Data processing softwares and formats *(partly written in french)*

- Connection between the functions of SIR and those of CLASS (stage P. Hoegstoel 96, in french).
- Documentation about FITS format (in french, includes US links).
  CFITS cookbook: How to read FITS file with the CFITS program (from the Grenoble package).
- The CLASS software is available on the nanrt5 Nanï¿½ay computer:
  Since the CFITS program of this version does not work, please try the old Meudon `mesioq` computer version with the following setup:
  alias -x class="/recqdata/martin/osf1/bin/class"
  alias -x cfits="/recqdata/martin/osf1/bin/cfits" (etc...)
  export GAG_ROOT=/recqdata/martin
  export GAG_SYS=osf1
- stokes parameters CLASS procedure to compute the Stokes parameters, linear polarisation parameter and PA : under the CLASS prompt, call the procedure stokes.class (@stokes, version FEB 2008).
- CLASS abbreviations : under the CLASS prompt, call the procedure abb.class (@abb, version April 2002).
- NCDRT : Nancay Continuum Data Reduction Tool, ***M. Roos Serote's*** software developed for reducing the continuum data obtained with the NRT.

October 19th, 2007. P. Colom, J.-M. Martin
Last modification: 26 March 2008. P. Colom

AOUT 96.

Help de comparaison des instructions SIR et CLASS

Systeme Interactif de Reduction

Continuum and Line Analysis Single-dish Software

Preliminaires :

Ce document reprend l'ordre alphabetique de l'aide
de SIR et pour chaque commande donne un equivalent
dans CLASS.

Il est bases sur les version suivantes :

SIC	V 8.0	du 15-DEC-94
LAS	V 4.1-0	du 08-APR-88
SIR	V 05/95	du 29-MAY-95

La plupart des commandes arithmetiques, des fonctions de
langages, comme les boucles, les tests logiques... sont
en fait des commandes SIC.

Les fonctions de plus haut niveau (typiquement les manipulations
de spectres ou de headers) sont des commandes CLASS.

Ces deux modules etant intimement lies (avec d'autres en plus),
il n'est pas a la charge de l'utilisateur de preciser si il
s'agit d'une commande CLASS ou SIC ou autre, sauf en cas
d'ambiguite, ce qui est de suite signale par l'interpreteur
de commandes.

Dans ce qui va suivre, il peut arriver qu'un paragraphe (la
description d'une commande) fasse allusion a une autre commande.
Dans ce cas il est ecrit COM1 => COM2 ou COM1 represente la
commande SIR et COM2 la commande equivalente CLASS.

Tous les paragraphes sont construits de la maniere suivante :

COMMANDE SIR

  COMMANDE CLASS : commentaires
                    (eventuellement un exemple)

la commande Class est remplace par "inexistant" si il n'y a pas
d'equivalent au sens strict du terme.

Les instructions LET sont facultatives et sont precisees pour bien
montrer qu'il s'agit d'une affectation.

ABSO

 ABS : Retourne la valeur absolue d'un argument.
        ex :   DEFINE INTEGER A
                LET A = ABS(-10)

        On ne trouvera pas d'equivalent a ABSO* ou tout autre
        manipulation sur les tableaux 1..8 specifiques a SIR.
        Cela restera vrai pour toutes les commandes les manipulant.

ADDI

 "+" : Cette fonction est remplace par l'operateur mathematique
        "+" et donc est plus naturel d'emploi.
        ex :   DEFINE INTEGER A B
                LET A = 10
                LET B = 20
                LET A = (A+B)

        SIR autorise la concatenation de chaines avec ADDI.
        Sous CLASS, la concatenation s'effectue de la maniere
        suivante :

```
ex :    DEFINE CHARACTER CHAINE1*5       !*5, *6, *11
        DEFINE CHARACTER CHAINE2*6       !representent les
        dEFINE CHARACTER CHAINE3*11      !longueurs des chaines
        LET CHAINE3 = 'CHAINE1''CHAINE2'
```

On prendra soin de declarer une variable cible suffisamment grande en taille.

AREA

AREA :

AREA est une variable globale du header dont la valeur peut etre editee a l'aide de EXAMINE ou SAY. Cette variable est modifiee en fonction des operations qui sont effectuees sur les donnees. Par exemple, a l'issue d'une soustraction de ligne de base, AREA va prendre la valeur de l'aire non masquee sous la ligne de base.
La maniere "classique" de faire un calcul d'aire est de charger un spectre (ou une moyenne), de definir un masque, de soustraire une ligne de base, definir le nombre de gaussiennes a fitter, et rechercher ces gaussiennes.

```
ex :    FILE IN toto.nan
        FIND/ALL
        SUM
        DEVICE XAUTO WHITE
        PLOT
        SET WINDOW              ! voir CLIP => SET MASK
                                !           => SET WINDOW
        BASE n/PLOT             ! n = degre du polynome
        LINES 2                 ! pour 2 gaussiennes
        GAUSS                   ! recherche interactive
                                ! a l'aide du curseur.
                                ! le click souris marche !
        FIT                     ! affiche le resultat
```

Les parametres tels que l'aire sont alors affiches a l'ecran.

ASKN

LET : La saisie d'une valeur numerique ou alphanumerique se programme par l'instruction LET sans utiliser de valeur derriere.

```
ex :    LET x = 10 initialise a 10 la variable x.
        LET x      invite l'utilisateur a entrer
                   la valeur de x et fourni l'affichage
                   suivant a l'ecran : x =
```

Un retour chariot sans entrer une valeur ou la saisie d'une valeur incoherente vis a vis du type de variable provoque une erreur et la pause caracteristique de tout probleme dans l'execution d'une procedure CLASS.

L'option SYNTAX | FIXED | FREE | permet de ne pas preciser (dans le cas de FREE) la commande LET. Ainsi l'instruction x = 10 sera valide. FREE est l'option par defaut.

Dans les versions recentes de SIC, vous pourrez trouver l'option /RANGE ou /CHOICE permettant de controler immediatement la validite de la valeur saisie.

BASE

BASE : La soustraction d'une ligne de base fonctionne a peu pres de la meme maniere dans les deux systemes.
CLASS charge un spectre dans un tableau R. Lors d'un appel a BASE, ce tableau est copie dans un tableau memoire T. La soustraction de la ligne de base est alors appliquee sur le tableau R.

```
ex :    FILE IN toto.nan
        FIND
        GET FIRST
        SET WINDOW      ! definition interactive d'un masque
        BASE 4/PLOT     ! degre 4 et imprimee sur le graphique
```

le degre du polynome a soustraire peut aussi etre defini par SET BASE n. Par defaut n = 1.

