GIPSY Programmers Guide

by K.G. Begeman

(draft)

This manual is designed for persons wishing to write programs using the Groningen Image Processing SYstem (GIPSY). It should be useful for a wide range of applications from making minor changes in existing programs to writing major new applications routines. All basic aspects of GIPSY programming are dealt with in some detail.

Contents

Chapter 1

Introduction

The aim of this document is to introduce new applications programmers to the GIPSY programming environment. It is assumed that the new applications programmer is acquainted with some basics about computer languages, so that he or she can easily learn how to write GIPSY programs in SHELTRAN or FORTRAN. More experienced programmers can write GIPSY applications in C. Furthermore it is assumed that the applications programmer is an experienced GIPSY user, or has at least read the GIPSY USERS GUIDE.

The SHELTRAN (or S77) compiler is a simple FORTRAN (or F77) pre-compiler which forces you to write structured programs. Therefore it is strongly recommended to write your applications in SHELTRAN so that other programmers can also understand your code and adapt it for their special purposes. (All examples shown in this document are written in S77.)

This document will show you how to get parameters from the user, how to move a chunk of data from the mass storage device into your program and vice versa, how to get the physical coordinates of this chunk of data, how to use the GDS-Table system, how to make plots and how to display your data on a monitor. The routines which perform these functions will not be discussed in great detail meaning that only their function will be described and some important arguments. Each GIPSY routine is accompanied by a document with an extensive description of its function and arguments. Whenever a routine is mentioned in this document, it will appear *SLANTED* in the text. SHELTRAN statements will appear `LIKE THIS`.

Chapter 2

User Interface

2.1 Introduction

GIPSY programs do not interact with the user directly, but via the master control program HERMES. Whenever a program needs some input from the astronomer, it will send a command to HERMES who will then prompt the user for the required parameters. The same goes for informational messages intended for the user. If a program wants to inform the user that the integrated flux density of a source is xxx Janskys, it will not do this with an ordinary `WRITE` statement, but via HERMES. In this chapter we will describe the routines which communicate with HERMES.

2.2 User Interaction

The first thing an applications program must do when it is running is to tell HERMES that it's alive and ready for communication. This is done with subroutine *INIT*. A call to *INIT* must always be the first executable statement in an applications program. Likewise we need to inform HERMES that the program has finished it's job. This is done with a call to *FINIS* (always just before the `STOP` or `END` statement).

2.3 User Input Routines

Input from the user is requested with the *USERxxx* routines. There are eight of those routines, each for different types of parameters.

- *USERINT* is for `INTEGER` type of parameters.
- *USERLOG* is for `LOGICAL` type of parameters.
- *USERREAL* is for `REAL` type of parameters.

- *USERDBLE* is for `DOUBLE PRECISION` type of parameters.
- *USERCHAR* is for `CHARACTER` type of parameters. The characters typed in by the user are returned so that words (separated by spaces or commas) end up as separate items of the character array.
- *USERCHARL* same as *USERCHAR*, but now all characters are converted to lower case.
- *USERCHARU* same as *USERCHAR*, but now all characters are converted to upper case.
- *USERTEXT* if for `CHARACTER` type of parameters, but now all characters are passed on as they were typed in by the user.

All these routines are integer functions which return the number of parameters entered by the user.

```
Use:          INTEGER USERxxx( ITEMS    ,  Output   xxx ARRAY
                               MAXITEM  ,  Input    INTEGER
                               DEFAULT  ,  Input    INTEGER
                               KEYWORD  ,  Input    CHARACTER*(*)
                               MESSAGE  )  Input    CHARACTER*(*)
              USERxxx   Returns number of items entered by user.
              ITEMS     Array with decoded values entered by user.
              MAXITEMS  Maximum number of items (not present for
                        USERTEXT).
              DEFAULT   Specifies type of HERMES' prompt.
              KEYWORD   Keyword to prompt user with.
              MESSAGE   Message for user.
```

One of the arguments of the *USERxxx* routines is the default level. This default level determines the type of HERMES' prompt.

DEFAULT LEVELS	
Value	Meaning
0	No default possible
1	Default possible
2	Keyword is hidden and default possible
+4	Exact number of items required

Input from the user concerning the data set where the applications program is supposed to work on is obtained with the *GDSxxx* routines. There are four of these routines:

- *GDSINP* which is used to get the input data set, i.e. where the program is supposed to get the data from. This routine will then check whether the set (and subsets) entered by the user exist. If not, the user is informed and will be prompted again to enter input set name.
- *GDSOUT* which is used to get the output data set, i.e. where the program is supposed to write its modified data to. If this output set does not exist, *GDSOUT* will create a new data set, if it does exist *GDSOUT* will check whether the size of the input subsets and output subsets agree and ask the user if it's allowed to overwrite the existing data set.
- *GDSBOX* which will prompt the user to select a certain part of the subset to work on by the program
- *GDSPOS* will prompt with keyword 'POS=' for positions in a subset.

