sans les espaces à gauche du texte utile.
LYES(str)	retourne 1 si *str* vaut "y", "yes", "o" ou "oui" .
MAX(x1,x2,...xn)	retourne le maximum parmis les argument *x1...xn*.
MAXSIZ(No_mat)	retourne la taille actuelle ou maximum (dans le cas d'une matrice en mémoire partagée) de la matrice No_mat.
MAXV(vec)	retourne le plus grand élément de *vec*.
MEDDEV(vec)	retourne le sigma median (nag :g07daf)
MEDIAN(vec)	retourne le median (nag :g07daf)
MIN(x1,x2,...xn)	retourne le minimum parmis les argument *x1...xn*.
MINV(vec)	retourne le plus petit élément de *vec*.
MOD(x,y)	retourne le modulo de *x* par *y*.
NBCOU(No_mat)	retourne le nombre de couche pour la matrice *No_mat*.
NBMAT()	retourne le nombre de matrices accessibles.
NBPIX()	retourne le nombre de pixels alloués par l'ensemble des matrices en mémoire partagée.

Fonctions	Descriptions
NINT(x)	retourne la plus proche valeur entière de *x*.
NOCOU(str|num)	retourne le No de couche d'un No de matrice.
NOMAT(str|num)	retourne le No de matrice d'un No de matrice.
NOT(a)	retourne le résultat du 'not' binaire
NUTIN([j][,[m][,[a]]])	initialise les variables internes pour le calcul de la nutation.
OR(a,b)	retourne le résultat du 'or' binaire
PAD(str)	retourne *str* sans les espaces en début et fin de chaîne et avec une seule barre de soulignement ("_") pour chaque suites d'espaces entres les mots.
PARSE(str,No)	retourne le champ *No* de *str*, avec pour séparateur le premier caractère de *str*.
POLYF(xvec, yvec, degre [,pvec])	fit polynômial à une inconnue de degré *degré*.
PRECIN([j][,[m][,[a]]])	initialise les variables internes pour le calcul de la précession.
RAD2DEG(x)	conversion radians degrés.
RAND(flag)	retourne un nombre aléatoire.
REFCO(tdk, pmb, rh, [wl], [eps], [tlr], [hm], [phi])	calcul des parametres AREFRA et AREFRA du modèle de réfraction atmospherique.
REFRAC(dist_zénitale)	calcul de la réfraction atmosphérique pour La Silla.
RFITS(file, key)	Lit un keyword *key* du fichier *file*.
RM ?(string)	Cette fonction supprime les points d'interrogation en fin de ligne. (utile pour les commandes @@ et @@@ ...).
ROTHPOS(ah,delta)	angle du derotateur. Remarque : utilise longit et latitu.
ROTHVEL(ah,delta)	vitesse du derotateur [deg/s]. Remarque : utilise longit et latitu.
ROTPOS(alpha,delta)	angle du derotateur moment courant. Remarque : utilise longit et latitu.
ROTVEL(alpha,delta)	vitesse du derotateur moment courant [deg/s]. Remarque : utilise longit et latitu.
RTRIM(str)	retourne *str* sans les espaces à droite du texte utile.
SELECT(server)	Séléctionne le serveur sur lequel vont travailler toutes les fonctions de communication.
SEMAGET(No)	retourne la valeur du sémaphore *No*.
SEMASET(No, val)	Met le sémaphore *No* à *val*.
SETTENV("evar=val")	assigne une variable d'environnement *evar*.
SETV(start,stop[,step])	remplis un vecteur selon les paramètres de boucle.
SHMACK()	indique au serveur d'exécuter la commande placée dans "COMMAND" et lui signale de ne pas rendre la main.
SHMADD(key, content)	ajoute un paramètre référencié par *key* et son contenu *content*, dans le bloc de communication .
SHMCLI([name])	retourne 1 si Inter a été lancé en mode client. Si name est précisé, alors retourne 1 si Inter est client de *name*
SHMCMD(cmd [,to])	Suspend le client, envoie une commande et libère le client.