L'utilisation de masques (comme SETM de SIR) fait intervenir dans CLASS la notion de fenetre (WINDOW).
Voir SETM => SET WINDOW

BRAL

inexistant :
ne pas chercher l'equivalent d'un goto ou tout autre branchement inconditionnel comme BRAL dans CLASS puisque cela n'existe pas ! Dans un souci de lisibilite, de maintenance et de fiabilite de programmation, les structures de programme qui ont ete inclue dans SIC (et donc dans CLASS) sont les IF..THEN..ELSE..ENDIF, FOR..NEXT et FOR/WHILE..NEXT que l'on retrouve dans tous les langages de programmation classique.
Cela necessite de penser ses procedures differemment.
Le BRAL n'est absolument pas necessaire sir l'on agence bien les differentes etapes de son programme. Les boucles WHILE, associees a des tests IF, permet de venir a bout de n'importe quel algorithme du 20eme siecle. (Mis a part les besoins incontournables de la recursivite).

ex : Imaginons la situation SIR suivante :

```
1: ASKN "Entrer 0 ou 1 :", A
   BREQ 2, A
   COMP 1, A
   BREQ 3
   BRAL 1
2: MESS Vous avez entre 0
3: MESS Vous avez entre 1
```

Cet exemple va boucler tant que l'utilisateur n'aura pas entre effectivement 0 ou 1 et sauter au paragraphe 2 ou 3 selon la valeur entree.

L'equivalent en CLASS s'ecrira :

```
DEFINE INTEGER VALEUR
LET VALEUR
FOR/WHILE VALEUR.NE.0.AND.VALEUR.NE.1
        LET VALEUR
NEXT
IF VALEUR.EQ.0 THEN
        SAY "Vous avez entre 0"
ELSE
        SAY "Vous avez entre 1"
END IF
```

Certes, vous aurez tape 10 lignes de code au lieu de 7, mais la lisibilite sera nettement meilleure, et la comprehension ulterieure du programme s'en resentira. Seuls le temps et la pratique permettent de se convaincre de l'interet de ce genre de syntaxe.

Pour tester la validite d'une reponse, voir aussi la derniere remarque de ASKN => LET.

BREA

PAUSE :Il n'existe malheureusement pas d'equivalent a la commande BREAK de SIR. Celle qui s'en rapproche le plus est l'instruction PAUSE qui met en suspens l'execution de la procedure jusqu'a ce que l'utilisateur entre (pour continue) puis ou bien " puis pour quitter.

ex : PAUSE "Ceci est une pause. pour continuer"

BREQ

inexistant :
Comme pour BRAL, on ne trouvera pas d'equivalent dans SIC. Il faut structurer son programme de maniere a n'utiliser que des boucles FOR..NEXT, FOR/WHILE..NEXT et des tests IF..ENDIF
Ne pas chercher a faire IF condition THEN GOTO puisque le goto n'existe pas. Ce qui aurait du figurer dans le paragraphe du goto devra tout simplement etre mis a la suite du IF.
ELSE et/ou l'imbrication de plusieurs IF permet d'avoir ainsi plusieurs cas possibles.
Les tests logiques possibles sont detaille ci-dessous.

BRGE

inexistant :
Meme remarque que pour BRAL.
en ce qui concerne les tests logiques destinees a effectuer telle ou telle operation en fonction du resultat, SIC offre les fonctions suivantes :

.OR.	ou logique
.AND.	et logique

```
.NOT.           non logique
.GT.            >
.LT.            <
.GE.            >=
.LE.            <=
.NE.            <>
.EQ.            =
```

Les . sont a respecter dans la syntaxe.
voir exemple de BRAL=>inexistant

BRGT

inexistant : idem ci-dessus

BRLE

inexistant : idem ci-dessus

BRLT

inexistant : idem ci-dessus

BRNE

inexistant : idem ci-dessus

CALL

@ : l'appel d'une procedure a l'interieur d'une autre procedure est possible dans CLASS, tant que l'on ne fait pas d'appel recursif, c'est a dire appeler une procedure a l'interieur d'elle meme.
ex : Si une procedure A doit appeler une procedure B, on tapera au prompt /LAS de CLASS :

```
@ A pour appeler la procedure principale.
( on peut se creer des racourcis ou alias par
la commande SYMBOL)
le corps de la procedure A fera de la meme
maniere a B : @ B
```

Un des avantages majeur de SIC, est qu-il permet de passer en argument des parametres aux fonctions appelees. Pour reprendre l'exemple du paragraphe BRAL => inexistant, imaginons que nous voulions remplacer le IF..THEN..ELSE par une procedure independante. Nous ecrivons alors :

```
DEFINE INTEGER VALEUR
LET VALEUR
FOR/WHILE VALEUR.NE.0.AND.VALEUR.NE.1
        LET VALEUR
NEXT
@TEST VALEUR
```

La procedure TEST s'ecrira donc :

```
! procedure TEST.CLASS

DEFINE INTEGER VALEUR_RECU
LET VALEUR_RECU = &1
IF VALEUR_RECU.EQ.0 THEN
        SAY "Vous avez entre 0"
ELSE
        SAY "Vous avez entre 1"
END IF
RETURN
```

Dans la premiere procedure, nous appelons @ TEST en lui donnant comme argument la variable VALEUR.
La procedure destination reconnait les arguments qui lui sont donnes grace a leur position. Il faut donc, dans le cas d'arguments multiples, qu'il y ai adequation entre la position des arguments donnes et celle des arguments attendus. Le nombre maximum d'argument est 8. Leurs noms , dans la procedures destination sont &1, &2, &3...
Je conseillerai vivement de recopier de suite la valeur dans une variable plutot que de travailler directement sur &x, ce qui d'ailleur n'est pas toujours possible.

Les procedures ne sont pas des fonctions et le passage d'arguments est unilateral c'est a dire q'une procedure ne peut pas donner de parametre a sa procedure appelante.

On peut contourner ce probleme en utilisant des variables globales :
Si l'on declare dans une procedure :

DEFINE REAL TRUC/GLOBAL, cette variable sera connu de toutes les procedures appelees, quelque soit leur niveau d'imbrication. Ainsi si une procedure B appelee par @B dans une procedure A fait LET TRUC = TRUC+1, ce resultat sera connue de la procedure appelante A puisque la variable est globale.

La difference avec une variable dite locale dans une procedure, c'est que cette derniere est detruite (enlevee de la memoire) lorsque la procedure se finit. Exactement comme quand on effectue un DELETE/VARIABLE nom_variable.

CHPY

inexistant :
voir HARD => HARDCOPY

CIDX

FIND : Lorsque l'on cherche des spectres a partir d'un certain nombre de criteres (voir CRIT => SET), on cree le resultat de la requete a partir de l'instruction FIND. Un tableau appele INDEX contient tous les numeros de spectres issus de la recherche. INDEX est un tableau comme tous les autres qui peut se manipuler element par element : INDEX[1], INDEX[10], INDEX[i].
Le nombre d'elements trouves, ce qui correspond aussi a la taille du tableau INDEX est donne par la variable globale FOUND.

FIND/ALL travaille de la meme maniere que FIND en ommettant le critere de selection sur les numero de version.
Si aucun critere de version n'a ete defini, FIND et FIND/ALL retourneront le meme resultat d'INDEX.

LIST IN affiche a l'ecran la liste des spectres contenus dans le fichier FILE IN sans tenir compte des criteres de recherche.

voir aussi CRIT => SET
MCRI => SET

CLEA

inexistant :
Cependant, on peut soit utiliser une boucle FOR pour reinitialiser le tableau. En fait la boucle est implicite puisqu'il suffit d'ecrire :

```
DEFINE INTEGER TAB[10]  ! tableau de 10 entiers

LET TAB[i] = 0          ! cela marche aussi pour plusieurs
                        ! dimensions
```

ON peut aussi effacer la variable et la recreer :

```
DELETE/VARIABLE TAB
DEFINE INTEGER TAB[10]
```

mais la 1ere solution est bien meilleur car il n'y a qu'une seule declaration.