GDSINP and *GDSOUT* are integer functions which return the number of subsets entered by the user. *GDSPOS* is also an integer function which returns the number of positions entered by the user. *GDSBOX* is a subroutine. These routines are discussed in more detail in chapter 4.

2.4 User Output Routines

Output to the user is possible in GIPSY in various ways on various levels. With subroutine *ANYOUT* you can send text strings to the screen and/or to HERMES' logfile.

```
Use:          CALL ANYOUT( DEVICE  ,   Input    INTEGER
                           MESSAGE )   Input    CHARACTER*(*)

              DEVICE    Device to which the text is send.
              MESSAGE   Text for user.
```

The possible devices are listed in the table below.

ANYOUT DEVICE	
Device	Meaning
0	HERMES default device
+1	Output to terminal
+2	Output to HERMES' logfile
4	Reserved
8	Output for inexperienced users only, disabled with HERMES' switch /MODE 0
16	Output in test mode only, enabled with HERMES' switch /MODE 2

Subroutine *ERROR* is used to display error messages on the screen. The severity of the error is determined by the programmer and the user. The programmer specifies the error level in the call to *ERROR*, which is then compared by HERMES with the error level set by the user (with /ERRLEV). Whenever the error level set by the programmer is greater than or equal to the level set by the user (default 4), then the program will be aborted automatically.

```
Use:          CALL ERROR( ERRLEV   ,  Input   INTEGER
                          MESSAGE )   Input   CHARACTER*(*)

              ERRLEV    Level of error.
              MESSAGE   Message for user explaining the error.
```

ERROR LEVELS	
Level	Meaning
1	Warning message
2	Minor error
3	Serious error
4	Fatal error

Information about the status of the program, i.e. what your program is doing at the moment, is sent to the Task Status Area with subroutine *STATUS*. For example program CLEAN displays in the Task Status Area every 5 seconds how many components were found and subtracted sofar.

```
Use:          CALL STATUS( MESSAGE )  Input   CHARACTER*(*)

              MESSAGE    Message to be displayed in Task Status Area.
```

2.5 Keywords

For each call to one of the user-input routines, the user is prompted with a keyword in the User Command Area. The user then has to type in the required parameters. HERMES sends the parameters back to the servant task, and writes the keyword and the text typed in by the user to his internal keyword list. The next time the same keyword is asked by the servant, HERMES will first look in his internal list whether there was already some input associated with that keyword. If so, he will not prompt the user again but take the parameters which were entered earlier by the user. Sometimes this feature is not wanted. With the subroutine *CANCEL* one can tell HERMES to remove a keyword (and its parameters) from the internal list. This is usually done when the user supplied wrong input. The program checks whether the input is wrong, and if so then it cancels the keyword and prompts the user again.

```
Use:          CALL CANCEL( KEYWORD )   Input   CHARACTER*(*)

              KEYWORD    Keyword to be cancelled from HERMES'
                         keyword list
```

One can also write keywords to HERMES' keyword list. This is done with the subroutine *WKEY*.

```
Use:          CALL WKEY( TEXT )   Input   CHARACTER*(*)

              TEXT     Keyword plus parameters tobe put in HERMES'
                       keyword table.
```

2.6 Spawning other GIPSY Tasks

For GIPSY application programmers it is sometimes useful to spawn another GIPSY task. For example the program MNMX is sometimes spawned to update the minimum and maximum in the output set. This is done with the subroutine *DEPUTY*.

```
Use:          CALL DEPUTY( TASK ,    Input    CHARACTER*(*)
                           IERD )    Output   INTEGER

              TASK     Name of servant task to spawn.
              IERD     Return code containing information about the fate
                       of the command.
```

The deputized program uses the same keyword list in HERMES as the initiating program. So with subroutine *WKEY* one can coordinate the action of the deputized program. For simple programs like MNMX it is not necessary to overwrite HERMES' keyword list. With a call to *SUBST* one can substitute keywords. For example if a certain program wants to deputy MNMX to update the minimum and maximum of an output set just created by that same program, he just needs to substitute the keyword 'SET=' (indicating the input set for MNMX) with the keyword 'SETOUT=' (indicating the output set of the program who just created this set).

```
Use:          CALL SUBST( TEXT ,    Input    CHARACTER*(*)
                          IERS )    Output   INTEGER

              TEXT      Text containing keyword pairs. The first member
                        of the pair is the keyword as referenced by the
                        spawned task; the second is the keyword as
                        referenced by the calling task.
              IERS      Error return code.
```