SHMCMDW(cmd [,to_wait [,to_cmd]])	Suspend le client, envoie une commande et attend la fin de la commande pour libérer le client.
SHMCONT()	indique au serveur d'exécuter la commande placée dans "COMMAND" et lui signale de rendre la main.
SHMFREE()	Rend la main (après un SHMWACK()).
SHMGACK()	retourne la valeur du flag ackno.
SHMGCOD()	retourne le code d'erreur (ascii).
SHMGERR()	retourne le code d'erreur (numérique).
SHMGET(key)	lit un paramètre référencié par *key* dans le bloc de communication.
SHMGID()	retourne l'identificateur du bloc de mémoire partagée.
SHMGMES()	retourne le texte du message d'erreur.
SHMGSRV()	retourne le nom du serveur courant.
SHMGSTA()	retourne le status d'erreur.
SHMINIT()	initialise le bloc de communication en le vidant.
SHMNCNT()	retourne le nombre de client en attente sur le serveur courant.
SHMPCOD()	met le code d'erreur, l'erreur et le message d'erreur en shared memory.
SHMPUT(key, content)	écrit un paramètre référencié par *key* et son contenu *content* dans le bloc de communication et initialise le bloc de communication.
SHMSHOW()	visualise le bloc de communication à l'écran.
SHMSRV()	retourne 1 si Inter a été lancé en mode serveur.
SHMWACK([timeout])	attend que le serveur ait finis (après un SHMACK()).
SHMWAIT([timeout])	suspend le client jusqu'à que le serveur soit prêt.
SHMZCNT()	retourne le nombre de process en attente de zero sur sem No.
SHMZERO()	met le semaphore zéro à 1 (reset).
SHOWSEL()	affiche le nom de tous les serveurs possible ainsi que le serveur actuellement séléctionné.

Fonctions	Descriptions
SIGNAL(signal)	Envoie un signal au serveur.
SIN(alpha)	retourne le sinus de *alpha* donné en radian.
SIND(alpha)	retourne le sinus de *alpha* donné en degré.
SLEEP(sec)	Suspend le process durant un temps donné en seconde
SPARSE(str,No)	retourne le champ *No* de *str*, avec pour séparateur l'espace.
SQRT(x)	retourne la racine carrée de *x*.
SRVEXIS()	retourne 1 si le serveur existe.
SRVWAIT()	indique si le client a la main sur le serveur courant.
SRVWORK()	indique si le serveur est en train de travailler.
STDDEV(vec)	retourne le sigma (nag :g07daf)
STRIP(str)	supprime les blancs, les 0 non significatifs et le point non significatif sur une chaine numérique.
SUM(vec)	retourne la somme des valeur contenus dans *vec*.
SYSTEM(cmd)	retourne le résultat fournis par la commande system *cmd*.
TAN(alpha)	retourne la tangente de *alpha* donné en radian.
TAND(alpha)	retourne la tangente de *alpha* donné en degré.
TEMP(flag)	retourne certaines températures du CCD.
TIME()	donne l'heure de la machine (heures décimales).
TPARSE(str,No)	retourne le champ *No* de *str*, avec pour séparateur le tabulateur.
TRIM(str)	retourne *str* sans les espaces avant et apres le texte utile.
TS([tcl][,[j][,[m][,[a]]]])	donne l'heure sidérale.
UPPER(str)	retourne *str* mis en majuscule.
USER()	retourne le No d'utilisateur.
VEC(arg1,arg2,...argn)	fabrique un vecteur en joignant tous les arguments.
VERS()	retourne la date de la dernière compilation d'Inter.
XBARY([mat])	retourne le x barycentre
XOR(a,b)	retourne le résultat du 'xor' binaire
YBARY()	retourne le y barycentre (il faut utilise xbary(mat) avant)
$(string)	retourne le contenu de la variable nommée dans string.