CLIP

SET MASK :
SET WINDOW :
toutes les operations de masquage de points passent par SET WINDOW ou SET MASK.

SET WINDOW est utiliser pour masquer des points dans le cas d'une recherche de ligne de base.
On peut definir plusieurs fenetres :

```
SET WINDOW pt11 pt12 pt21 pt22 ...
```

Les points definis dans la fenetre sont masques.
Ce parametrage peut s'effectuer interactivement et graphiquement par le biais d'un curseur. (SET CURSOR ON).
Dans ce cas des touches sont predefinies car les boutons de la souris ne sont pas geres.

```
                        N       new boudary
                        C       cancel last entry
                        H       help
                        E       exit
        SHOW WINDOW permet d'afficher les fenetres definies.
        DRAW WINDOW les affiches en surimpression du spectre.


        SET MASK cache des points pour la recherche d'une gaussienne.
        Memes remarques pour    SET CURSOR ON
                                les touches predefinies
                                SHOW MASK
                                DRAW MASK


CLRM

 SET MASK :
 SET WINDOW :
        Pour effacer un masque precedemment defini, il faut
        executer la commande SET WINDOW (ou SET MASK) sans
        parametre et taper immediatement E pour sortir de la
        saisie interactive.
        Utiliser SET WINDOW 0 0 reduit le masque a un intervalle
        ridicule mais ne le detruit pas vraiment.

        ATTENTION : CLEAR WINDOW existe mais n'a rien a voir avec
        la notion de masque et efface la fenetre graphique de l'ecran
        ainsi que tout ce qui s'y rapporte.

COMP

 inexistant :
        Il n'y a pas d'operateur de comparaison au sens de COMP.
        Cependant, on trouve tous les operateurs logiques necessaires.
        Ceux-ci sont donnes au paragraphe BRGE => inexistant.
        Associes a la fonction IF, ils rendent le meme service.

CORR

 inexistant

COSR

 COS  :
 COSH : cette fonction retourne le cosinus (ou cosinus hyperbolique)
        de la valeur qui lui est donne en argument.

        ex :    DEFINE REAL A B
                LET A = 3.14
                LET B = COS(A)   ! retournera -1

        Les arguments doivent etres donnees en radians.
        La constante PI est definie en globale pour d'eventuelles
        conversions.

CRIT

 SET :  Pour definir un ou plusieurs criteres de recherche, on
        impose a l'aide de la fonction SET des valeurs a un certain
        nombre de rubriques du header des spectres.

        ex :    si l'on recherche tous les spectres dont le
                numero de version est 2 on pourra taper :
                SET NUMBER *2

                * faisant office de "joker" et remplacant
                n'importe quel caractere.
                La variable VERSION existe et peut etre interrogee
                grace a EXAMINE mais ne peut pas faire l'objet
                d'un SET particulier.
                On peut effectuer un SET sur la majorite des rubriques
                du header.

        SET DEFAULT permet d'annuler tous les criteres definis.

        Une instruction supplementaire, IGNORE, permet d'isoler
        des spectres qui ne seront jamais selectionnes quelque
        soient les criteres definis par SET.

                voir aussi CIDX => FIND
                           MCRI => SET


CURS
```

```
 SET CURSOR ON :
        rend visible le curseur en vue de son utilisation sur
        la fenetre graphique.
        voir aussi SETM => SET WINDOW
                        => SET MASK

DBUG

 inexistant :
        cette option de debugage peut etre remplace par l'affichage
        de message et de contenus de variables judicieusement
        places.

DEFI

 inexistant :
        Les 8 tableaux predefinis de SIR n'existent pas dans CLASS,
        cependant, on peut declarer autant de tableaux que l'on veut
        par DEFINE type nom_var[dim]

        ou type est le type des elements contenus dans le tableau
           nom_var le nom du tableau
           dim sa dimension (unique ou multiple)

        dans ce cas, l'intervalle d'utilisation de ce tableau
        est sa dimension.

        voir aussi LOAD => GET


DIVI

 "/" :
 "|" :  La fonction de division est remplacee par l'operateur
        mathematique qui y est habituellement associe.
        l'operateur "/" etant egalement utilise pour certaines
        options comme LET.../WHERE, SIC offre la possibilite
        d'effectuer une division a l'aide du symbol "|"

        ex :    DEFINE INTEGER A
                DEFINE INTEGER B
                DEFINE REAL    C

                LET C = (A|B)

EINT

 INT :  INT retourne la partie entiere de l'argument qui lui est
        donne.

        ex :    DEFINE REAL    A
                DEFINE INTEGER B
                LET A = 4.567
                LET B = INT(A)

                B prend la valeur 4

        A noter qu'il existe dans CLASS une fonction qui arrondie
        une valeur a l'entier le plus proche : NINT.

                si l'on ecrit :
                LET B = NINT(A)

                B prendra la valeur 5


ENDL

 NEXT : la fin d'une boucle, dans CLASS, est marquee par l'instruction
        NEXT, qu'il s'agisse d'une boucle FOR...NEXT ou d'une boucle
        FOR/WHILE...NEXT.

        rem : les variables compteur des boucles FOR (I) sont declares
              implicitement.

        ex :    DEFINE INTEGER TAB[10]
                FOR I 1 TO 10
                        EXAMINE TAB[I]
                NEXT

        ex :    DEFINE INTEGER TAB[10]
                DEFINE INTEGER COMPTEUR
                LET COMPTEUR = 10
                FOR/WHILE (COMPTEUR.GT.0)
                        EXAMINE TAB[I]
                        LET COMPTEUR = COMPTEUR-1
```

```
                NEXT
```

ENTE

LET : Il n'y a pas d'equivalent a cette fonction dans le sens ou le vecteur @ de SIR n'existe pas. Cependant, une fois que l'on a declare un tableau (DEFINE) on peut initialiser chacun de ses elements a l'aide de la fonction LET.

```
ex :    DEFINE INTEGER TAB[10]
        LET TAB[3] = 5
        LET TAB[7] = 4
```

si l'on veut que la valeur soit definie a l'execution, il suffit de ne pas mettre de valeur.

```
        LET TAB[3]
        LET TAB[7]
```

EVAL

inexistant :
l'evaluation d'une expression arithmetique composee ne necessite pas, dans CLASS, d'instruction particuliere.

```
ex :    DEFINE REAL A
        LET A = MIN((EXP(COS(3.14))),(EXP(SIN(3.14))))
```

SIC propose a peu pres toutes les fonctions classiques :

fonctions a 1 argument :
ABS, ACOS, ASIN, ATAN, COS, COSH, EXP, INT, LOG, LOG10, NINT, SIN, SINH, SQRT, TAN, TANH

fonctions a 2 arguments :
ATAN2, MAX, MIN, MOD, SIGN

operateurs :
+, -, *, / ou |, ** ou ^

Les operations sur dates gregoriennes ou juliennes ne sont pas predefinies dans CLASS mais peuvent reprogrammees comme une procedure ou comme une fonction. En effet, l'utilisateur peut definir ses propre fonctions a l'aide de DEFINE FUNCTION ainsi l'exemple ci-dessus pourrai devenir :

```
DEFINE REAL A
DEFINE REAL ARG1 ARG2

DEFINE FUNCTION TRUC(P1, P2 ) MIN((EXP(COS(P1))),(EXP(SIN(P2))))

LET ARG1 = 3.14
LET ARG2 = 3.14
LET A    = TRUC(ARG1, ARG2)
```

EXPN

EXP : Cette fonction retourne l'exponentiel de son argument.

```
ex :    DEFINE INTEGER A
        LET A = 1
        SAY 'EXP(A)'    affichera 2.718
```

EXTR

inexistant :
Cette fonction qui dans SIR fournie les indices et valeurs du minimum et/ou du maximum d'un tableau est facilement reprogrammable a l'aide d'une boucle parcourant un tableau.