2.7 Example

The following example program shows how to use the User Interface routines. This program is also available from Update.

```
      PROGRAM EXAMPLE1

      CHARACTER*80 MESSAGE
      INTEGER      ARRAY(10)
      INTEGER      I
      INTEGER      DEFAULT
      INTEGER      USERINT, USERTEXT, USEREAL
      REAL         H0

N     Get in touch with HERMES
      CALL INIT
      CALL ANYOUT( 1, ' This message is for the screen only ' )
      CALL ANYOUT( 2, ' This message is for the log file only ' )
      CALL ANYOUT( 1+2, ' This message to screen AND log file ' )
      CALL ANYOUT( 8, ' This message you will not see with /MODE 0 ' )
      CALL ANYOUT( 16, ' This message you will only see with /MODE 2 ' )
N     Set default level to no default
      DEFAULT = 0
N     Get an integer from the user
      NEL = USERINT( ARRAY, 10, DEFAULT, 'INT=', 'Type some integers' )
N     Thanx
      CALL ANYOUT( 1, ' Thank you very much for this input ' )
N     Prepare message
      WRITE(MESSAGE,'('' Values are:'',10(I4,1X:))') (ARRAY(I),I=1,NEL)
N     Send message to HERMES
      CALL ANYOUT( 3, MESSAGE )
N     Display what we are doing
      CALL STATUS( 'Now I will count from 1 to 1000000 ' )
      I = 0
      WHILE (I .LT. 1000000)
        I = I + 1
      CWHILE
N     Prepare default
      MESSAGE = 'JURJEN'
N     Allow default
      DEFAULT = 1
N     Now ask user his name
```

```
      NEL = USERTEXT( MESSAGE, DEFAULT, 'NAME=', 'Enter your name ['//
     #                MESSAGE(1:6)//']' )
N     Was default used ?
      IF (NEL .EQ. 0)
      THEN
N       Number of characters in default name
        NEL = 6
      CIF
N     Demonstrate use of error messages
      IF (MESSAGE(:4) .EQ. 'MR.X')
      THEN
        CALL ERROR( 4, ' I do NOT work for Mr. X. ' )
      CIF
N     Show user his name
      CALL ANYOUT( 0, MESSAGE(:NEL)//', I will now demonstrate WKEY' )
N     Now write keyword to HERMES' keyword list
      CALL WKEY( 'HUBBLE=42' )
N     Now ask keyword HUBBLE
      NEL = USERREAL( H0, 1, 0, 'HUBBLE=', 'First time we ask' )
N     You didn't have a chance to specify H0
      WRITE( MESSAGE, '(F12.4)') H0
      CALL ANYOUT( 0, ' H0 = '//MESSAGE(1:12) )
N     Now I will cancel keyword 'HUBBLE='
      CALL CANCEL( 'HUBBLE=' )
N     Now ask keyword HUBBLE
      NEL = USERREAL( H0, 1, 0, 'HUBBLE=', 'Second time we ask' )
N     Now you had your chance to specify H0
      WRITE( MESSAGE, '(F12.4)') H0
      CALL ANYOUT( 0, ' H0 = '//MESSAGE(1:12) )
N     Tell HERMES we're finished
      CALL FINIS
      STOP
      END
```

Chapter 3

Data Base Interface

3.1 Introduction

In this chapter we will learn how to access your data. The facility which is used in GIPSY for storing and retrieving image data is called GDS (GIPSY Data System). Image is used in a very general sense: it can be any equally spaced 0...N-dimensional structure of floating point numbers: spectra, images, Westerbork cubes, etc. The image data is accompanied by a file in which information can be stored which describe the image or is related to it. This is the so called descriptor file. The descriptor file can also exist without image data, but image data cannot exist without a descriptor file.

The descriptor (or header) file describes the structure and coordinate system of the data in the image file with FITS keywords. The keywords that describe the size and dimensionality of the image are:

- 'NAXIS' specifies the number of coordinate axes in the image.
- 'NAXIS1 specifies the number of pixels along the most rapidly varying axis within the image.
- 'CTYPE1' type of physical coordinate along first axis.
- 'NAXIS2' specifies the number of pixels along the second most rapidly varying axis within the image.
- 'CTYPE2' type of physical coordinate along second axis.
- etc.

So each N-dimensional database, or set as it is called in GIPSY, has N different coordinate axes, which is specified in the descriptor file by descriptor 'NAXIS'. Each axis has a unique name specified by 'CTYPEn' and a number of pixels 'NAXISn', where n is the sequence number of the axis, running from 1 to 'NAXIS'. These FITS descriptors hold the information of how the set is stored on the mass storage device (disk).

3.2 Coordinates

In GDS there are three levels of coordinates:

- File coordinates. They are directly related to the position in the file. For example, a three dimensional set with axis lengths $N1$, $N2$ and $N3$ has file coordinates ranging from 1 to $N1$, 1 to $N2$ and 1 to $N3$. Here (1,1,1) corresponds with the first data value in the image file, and $(N1, N2, N3)$ corresponds with the last data value. The position fp in the image file of an arbitrary position (f_1, f_2, f_3) in the set is calculated from:

$$fp = 1 + (f_1 - 1) + (f_2 - 1) \times N1 + (f_3 - 1) \times N1 \times N2$$

  This position is called a coordinate word, which is nothing more than a packed representation of a vector of file coordinates in the N-dimensional structure. In GDS a bigger range for the coordinates is used than is actually necessary for addressing the pixels. A GDS coordinate word is calculated from:

$$cw = f_1 + f_2 \times (N1 + 1) + f_3 \times (N1 + 1) \times (N2 + 1)$$

  In this extended range file coordinates run from 0 to $N1$, 0 to $N2$, etc. So an extra file position is added per axis which is not associated with a position in the image file. In this way it is possible to designate substructures called subsets with coordinate words by setting one or more file coordinates to zero. Consider the following example of a 4×5 image. The coordinate words for this image are as follows:

CWs for a 4×5 GDS image CWs are printed *slanted*.					
5	*25*	*26*	*27*	*28*	*29*
4	*20*	*21*	*22*	*23*	*24*
3	*15*	*16*	*17*	*18*	*19*
2	*10*	*11*	*12*	*13*	*14*
1	*5*	*6*	*7*	*8*	*9*
0	*0*	*1*	*2*	*3*	*4*
$f_2 \uparrow / f_1 \rightarrow$	0	1	2	3	4

For example $cw = 17$ designates a pixel at $f_1 = 2$ and $f_2 = 3$, $cw = 2$ designates a line at constant $f_1 = 2$ and $cw = 15$ designates a line at constant $f_2 = 3$. Consequently, $cw = 0$ designates the whole image (set). So with one coordinate word we can designate a subset. This is the so called subset coordinate word. If for an arbitrary axis the corresponding file coordinate is zero, we call this axis or dimension undefined. A subset coordinate word designates a subtructure with one or more undefined dimensions. The dimension of a subset is equal to the number of undefined dimensions in the subset coordinate word. The total number of different oriented subsets in a GDS set is equal to 2^{NAXIS}.

File coordinates are only of use to the programmer in sofar that he or she has to pass them along from one GDS routine to another. GDS routines which manipulate coordinate words all have prefix *GDSC_* and are discussed in section 3.4.

- Grid coordinates. These are coordinates which are mostly used by the programmer. Grid coordinates are related to file coordinates by the FITS descriptor 'CRPIX*n*'. 'CRPIX*n*' indicates the location of the reference pixel along axis n and is always of double precision type. The relation between file coordinates and grid coordinates is the following:

$$g_n = f_n - NINT(\text{`}CRPIXn')$$

Routines which extract grid coordinates from coordinate words or which insert grid coordinates in coordinate words are described in section 3.4.

- Physical or astronomical coordinates. These will be discussed in chapter 4.

3.3 The basic GDS routines

The basic GDS routines all have prefix *GDS_* and are usually of no interest for the applications programmer. They are discussed here very briefly only for the sake of completeness. The basic GDS routines are:

- *GDS_CLOSE* closes a GDS set.
- *GDS_CREATE* creates a new GDS set.
- *GDS_DELETE* deletes a GDS set.
- *GDS_EXIST* tests whether a specific GDS set exists.
- *GDS_EXTEND* increases the number of dimensions in a GDS set.

3.4 Manipulating coordinate words

The GDS routines which manipulate coordinate words all have prefix *GDSC_*. These routines are very important to the application programmer since they relate grid coordinates to coordinate words and vice versa. The following *GDSC_* routines are available:

- *GDSC_FILL* fills the undefined coordinates in a coordinate word with grids from an input array.
- *GDSC_GRID* extracts a grid coordinate from a coordinate word.
- *GDSC_INTERSECT* returns the coordinate word of a substructure which is the intersection of two other substructures.
- *GDSC_NAME* returns the name of an axis in a GDS set.
- *GDSC_NDIMS* returns the dimensionality of a substructure.
- *GDSC_ORIGIN* returns the location of the reference pixel of an axis.
- *GDSC_RANGE* returns two coordinate words denoting the lower left grid position and the upper right grid position in a subset.
- *GDSC_SIZE* returns the size of an axis in a GDS set.
- *GDSC_SUBSTRUCT* returns the subset coordinate word from two coordinate words containing grid positions in the subset.
- *GDSC_WORD* inserts a grid coordinate in a coordinate word.

We will discuss here briefly some of the most used *GDSC_* routines.

3.4.1 *GDSC_FILL*

The integer function *GDSC_FILL* fills the undefined coordinates with grid values taken from an array. It is primarily used to create coordinate words which denote the lower and upper file positions of the data an application needs to access.

```
Use:          INTEGER GDSC_FILL( SET    ,   Input    CHARACTER*(*)
                                 SUBSET ,   Input    INTEGER
                                 GRIDS  )   Input    INTEGER ARRAY

              GDSC_FILL     Returns coordinate word with the undefined
                            coordinates in SUBSET filled with grid
                            positions from array GRIDS.
```

```
SET          Name of GDS set.
SUBSET       Subset coordinate word.
GRIDS        Array with grid positions.
```