FFTA

FFT : l'instruction FFT lance le calcul de la transformee de Fourier sur le tableau R et trace le resultat dans la fenetre graphique. Le contenu du tableau R n'est pas affecte par ce calcul.
On peut definir des zones a effacer en utilisant le curseur comme pour definir une fenetre. E (exit) permet de sortir de la fenetre graphique.
FFT/REMOVE.

FOLD

FOLD : La commande FOLD

GAUS

GAUSS :fit d'une gausienne sur un spectre. Il faut avoir prealablement
defini un masque et soustrait une ligne de base.
l'instruction LINES permet de definir certains parametres comme
le nombre de gaussiennes a fitter.

Une fois ces elements definis, il suffit de taper GAUSS puis FIT
pour visualiser dans la fenetre graphique.

METHOD GAUSS
SHELL
NH3(n,n)
HFS
CONTINUUM

permet de fixer une methode de fit.

voir BASE => BASE
voir CLIP => SET MASK
=> SET WINDOW

GETF

inexistant :
En ce qui concerne les exportations et importations de fichiers,
il faut s'orienter vers la commande GREG. +fits (CFITS)

GETM

SHOW MASK :
SHOW WINDOW :
ces instructions permettent d'afficher les masques et fenetres
definies par SET MASK et SET WINDOW.

HARD

HARDCOPY :
Fournie un fichier postscript de l'ecran graphique.
cette fonction propose differentes options concernant
l'orientation de la feuille, le peripherique de sortie...

ex : HARDCOPY nom_fic /DEVICE PS FAST ! (ou GREY ou COLOR)

L'option /PLOT permet de faire une sortie immediate.

HEAD

HEADER :
MODIFY :
TAG : HEADER affiche le contenu du header du spectre courant.
MODIFY et TAG permettent d'intervenir sur le contenu.
TAG est juste destine a renseigner sur la qualite de
l'observation. C'est une valeur entiere de 0 a 9.
MODIFY permet de changer les principaux parametres du
header :
frequence, offset, canal de ref, nom de raie,
vitesse au repos, resolution...

La syntaxe est a peu pres toujours la meme :

MODIFY nom_rubrique valeur

HELP

HELP : Aide "en ligne" de CLASS comprenant principalement la
description des instructions SIC, LAS, ANALYSE et GTVL.
Le manuel de reference reste cependant indispensable
pour un bon nombre de details.

HIST

inexistant :
Il n'y a pas la possibilite, sauf creation d'une
procedure particuliere, de construire l'histogramme
d'un tableau.

SET PLOT H permet de tracer un spectre sous forme
d'histogramme. Cependant, il ne s'agit pas de
l'histogramme du spectre qui doit fournir une
quantification du nombre de canaux pour une temperature
donnee. (comme c'est le cas dans SIR)

INDV

inexistant :
Pour obtenir la vitesse correspondant a un indice donne
, il faut interroger le tableau RX.

ex : SAY 'RX[30]' affichera la vitesse du point
d'indice 30.

Pour la fonction reciproque il faut programmer une
fonction qui recherche la vitesse souhaitee et qui
retourne l'indice correspondant.

INTE

ACCUMULATE :
SUM : ACCUMULATE permet d'integrer les 2 spectres des tableaux
R et T, le resultat etant conserve dans R. Cette integration
est ponderee en tenant compte du poids de chacun des spectres.

Si les spectres ne coincident pas dans leur position et
dans leur calibration (SET MATCH et SET CALIBRATION), un
message d'alerte est affiche.

SUM effectue l'integration de l'ensemble des spectres
contenus dans le tableau INDEX (voir CRIT => SET ou
CIDX => FIND). Il y a egalement verification de la
coherence entre les calibrations et les positions.

INTV

ACCUMULATE :
SUM : Voir ci-dessus.

INVM

inexistant : Il n'y a pas de methode prevue pour prendre le
complement des masques ou fenetres definies.

IPOL

inexistant

LCRI

SHOW : La fonction SHOW est assez generale pour l'affichage
d'information. Associee a un champ particulier du HEADER,
elle renseigne sur son contenu. L'option ALL (SHOW ALL)
fourni la liste des parametres definis par l'instruction
SET. Une * doit signifier qu'il n'y a pas de contraintes
particulieres imposes sur un champs du header pour la
recherche par FIND.

LIST

TYPE : La commande TYPE utilisee seule affiche le contenu de la
pile des commandes entrees interactivement.

ex : Si depuis le debut de la session, l'utilisateur

a tape :
FILE IN toto.nan
FIND/ALL
DEVICE XAUTO WHITE
@ proc1.class
CLEAR WIN

TYPE fournira la liste de toutes ces instructions,
sauf celles qui seront issues de l'execution de
proc1.class.

TYPE nom_proc affiche le code de la procedure nom_proc a
l'ecran. L'extension du fichier peut etre omis si
celui ci est .class. Si ce n'est pas le cas, il faut
la preciser.

```
ex :    TYPE ma_proc            ! si le fichier est ma_proc.class
        TYPE mine_too.txt       ! si pas .class
```

ATTENTION : l'instruction LIST existe dans CLASS mais sert
a afficher la liste des spectres de l'INDEX issu de la
recherche sur criteres par FIND.

LOAD

GET : Il n'y a pas d'equivalent des 8 tableaux predefinis de SIR.

La structure de donnees qui s'en rapproche le est le tableau
defini automatiquement lors du chargement d'un spectre.
Lors de cette operation, un tableau RY conserve les valeurs
des ordonnees, RX celles des abscisses et toutes les donnees
du header sont egalement stockees en memoire. On peut memoriser
le spectre et son header dans une structure de copie par :

ex : FILE IN TOTO.NAN ! fichier de spectres

```
FIND/ALL
GET FIRST
MEMORIZE une_memoire
GET NEXT
MEMORIZE deux_memoires
GET NEXT

! a ce moment, c'est le 3eme spectre de TOTO.NAN qui
! est en memoire, mais on peut rappeler les deux
! premiers en faisant :

RETRIEVE une_memoire

! le 3eme est alors place dans une memoire tampon
! TX, TY et "T"header. on peut le remettre dans la
! memoire R par SWAP

RETRIEVE deux_memoires

! dans ce cas, le 2eme spectre est mis dans R
! le 1er qui etait dans R passe dans T
! et le 3eme aui etait dans T disparait puisque
! on ne l'a pas memorise.
```

LOGN

LOG10 :
LOG : Retourne le logarithme (ou le logarithme decimal) de
l'argument qui lui est donne.

```
ex :    SAY 'LOG(10)'   ! affiche 2.3
        SAY 'LOG10(10)' ! affiche 1
```

voir aussi EVAL => inexistant

LOOK

STAMP :La visualisation le plusieurs spectres sur la fenetre
graphique peut etre obtenu par STAMP.
cette fonction exige 2 arguments qui representent le
nombre de spectres horizontalement et verticalement.
On peut preciser l'option /NUMBER pour faire apparaitre
dans le coin superieur gauche de chaque fenetre le numero
de spectre concerne.

```
ex :    FILE IN toto.nan
        FIND/ALL
        DEVICE XAUTO WHITE
        STAMP 4 5 /NUMBER
```

affichera 20 spectres avec leur numero respectif, sous
forme d'un tableau de 4 colonnes de 5 spectres.