3.4.2 *GDSC_GRID*

The integer function *GDSC_GRID* extracts a grid position from a coordinate word. It is primarily used to get the grids of the lower and upper bounds of a subset.

```
Use:          INTEGER GDSC_GRID( SET    ,   Input   CHARACTER*(*)
                                 AXNUM  ,   Input   INTEGER
                                 CWORD  ,   Input   INTEGER
                                 GERROR )   Output  INTEGER

              GDSC_GRID    Returns the grid coordinate extracted from
                           coordinate word CWORD of axis number AXNUM.
                           GERROR will be equal to -19 if the coordinate
                           in CWORD is undefined.
              SET          Name of GDS set.
              AXNUM        Number of axis for which to extract the grid.
              CWORD        Coordinate word from which to extract the
                           grid.
              GERROR       Will be -19 when coordinate was not defined
                           in CWORD.
```

3.4.3 *GDSC_NDIMS*

The integer function *GDSC_NDIMS* returns the number of undefined coordinates in a coordinate word. In other words it returns the dimensionality of a subset.

```
Use:          INTEGER GDSC_NDIMS( SET    ,   Input   CHARACTER*(*)
                                  SUBSET )   Input   INTEGER

              GDSC_NDIMS   Returns the dimensionality of SUBSET.
              SET          Name  of GDS set.
              SUBSET       Subset coordinate word.
```

3.4.4 *GDSC_RANGE*

The subroutine *GDSC_RANGE* calculates for the undefined coordinates in a subset coordinate word the first and last file coordinate of the subset.

```
Use:         CALL GDSC_RANGE( SET    ,   Input   CHARACTER*(*)
                              SUBSET ,   Input   INTEGER
                              CWLO   ,   Output  INTEGER
                              CWUP   ,   Output  INTEGER
                              GERROR )   Output  INTEGER

             SET          Name of GDS set.
             SUBSET       Subset coordinate word.
             CWLO         Lower position in SUBSET.
             CWHI         Upper position in SUBSET.
             GERROR       GDS error code.
```

The next example demonstrates how to obtain the grid positions of the first and last file position in a GDS set. Note that the subset coordinate word which denotes the whole set has no defined coordinates, so it is zero.

```
C
C     GDSC_RANGE returns in CWLO and CWUP coordinate words which
C     contain the position of the first pixel and the last pixel in
C     the subset.
C
      CALL GDSC_RANGE( SET, 0, CWLO, CWUP, GERROR )
C
C     GDSC_NDIMS returns the dimensionality of the subset, so the number
C     of undefined dimensions.
C
      SETDIM = GDSC_NDIMS( SET, 0 )
C
C     GDSC_GRID extracts a grid coordinate from a coordinate word.
C
      FOR NAXIS = 1, SETDIM
         LOWERGRID(N) = GDSC_GRID( SET, NAXIS, CWLO, GERROR )
         UPPERGRID(N) = GDSC_GRID( SET, NAXIS, CWUP, GERROR )
      CFOR
C
```

3.5 Descriptors

The descriptor file (or “header”) is a hierarchical structure which allows the user to store information at every possible subset level by specifying the coordinates of that subset. All subsets, from the whole set down to the pixel level, can have descriptors of their own.

Descriptor items are FITS keywords like 'DATAMIN' or 'DATAMAX'. These two hold the information about the minimum and maximum value in a particular subset. So a descriptor item is fully identified by the combination of name and subset coordinates. GDS routines which manipulate descriptor items all have prefix *GDSD_*. A list of the most commonly used descriptor items is shown in Appendix A.

Descriptor items are read from the descriptor file with subroutine *GDSD_Rxxx*. There are five of those routines, each for different types of parameters.

- *GDSD_RINT* reads FITS descriptor of `INTEGER` type.
- *GDSD_RLOG* reads FITS descriptor of `LOGICAL` type.
- *GDSD_RREAL* reads FITS descriptor of `REAL` type.
- *GDSD_RDBLE* reads FITS descriptor of `DOUBLE PRECISION` type.
- *GDSD_RCHAR* reads FITS descriptor of `CHARACTER` type.

```
Use:          CALL GDSD_Rxxx( SET    ,   Input    CHARACTER*(*)
                              NAME   ,   Input    CHARACTER*(*)
                              SUBSET ,   Input    INTEGER
                              VALUE  ,   Output   xxx
                              ERROR  )   Output   INTEGER

              SET       Name of GDS database.
              NAME      Name of FITS descriptor item. Maximum length
                        is 8 characters.
              SUBSET    Subset coordinate word stating the level at
                        which the item is to be read.
              VALUE     The decoded value of the FITS keyword.
              ERROR     Error return code.
```

These descriptor reading routines start looking at the specified subset level and if the requested information is not found, they inspect higher subset levels until the information is found or proved to be not present. The hierarchy in subset levels can best be explained with the following example:

Consider a Westerbork data cube with 3 dimensions: RA, DEC, and FREQ. The 3 dimensions lead to $2^3 = 8$ different subset levels. For the following table we assume that "1" means that a coordinate has a specific value, while "0" means that a coordinate is undefined.