On peut afficher plusieurs spectres a des emplacements
determines par l'utilisateur a l'aide de SET BOX_LOCATION.
Cette instruction fonctionne de la maniere suivante :

```
ex :    SET BOX_LOCATION 3 10 3 10
        BOX
        SPECTRUM
```

defini une boite dont le coin inferieur gauche est a 3
centimetres des bords de la feuille et le coin superieur
droit a 10 centimetres. Le point de coordonnee 0,0 etant
le coin inferieur gauche de la feuille d'affichage. Il
est sous entendu que la fenetre graphique represente une

feuille de format 21*29.7 d'ou la definition en cm des
arguments donnees a la fonction.

format general : SET BOX_LOCATION X1 X2 Y1 Y2 avec
X1 Y1 les coordonnees du coin inf. gauche et
X2 Y2 du coin sup. droit de la boite a
afficher.

BOX affiche cette boite.
SPECTRUM insere dans cette boite le spectre courant.

TITLE n'est pas encore au point (V 4.1 88). Il s'affiche
toujour en haut de la fenetre graphique et non pas au dessus
de la boite.

Pour effacer la derniere operation graphique, utiliser
CLEAR SEGMENT puis ZOOM REFRESH.

LOOP

FOR : Les boucles LOOP de SIR ont ete remplacees par des
structures plus evolues comme FOR...NEXT et FOR/WHILE...NEXT.

La boucle FOR...NEXT est une boucle a compteur d'iteration.

```
ex :    FOR I 1 TO 50
                .....
                suite d'instruction...
                .....
        NEXT
```

Cette boucle va tourner jusqu'a ce que I prenne la valeur 51.
Les variables compteur de ce genre de boucle n'ont pas besoin
d'etre declarees. Elles le sont implicitement par l'interpreteur.

La boucle FOR/WHILE...NEXT est une boucle a interruption
conditionnelle suivant un test logique.

```
ex :    DEFINE CHARACTER REPONSE*3

        LET REPONSE
        FOR/WHILE (REPONSE.EQ."O".OR.REPONSE.EQ."OUI")
                ....
                suite d'instruction...
                ....
                SAY " Recommencer le traitement ? (O/N)"
                LET REPONSE
        NEXT
```

tant que la valeur de REPONSE est O ou OUI,
les traitements inclus dans la boucle seront effectues.

MCRI

SET : Pour ajouter un nouveau critere de selection (voir CRIT => SET)
, ou pour modifier un critere deja defini, il faut utiliser
l'instruction SET.

```
ex :    SET NUMBER 1000 2000
        SET OBSERVED 1-JAN-1996 31-JAN-1996
        FIND
```

trouve tous les spectres dont le numero est compris entre
1000 et 2000 et dont l'observation a ete effectuee entre
le 1er et le 31 janvier 1996.

```
        SET OBSERVED 1-JAN-1996 31-FEB-1996
        SET LINE HI 21-cm
        FIND
```

effectue une nouvelle recherche avec un intervalle de date
plus etendu et une nouvelle contrainte sur le nom de la raie.

MESS

SAY : L'affichage a l'ecran d'un message s'obtient en utilisant
SAY de deux manieres possibles :

Afficher du texte :

```
ex :    SAY "Ceci est un message"
```

Afficher le contenu de variables :

```
        ex :    DEFINE INTEGER A
                GET FIRST
                SAY 'A''RY[3]'  ! affiche le contenu de A et
                                ! et la valeur du canal 3 du 1er spectre.
```

On peut combiner les deux possibilites :

```
        ex :    SAY "le contenu de A est :"'A'
```

MOVE

LET :
"=" : MOVE utilise comme operateur d'affectation, peut etre remplace par l'operateur mathematique classique precede eventuellement de l'instruction LET.

```
        ex :    DEFINE INTEGER A B
                LET A = B       ! met le contenu de B dans A
```

Cette affectation est la meme quelque soit le type de donnee : entiers, reels, caracteres, tableaux, booleens...

L'affectation globale d'un tableau est possible si leurs tailles sont identiques. On peut egalement affecter un tableau d'entiers a un tableau de reels ou vice versa. Dans ce dernier cas, les valeurs seront tronquees a leur partie entiere.

```
        ex :    DEFINE INTEGER TAB1[5]
                DEFINE INTEGER TAB2[5]
                DEFINE REAL    TAB3[5]
                DEFINE INTEGER TAB4[9]


                TAB1 = TAB2
                TAB2 = TAB3

                sont des affectations valides

                TAB3 = TAB4

                ne l'est pas car il n'y a pas coherence de taille.
```

MULT

"*" : Il s'agit de l'operateur classique de multiplication. on peut multiplier une variable unique ou multiplier un tableau en une seule instruction.

```
        ex :    DEFINE INTEGER TAB1[3]
                DEFINE REAL    TAB2[3]

                (ici aussi LET est optionnel)
                LET TAB1 = TAB1*2
                LET TAB2 = TAB2*TAB1

                sont des affectations valides
                On retrouve la contrainte de taille exprimee dans
                le paragraphe precedent.
```

PFIT

inexistant : pas de fonction pour avoir les coefficients d'un polynome qui fitte sur un spectre.
Voir BASE => BASE.

PLOT

HARDCOPY :
Fournie un fichier postscript de l'ecran graphique. cette fonction propose differentes options concernant l'orientation de la feuille, le peripherique de sortie...

C'est cette fonction aui permet dans CLASS d'effectuer une sortie vers une table tracante en utilisant l'option /PLOT.

ATTENTION : l'instruction PLOT existe dans CLASS mais sert a tracer le spectre courant a l'ecran, ainsi que son titre, et ses axes de coordonnees.

PRIN

SIC OUTPUT :
Il ne s'agit pas tout a fait d'un equivalent puisque SIC OUTPUT redirige les affichages SAY (voir MESS => SAY) dans un fichier de sortie.
Ce dernier est ecrit uniquement lorsque l'on ferme le fichier de sortie par l'instruction SIC OUTPUT.
Les affichages a l'ecran sont conserves.

ex :
```
SAY "ce message s'affiche a l'ecran"
SIC OUTPUT sortie.txt
SAY "ce message sera a l'ecran et bientot dans le fichier"
SIC OUTPUT              ! fermeture du fichier
SAY "Le second message est maintenant dans le fichier"
```

ATTENTION : l'instruction PRIN existe dans CLASS mais sert a rediriger vers l'ecran ou vers un fichier un certain nombre de parametres FIT, AREA, CHANNEL...

PUTF

GREG : On peut exporter un spectre dans un fichier binaire avec la commande GREG. Il n'est pas possible de relire ces fichiers avec CLASS. (il n'y a donc pas d'equivalent de GETF).

RDSY

"@" :
Les symboles (ou alias) sont une propriete du langage SIC et donc independant de CLASS. Si l'on souhaite conserver une sauvegarde de plusieurs symboles, cela doit se faire dans une procedure separee qui sera rappelee pour initialiser ces alias. On ne peut les sauvegarder dans un fichier resultat comme c'est le cas dans SIR.

La sauvegarde sera effectuee par SAVE qui ecriera un .CLASS.

SYMBOL donne la liste des symboles.
SYMBOL X donne l'equivalent dy symbole X
SYMBOL X "equivalent" defini un nouveau symbole.

READ

GET : Permet de charger un spectre dans le tableau R. Ce spectre est obligatoirement issu du fichier defini comme fichier d'entree a l'aide de l'instruction FILE IN. La destination du spectre sera le tableau R, l'ancien contenu de ce tableau etant recopie dans le tableau T. Si on lit un nouveau spectre, le contenu de T est alors perdu. On peut l'eviter en memorisant sont contenu a l'aide de MEMORIZE.

ex :
```
SET EXTENSION nan
FILE IN toto            ! sous entendu toto.nan
FIND/ALL
LIST
GET 1234                ! charge 1234 dans R
GET 1235                ! charge 1235 dans R et 1234 dans T
SWAP                    ! permute R et T
MEMORIZE save1          ! memorise T dans save1
SWAP
GET 1236                ! charge 1236 dans R
RETRIEVE save1          ! charge save1 dans R
```

GET sans option charge le 1er spectre de la liste
GET FIRST charge le premier
GET NEXT charge le suivant
GET LAST charge le dernier

REGR

inexistant

RESM

"@" :
On ne peut pas sauvegarder les parametres d'un masque dans les fichiers resultats. Cependant, SAVE permet de sauvegarder les parametres d'une session CLASS (parametres

etablis par SET) dans une procedure .CLASS.
Cette derniere peut donc etre rappelee par un "@" comme
n'importe quelle procedure.

RETU

RETURN :
Cette fonction de retour de procedure peut etre utilisee de la
meme maniere que dans SIR, cependant, etant donne que SIC (et
donc CLASS) propose des structures de programme (boucles et
tests) evolues, on s'arrangera toujours pour avoir un
algorithme qui, quelque soit le devenir des variables, arrive
en fin de procedure avant de terminer son execution.
Une instruction RETURN a la fin d'une procedure aura pour
role de terminer proprement l'algorithme en propageant
une eventuelle erreur a la procedure appelante.

REVE

inexistant :
Des lors que l'on peut manipuler les tableaux, on peut
reprogrammer ce genre de fonction.

ROTA

inexistant :
meme remarque que precedemment.

SAVM

SAVE : Il ne s'agit pas a proprement parle d'un equivalent puisque
le masque n'est pas sauve dans le fichier resultat mais dans
un fichier a part qui est en fait une procedure.
Cette procedure obtenue a l'aide de SAVE sauvegarde non
seulement les masques mais egalement tous les parametres qui
ont ete definis en interactif par SET.

voir RESM => "@"

SCAL

SET MODE :
Permet de preciser un intervalle de valeurs pour les
abscisses et/ou les ordonnees pour le trace des spectres.
SET MODE accepte differents arguments :

SET MODE X pour preciser l'intervalle des abscisses.

ex : SET MODE X AUTO
SET MODE X 1800 2200

definissent respectivement une echelle d'abscisse automatique
en fonction des valeurs trouvee dans le fichier, une echelle
d'abscisse definie par l'utilisateur.
Les valeurs precisees dans la 2eme forme doivent etre en accord
avec les unites choisies par SET UNIT.

SET MODE Y fonctionne de la meme maniere pour definir
l'intervalle des ordonnees.

SDCL

SYSTEM :
SHELL :
La commande qui calque le SDCL de SIR est SYSTEM. Cette
instruction permet de lancer une commande systeme (DCL si
c'est un environnement VMS, KSHELL par ex. si c'est un
environnement UNIX).

ex : SYSTEM ls

./
../
essai1.class
essai2.class
stamp_carre.class
timer.class

SYSTEM "ls -n"

-rw-r--r-- 1 207 3000 744 Jul 22 15:00 essai1.class
-rw-r--r-- 1 207 3000 186 Jul 24 12:46 essai2.class
-rw-r--r-- 1 207 3000 889 Jul 29 15:37 stamp_carre.class
-rw-r--r-- 1 207 3000 882 Jul 24 17:40 timer.class

Les commandes systemes qui necessitent des arguments doivent
etre mises entre guillemets.

rem : On peut lancer ainsi l'execution d'un programme externe
(programme C compile par exemple). Le passage d'arguments a
un programme C est eventuellement possible par le biais des
arguments de la ligne de commande, moyennant quelques astuces
et contraintes qui ne seront pas discutes ici.

La commande SHELL bascule vers le prompt du systeme. On peut
y effectuer l'ensemble des commandes systeme habituelles.
Pour revenir a CLASS il faut taper EXIT.

SETM

SET MASK :
Defini un intervalle de points a masquer. La fonction autorise
deux argument qui representent le point de depart et le point
d'arrivee du masque.
On peut obtenir l'intervalle masque ulterieurement en utilisant
la fonction SHOW MASK. DRAW MASK permet de symboliser le
masque en surrimpression du graphique du spectre. Les valeurs
specifiees pour le masque doivent etre exprimees dans l'unite
choisie par SET UNIT.

```
ex :    SET MASK 100 200        ! exprime en canaux
        SHOW MASK               ! affiche l'intervalle 100 200
        DRAW MASK 0.5           ! "trace" le mask en ordonnee 0.5
```

Voir aussi CLIP => SET MASK
=> SET WINDOW

SHIF

MODIFY :
MODIFY peut etre utilisee avec differentes options.
MODIFY FREQUENCY
MODIFY VELOCITY
MODIFY RECENTER en ce qui concerne le lien avec SHIF.

Cela permet de modifier l'echelle des frequences (FREQUENCY),
de modifier le canal de reference (RECENTER ou LET REFERENCE x)
et de recentrer ce canal de reference sur une frequence donnee
(MODIFI VELOCITY).
L'axe des abscisses n'est modifie en consequence qu'a l'issue
d'un nouveau PLOT ou CLEAR puis BOX.

SHOW

PLOT : Trace le spectre dans la fenetre graphique. Cette fenetre doit
avoir ete prealablement definie par DEVICE XAUTO WHITE (pour un
terminal X). Un fois le spectre charge en memoire par GET (voir
LOAD => GET et READ => GET), PLOT affiche successivement une
fenetre vide (CLEAR), la boite determinant les abscisses et les
ordonnees (BOX), le spectre (SPECTRUM) et le titre (TITLE).
Ce trace prend toute la place disponible dans la fenetre.
(voir aussi LOOK => STAMP et les remarques sur SET BOX_LOCATION).

ATTENTION : L'instruction SHOW existe dans CLASS mais sert
a afficher dans la fenetre alphanumerique different parametres
definis par SET (SET MASK, SET NUMBER...)