Dimension	RA	DEC	FREQ	Corresponding subset
3	0	0	0	Whole set
2	0	0	1	Map at constant FREQ
2	0	1	0	Map at constant DEC (L-V map)
2	1	0	0	Map at constant RA (M-V map)
1	0	1	1	Row in map
1	1	0	1	Column in map
1	1	1	0	Velocity profile
0	1	1	1	Pixel

For a row in a map at constant DEC and FREQ there are three higher levels. First there is a map at constant FREQ and a map at constant DEC, and second there is the whole set. So higher levels to a subset are simply found by subsequently undefining the defined file coordinates in the subset coordinate word. Then the total number of higher levels to a subset with dimension SUBDIM is: $2^{NAXIS-SUBDIM}$.

Descriptor items are written to the descriptor file with the *GDSD_Wxxx* routines. Similar to the descriptor read routines there are five of those write routines, each for different type of data. Contrary to the read routines the write routines only write at the specified level.

For obtaining the FITS items 'NAXIS', 'NAXIS%', 'CTYPE%' and 'CRPIX%' there are *GDSC_xxx* routines. These routines are respectively *GDSC_NDIMS*, *GDSC_SIZE*, *GDSC_NAME* and *GDSC_ORIGIN*. These descriptor items may never be written to the descriptor file by the applications programmer. To add an extra dimension to your set use routine *GDS_EXTEND* which will increment descriptor 'NAXIS' with 1 and create the descriptors 'CTYPE%', 'CRPIX%' and 'NAXIS%'.

3.6 Reading and writing image data

After having obtained the input and/or output set and subsets you (probably) need to read some data from/to the input/output subsets. This is done with the *GDSI_xxx* routines. There are three of those routines:

- *GDSI_CANCEL* which cancels a read/write tansfer.
- *GDSI_READ* reads (part of) a sub image.
- *GDSI_WRITE* write (part of a sub image.

GDSI_READ and *GDSI_WRITE* have the same kind of arguments.

```
Use:          CALL GDSI_READ( SET    ,  Input      CHARACTER*(*)
                              CWLO   ,  Input      INTEGER
```

```
                  CWUP   ,  Input      INTEGER
                  BUFFER ,  Output     REAL ARRAY
                  MAXBUF ,  Input      INTEGER
                  NREAD  ,  Output     INTEGER
                  TID    )  In/Output  INTEGER

SET     Name of GDS set.
CWLO    Coordinate word of first pixel in sub image
        to read.
CWLO    Coordinate word of last pixel in sub image
        to read.
BUFFER  Array where the data read from sub image are
        stored.
MAXBUF  Size of BUFFER.
NREAD   Number of pixels read from sub image.
TID     Unique transfer identifier. Should be set to
        zero when starting to read. The return value
        will be zero again when all pixels from the sub
        image have been read.
```

Each sequence of calls to one of those routines is associated with an unique identifier, the so called transfer identifier. Before the first call, 0 should be assigned to this identifier. When a number < 0 is returned, an error has been detected. A number > 0 means that the transfer has not been completed with this call and that further calls are necessary. These further calls should be made using the unmodified transfer identifier. If 0 is returned, the transfer has been completed successfully. With the help of this identifier, it is possible to handle subsets which contain more data than there is room for in your read/write buffer.

```
N      Reset transfer id for input data
       TIDI = 0
N      Reset transfer id for output data
       TIDO = 0
N      Start of read/write loop
       REPEAT
N         Read input data
          CALL GDSI_READ( SET, CWLO, CWUP, BUFFER, MAXBUF, NREAD, TIDI )
N         Scale data
          FOR N = 1, NREAD
            BUFFER(N) = 2.0 * BUFFER(N)
          CFOR
N         Write output data
          CALL GDSI_WRITE( SET, CWLO, CWUP, BUFFER, NREAD, NREAD, TIDO )
```

```
N     Stop if TIDI again zero
      UNTIL (TIDI .EQ. 0)
```

Chapter 4

Input and Output sets

4.1 Introduction

In this chapter we will learn how to obtain the input and output set and subsets from the user and how to obtain a position or a part of a sub image from the user where the application is supposed to operate on. The routines which are involved in obtaining this information all have prefix *GDS*. The following *GDSxxx* routines are available:

- *GDSASN* assigns the coordinate system of an input set to an output set.
- *GDSBOX* requests a part of a subset (box or sub frame) from user.
- *GDSCAS* changes the selected output subset grids.
- *GDSCPA* changes/adds a primary axis of/to an output set.
- *GDSCSA* changes/adds a secondary axis of/to an output set.
- *GDSCSS* changes the size of an output subset.
- *GDSINP* requests input set and subsets from user.
- *GDSOUT* requests output set and subsets from user.
- *GDSPOS* requests positions in a subset from user.

User request are only issued by *GDSBOX*, *GDSINP*, *GDSOUT* and *GDSPOS*. The other routines are used to set the default coordinate system for *GDSOUT*, since *GDSOUT* not only requests output set and subsets from the user, but also creates the output set if it does not yet exist. Furthermore, *GDSOUT* copies the relevant descriptor items from input set to output set. All these routines will be discussed in detail in the following sections.