SHOM

DRAW MASK :
On peut representer les intervalles masques par SET MASK ou
par SET WINDOW sur le graphe en effectuant un DRAW MASK ou
un DRAW WINDOW. Il est possible de specifier l'ordonnee a
laquelle cet affichage va etre effectue.

voir aussi SETM => SET MASK

SIGM

SIGMA : variable globale du header ????

SINR

SINH :
SIN : voir COSR => COSH
=> COS

SMOO

SMOOTH :
On retrouve des types de lissage communs a SIR :
par defaut, lissage "hanning-smooth"
"boxcar-smoothing" ...

Le lissage est effectue sur le tableau R, ce dernier etant prealablement copie dans le tableau T.

SOMM

SUM : Integre les spectres dont les numeros sont dans le tableau INDEX. Ces numeros sont definis par la fonction FIND qui travaille sur les criteres de recherche qui lui sont imposes par l'instruction SET.
Voir CRIT => SET et CIDX => FIND
SUM verifie la coherence des positions des spectres ainsi que l'homogeneite des calibrations.

SORT

inexistant :
Il faut creer une procedure de tri. Les seuls tris efficaces du marche etant bases sur des algorithmes recursifs, il sera difficile d'obtenir un tri rapide a partir d'une procedure CLASS puisque la recursivite y est interdite.

SQRT

SQRT : Meme fonction que dans SIR. Elle retourne la racine carre de la valeur qui lui est passe en argument.

ex : DEFINE INTEGER A
DEFINE REAL B
LET A = 5
LET B = SQRT(A)

STOP

EXIT : Instruction de terminaison du programme CLASS.

SUBT

"-" : Pour effectuer une soustraction, utiliser l'operateur mathematique classique "-".

ex : DEFINE INTEGER A
DEFINE REAL B C
LET A = 3
LET B = 2.5
LET C = (A-B)

TYPE

SAY : voir MESS => SAY

VISM

SHOW MASK :
visualise a l'ecran les masque actuellement definis.

WRIT

WRITE :Ecrit le spectre actuellement dans le tableau R dans le fichier de sortie (determine par FILE OUT nom_fic).
Si le numero du spectre existe deja, une nouvelle version est cree, laissant intact la version precedente.
Sinon un nouveau spectre est cree dans ce fichier.

L'instruction UPDATE permet d'ecraser un spectre apres modifications. Le spectre de depart est efface puisque le nouveau le remplace avec toutefois l'incrementation du numero de version.

```
        ex :    FILE BOTH toto.nan      ! en entree et en sortie
                FIND/ALL
                GET FIRST
                SET UNIT C
                KILL 100
                WRITE
```

ecrit les modifications dans une nouvelle entree du fichier resultat qui sera placee a la fin.

```
                FILE IN toto.nan
                FILE OUT essai.nan NEW
                FIND/ALL
                GET FIRST
                WRITE
```

si "FIRST" n'existe pas dans essai.nan, il sera cree.

```
                FILE BOTH toto.nan
                FIND/ALL
                GET FIRST
                UPDATE
```

On ecrase le spectre designe par FIRST. A l'issue de l'UPDATE, il n'y aura pas de spectre en plus dans le fichier mais le numero de version de "FIRST" aura ete incremente quand meme.

WTSY

inexistant :
        voir RDSY => inexistant

INSTRUCTIONS CLASS N'AYANT PAS D'EQUIVALENT DANS SIR.

beaucoup d'instructions de CLASS ne possedent pas d'equivalent sous SIR, soit parce que le concept n'existe pas (par exemple les nombreuses manipulations graphiques de CLASS) soit parce que CLASS offre une fonctionnalite de plus dans un domaine tout de meme present dans SIR.
Les instructions suivantes presentes les "inovations" les plus importantes.

La notion de type de variables :

Chaque variable SIC (et donc CLASS) possede un type c'est a dire qu'on lui attribue, des sa creation, un domaine d'existance en terme de valeurs. Une variable de type INTEGER ne pourra contenir que des valeurs entieres, le type CHARACTER n'acceptera que des valeurs alphanumeriques, etc...
On rencontre ainsi des INTEGER, CHARACTER, REAL, LOGICAL.
On peut evoquer ici la possibiliter de declarer une fonction de la meme maniere qu'une variable. Cela a ete discute dans la comparaison ci-dessus : EVAL => inexistant.

Manipulations graphiques :

Ce domaine est plus riche dans CLASS que dans SIR et apporte donc beaucoup de nouveautes.

DRAW TEXT :

Afficher du texte sur un graphique a une position determinee par ses coordonnees en centimetres.

BOX, SPECTRUM, TITLE :
CLEAR SEGMENT:
CHANGE VISIBILITY segname OFF:

On peut afficher uniquement certaines portions d'un graphique, c'est a dire la fenetre et ses axes gradues, le spectre, son titre. Ces elements sont appeles par CLASS des segments. Le segment le dernier afficher peut etre effacer par CLEAR SEGMENT et ZOOM REFRESH. Cela n'est valable que pour la derniere operation graphique (principe des piles Last In First Out). Si l'on souhaite effacer un segment qui n'est pas le dernier on peut avoir recours a CHANGE VISIBILITY spectrum OFF puis ZOOM REFRESH.

SET BOX_LOCATION :

Determine les positions d'une boite destinee a contenir un
spectre. On peut ainsi afficher autant de boites que l'on veut
aux emplacements que l'on desire. La commande SPECTRUM dessinera
le spectre dans la derniere boite definie mais rien n'empeche
de reiterer la commande SET BOX_LOCATION sur une position deja
existante pour en effacer le spectre ou en mettre un nouveau.

CREATE DIRECTORY :
CHANGE DIRECTORY :
CREATE WINDOW :
CLEAR TREE, WINDOW, WHOLE :

Il est possible d'utiliser plusieurs fenetres (au sens X11) en
meme temps dans CLASS. L'existance de ces fenetres est construite
sous forme d'arbres. CREATE DIRECTORY permet de creer une nouvelle
instance de la classe fenetre, identiques a la fenetre par defaut
ouverte par DEVICE XAUTO. Pour aller travailler dans cette fenetre
il faut changer de repertoire (CHANGE DIRECTORY).
On peut egalement creer des fenetres de taille definie grace a
CREATE WINDOW. CLEAR TREE, WINDOW, WHOLE efface l'arborescence des
fenetres, une fenetre ou la totalite de la structure de description.
"

M.a.j. le 14 fevrier 2002 - J.M. Martin

Documents de travail :

1. Publication JPL 86-2 (15/12/1985) : format FITS pour IHW.
2. FITS user's guide : ftp://nssdc.gsfc.nasa.gov/pub/fits (v. recente, mot de passe (!)), ou plutot http://fits.gsfc.nasa.gov/fits_home.html.
3. Documentation sur le serveur http://www.cv.nrao.edu/fits.
4. Ce qui se passe a **Green Bank** : - pour les donnees du 140 pieds, et les documents du GBT en ligne.
5. Un exemple de lecture de fichier FITS avec CLASS (P.Colom).

Les buts :

- Pouvoir exporter les données au format FITS, à partir du logiciel spécifique 'RT' de pré-traitement des données, et si possible à partir des données brutes via un programme de conversion.
- Le format FITS utilise doit être reconnu par les logiciels étrangers :
    1. AIPS++ le nouveau logiciel du GBT et de Parkes,
    2. CLASS, le logiciel développé à l'IRAM et à l'Observatoire de Grenoble.