4.2 Input set and subsets

GDSINP is intended only for obtaining input set and subsets (i.e. where your program gets its data from). *GDSINP* will decode the character string typed in by the user (like SET=AURORA FREQ 1:10) and check whether the set (in this case AURORA) exists. It will also check whether the subsets (in this case there are 10 subsets with defined coordinates along the FREQ axis) do exist. If they do not exist, *GDSINP* will inform the user and ask again for the input set. Also *GDSINP* will check whether the dimension of the subsets is what your program expects. If this is also not correctly given by the user he will also be informed about his mistake.

```
Use:          INTEGER GDSINP( SET      , Output     CHARACTER*(*)
                              SUBSETS  , Output     INTEGER ARRAY
                              MAXSUBS  , Input      INTEGER
                              DEFAULT  , Input      INTEGER
                              KEYWORD  , Input      CHARACTER*(*)
                              MESSAGE  , Input      CHARACTER*(*)
                              SHOWDEV  , Input      INTEGER
                              AXPERM   , Output     INTEGER ARRAY
                              AXCOUNT  , Output     INTEGER ARRAY
                              MAXAXES  , Input      INTEGER
                              CLASS    , Input      INTEGER
                              CLASSDIM ) In/Output  INTEGER

              GDSINP    Returns the number of subsets entered by user.
              SET       Name of GDS set selected by user.
              SUBSETS   Subset coordinate words of the subsets selected
                        by the user.
              MAXSUBS   Maximum number of subsets the application
                        allows.
              DEFAULT   Default level as in USERxxx.
              KEYWORD   Keyword for user prompting. Usually 'SET='.
              MESSAGE   Message for the user.
              SHOWDEV   Device as in ANYOUT to which some information
                        about the selected set is send.
              AXPERM    Contains the axes numbers in the order as
                        specified by user.
              AXCOUNT   Contains the number of grids along the axes
                        as specified by user in the order as specified
                        by user.
              MAXAXES   Maximum number of axes the application allows.
              CLASS     Class of application. CLASS=1 is for
```

```
          applications which repeat their operation for
          each subset specified, CLASS=2 is for
          applications which combine subsets in some way.
CLASSDIM  For CLASS=1 this item specifies the required
          dimension of the subsets. If CLASSDIM=0 then
          a subset may have any dimension.
          GDSINP returns in CLASSDIM the dimension of
          the subsets as selected by the user.
          For CLASS=2 this item specifies the required
          number of axes outside the subset. If CLASSDIM=0
          then the number of axes outside the subset may
          be any number greater than zero. GDSINP returns
          in CLASSDIM the number of axes outside the
          subset chosen by the user.
```

The total number of axes returned by *GDSINP* (read the dimensionality of the input set) can be obtained with *GDSC_NDIMS*, discussed in the previous chapter. The dimensionality of the subsets can also be obtained with *GDSC_NDIMS* or is returned in `CLASSDIM` for `CLASS = 1` applications.

GDSINP returns in `AXPERM` the axis numbers in the order specified by the user, and in `AXCOUNT` the number of grids along the axes in the same order. The `AXPERM` elements up to the dimension of the subset contain the axis numbers along the subset defined by the user in increasing order. So the first subset axis is always "faster" than the second subset axis, etc. The following elements in `AXPERM` contain the axis numbers of the axes which are specified by the user (so they are defined) in the order entered by the user. To illustrate this, consider for example a Westerbork data cube with name '1' and with 3 axes RA, DEC, FREQ with sizes N1, N2, N3 resp. In the following table the values of `AXPERM` are displayed as a result of the input from the user:

User input				Dimension of subset	AXPERM/AXCOUNT 1	2	3
SET=1				3	1/N1	2/N2	3/N3
SET=1	FR 1:63			2	1/N1	2/N2	3/63
SET=1	DE -4:5			2	1/N1	3/N3	2/10
SET=1	RA -7:8			2	2/N2	3/N3	1/16
SET=1	DE -4:5	FR 1:63		1	1/N1	2/10	3/63
SET=1	FR 1:63	DE -4:5		1	1/N1	3/63	2/10
SET=1	RA -7:8	FR 1:63		1	2/N2	1/16	3/63
SET=1	FR 1:63	RA -7:8		1	2/N2	3/63	1/16
SET=1	RA -7:8	DE -4:5		1	3/N3	1/16	2/10
SET=1	DE -4:5	RA -7:8		1	3/N3	2/10	1/16
SET=1	RA -7:8	DE -4:5	FR 1:63	0	1/16	2/10	3/63
SET=1	RA -7:8	FR 1:63	DE -4:5	0	1/16	3/63	2/10
SET=1	DE -4:5	RA -7:8	FR 1:63	0	2/10	1/16	3/63
SET=1	DE -4:5	FR 1:63	RA -7:8	0	2/10	3/63	2/10
SET=1	FR 1:63	RA -7:8	DE -4:5	0	3/63	1/16	2/10
SET=1	FR 1:63	DE -4:5	RA -7:8	0	3/63	2/10	3/16