FITS

```
    CFITS  cookbook                P.Colom    November 24, 1997
                                   J.M.Martin January 13, 2004

CFITS is a tool that permits format translation between FITS format and CLASS
format. It is part of the GILDAS package.

1) to export a CLASS file toward FITS format.
 CFITS> file in toto.30m          ! toto.30m is a CLASS binary file
 CFITS> find/all                  ! find all spectra
 CFITS> get f                     ! get first spectrum
 CFITS> fits\write toto.fits      ! writes in FITS format (in toto.fits)

2) to import a FITS file toward CLASS format.
 CFITS> read toto.fits            ! reads a FITS format file
 CFITS> file out toto.30m new     ! creates a new CLASS file (empty)
 CFITS> las\write                 ! writes in CLASS format (in toto.30m)


3) when there are many FITS files to import (all located in the same folder),
 one should create a CFITS procedure with the following shell script:

echo "file out data.nrt new" > fits2class.pro
ls -1 foldername | awk '$1 ~ /.fits/ {print "read",$1,"\nlas\\write"}'  >> fits2class.pro
echo "" >> fits2class.pro
chmod u+x fits2class.pro

remarks:
1) in order to obtain a summary of the CFITS commands:
 CFITS> help fits\

2) There are a few commands that pertain to both LAS and CFITS. To distinguish
them, you must to precise the langage. Example:
 CFITS> las\write       ! to write in CLASS format
 CFITS> fits\write      ! to write in FITS format
```

Last modification: January 13th, 2004. P. Colom, J.-M. Martin. ***W3 validator***

Listing of the `stokes.class` procedure

```
!    stokes.class
!
! written by : P. Colom, with help from Eric Gérard
!              Feb. 2008
! aims: stokes parameters, linear polarisation parameter, PA, (U/Q)
!
! you need 2 scans: first with 0 and 90 deg   (Rmer = -45)
!                   second    +45    -45      (Rmer =   0)
!
! to fit a baseline, you need to mask the line and the spectrum start/end
! channels
define character outfile*20, infile*20
define integer first_scan last_scan
define real Vel_start Vel_end Vmin Vmax
!
GREG1\SET /DEFAULT
GREG1\PENCIL /DEFAULT
say "input file (MAX 20 char) ? "
let infile =
say "output file (MAX 20 char, new name) ? "
let outfile =
!
file in 'infile'
file out 'outfile' new
!
say "first scan (0°,90°) :"
let first_scan =
say "last scan (+45°,-45°) :"
let last_scan =
!
set scan first_scan first_scan       ! angles : first, second = 0, 90°
find /all
list
set format brief    ! brief header for plot
set weigh equal     ! equal weight for sum or accu
get f
! define dimension of angle psi
define real psi[channels]
plot
get n
plot
accu                ! Stokes I
plot
say " eliminate spectrum start/end channels : Vel_start and Vel_end"
let Vel_start =
let Vel_end =
say " baseline fit : you need to mask the line between Vmin & Vmax"
let Vmin =
let Vmax =
!
set mode x Vel_start Vel_end       ! eliminates channels at both ends
set window Vmin Vmax               ! masks the line
base 2 /plot                   ! second order polynomial fit
! pause " Stokes I -- type cont to go ahead"
!
memorize I1
get f
get n
multiply -1
accu                ! Stokes Q
plot
base 2 /plot
memorize SQ
! pause " Stokes Q -- type cont to go ahead"
get n               ! LCP
get n               ! RCP
multiply -1
accu                ! Stokes V
plot
base 2 /plot
memorize V1
! pause " Stokes V -- type cont to go ahead"
!  and now: rotation of feed horn by 45 deg
set scan last_scan last_scan       ! angles : first, second = = +45, -45
find /all
list
get f
plot
get n
plot
accu                ! Stokes I
plot
```

```
base 2 /plot
memorize I2
! pause " Stokes I -- type cont to go ahead"
get f
get n
multiply -1
accu                ! Stokes U
plot
base 2 /plot
memorize SU
! pause " Stokes U -- type cont to go ahead"
get n               ! LCP
get n               ! RCP
multiply -1
accu                ! Stokes V
plot
base 2 /plot
memorize V2
! pause " Stokes V -- type cont to go ahead"
!
! compute averages of I and V, and ratio U/Q
retrieve I1
retrieve I2
accu
mult 0.5
memorize SI         ! Stokes I
plot
write
pause " average of Stokes I -- type cont to go ahead"
retrieve SQ
write
plot
pause " Stokes Q -- type cont to go ahead"
!  degree of linear polarization
memorize Pl         ! Q in Pl
retrieve Pl
let RY = RY*RY      ! Q^2
memorize Pl
retrieve SU
let RY = RY*RY      ! U^2
retrieve Pl
accu                ! Q^2 + U^2
let RY = sqrt(RY)
retrieve SI
swap                !  Exchange the contents of the R and T buffers
divide 0.5          !  sqrt(Q^2 + U^2) / I
memorize Pl
!
retrieve SQ
retrieve SU
write
plot
pause " Stokes U -- type cont to go ahead"
divide 0.5           ! divide U by Q  (0.5 is a threshold, avoid /0)
plot
pause " U/Q  -- type cont to go ahead"
!
psi = 0.5*atan(ry)        ! position angle
psi = psi*180./pi
ry = psi
set mode y -90 90
plot
write
pause " psi = 1/2 arctan(U/Q) -- type cont to go ahead"
!
retrieve Pl
set mode y 0 1
plot
write
pause " degree of linear polarization -- type cont to go ahead"
!
retrieve V1
retrieve V2
accu
mult 0.5
set mode y total
plot
write
pause " average of Stokes V -- type cont to go ahead "
set scan * *            ! release constraint on scan number
say " ------------------------------------------------"
say " conclusion: we have written in " 'outfile'
say " 1) "
say " 2)  Q "
say " 3)  U "
say " 4)  psi = 1/2 arctan(U/Q) "
```

```
say " 5)  degree of linear polarization"
say " 6) "
say " -------------- end of stokes.class -----------------"
```

P. Colom - March 26, 2008

Listing of the `abb.class` procedure

```
symbol bank1 "las\set telescope NANCAYRT-B1"    ! correlator banks
symbol bank2 "las\set telescope NANCAYRT-B2"
symbol bank3 "las\set telescope NANCAYRT-B3"
symbol bank4 "las\set telescope NANCAYRT-B4"
symbol bank5 "las\set telescope NANCAYRT-B5"
symbol bank6 "las\set telescope NANCAYRT-B6"
symbol bank7 "las\set telescope NANCAYRT-B7"
symbol bank8 "las\set telescope NANCAYRT-B8"
symbol fa "find /all"
symbol fi "file in spectra.nrt"       !  Nancay Radio Telescope file
symbol gn "get next"
symbol gf "get first"
symbol mx "las\set mode x"
symbol my "las\set mode y"
symbol b0 "base 0 /plot"
symbol b1 "base 1 /plot"
symbol b2 "base 2 /plot"
symbol b3 "base 3 /plot"
symbol b4 "base 4 /plot"
symbol cp "clear plot"
symbol ca "clear alpha"
symbol nrt "las\set extension .nrt"
symbol pico "las\set extension .30m"
set plot histo
set format long
nrt                                  ! default extension : NRT
```

PColom - April 30, 2002