So the contents of `AXPERM` can be used as input to some *GDSC_* routines as shown in the following example, where the lower and upper grid positions of a subset are determined:

```
C
C     GDSC_RANGE returns in CWLO and CWUP coordinate words which
C     contain the position of the first pixel and the last pixel in
C     the subset.
C
      CALL GDSC_RANGE( SET, SUBSET, CWLO, CWUP, GERROR )
C
C     GDSC_NDIMS returns the dimensionality of the subset, so the number
C     of undefined dimensions.
C
      SUBDIM = GDSC_NDIMS( SET, SUBSET )
C
C     GDSC_GRID extracts a grid coordinate from a coordinate word.
C
      FOR N = 1, SUBDIM
         LOWERGRID(N) = GDSC_GRID( SET, AXPERM(N), CWLO, GERROR )
         UPPERGRID(N) = GDSC_GRID( SET, AXPERM(N), CWUP, GERROR )
      CFOR
C
```

4.3 *GDSxxx* calls prior to *GDSOUT*

GDSINP will store the relevant coordinate information of the input set internally for later use by *GDSOUT*. The information stored is associated with the keyword used in *GDSINP*. If an application produces an output set, the coordinate system of the input set must be transferred correctly to the output set. Therefore *GDSOUT* needs to know where to copy the coordinate system from. This is done by assigning the internally stored coordinate information (by an earlier call to *GDSINP*) to an internal buffer for *GDSOUT*. This is done with subroutine *GDSASN*.

```
Use:          CALL GDSASN( GDSINPKEY,    Input    CHARACTER*(*)
                           GDSOUTKEY,    Input    CHARACTER*(*)
                           CLASS    )    Input    INTEGER

              GDSINPKEY   Keyword used in a previous call to GDSINP.
              GDSOUTKEY   Keyword to be used in a following call to
                          GDSOUT.
              CLASS       CLASS of application. CLASS=1 means that all
                          axes are copied to the internal GDSOUT buffer,
                          CLASS=2 means that only the subset axes are
                          copied.
```

Subroutine *GDSCSS* modifies the size of the output subsets. An application which makes use of this call is COPY.

```
Use:          CALL GDSCSS( GDSOUTKEY ,   Input    CHARACTER*(*)
                           LOWERGRID ,   Input    INTEGER ARRAY
                           UPPERGRID )   Input    INTEGER ARRAY

              GDSOUTKEY    Keyword associated with a following call
                           to GDSOUT.
              LOWERGRID    Array with the lower grid coordinates of
                           the output subset.
              UPPERGRID    Array with the upper grid coordinates of
                           the output subset.
```

With subroutines *GDSCPA* and *GDSCSA* the coordinate system of the output set can be changed. `CLASS=2` applications like MEAN use these routines to create an extra 'PARAM' axis, `CLASS=1` applications like REPROJ use it to change the coordinate system of the subsets.

```
Use:          CALL GDSCPA( GDSOUTKEY,   Input   CHARACTER*(*)
```

```
               AXNUM     ,   Input   INTEGER
               NAXIS     ,   Input   INTEGER
               CDELT     ,   Input   DOUBLE PRECISION
               CROTA     ,   Input   DOUBLE PRECISION
               CRPIX     ,   Input   DOUBLE PRECISION
               CRVAL     ,   Input   DOUBLE PRECISION
               CTYPE     ,   Input   CHARACTER*(*)
               CUNIT     ,   Input   CHARACTER*(*)
               PMASK     )   Input   INTEGER

GDSOUTKEY  Keyword associated with a following call
           to GDSOUT.
AXNUM      The axis number of the axis to be changed.
NAXIS      Size of the axis.
CDELT      Increment in physical units along axis.
CROTA      Rotation angle of axis.
CRPIX      Reference pixel of axis.
CRVAL      Physical reference value at reference pixel.
CTYPE      Axis name.
CUNIT      Physical units of axis.
PMASK      Bit mask denoting which of the six above values
           are defined. The codes are as follows:
           item        bit      add
           CDELT        5        32
           CROTA        4        16
           CRPIX        3         8
           CRVAL        2         4
           CTYPE        1         2
           CUNIT        0         1
           So PMASK = 14 (2 + 4 + 8) means that CTYPE,
           CRPIX and CRVAL are defined. These values must
           always be defined!
```

Subroutine *GDSCAS* changes the subset axis selection and is used by applications which are essentially `CLASS=2` but leave the names of the operation axes intact. VELSMO is an example program which demonstrates the use of *GDSCAS*.

```
Use:       CALL GDSCAS( GDSOUTKEY ,   Input   CHARACTER*(*)
                        SELSUBSET )   Input   INTEGER ARRAY

           GDSOUTKEY    Keyword associated with a following call
                        to GDSOUT.
```

```
SELSUBSET    Array with selected input subset coordinate
             words.
```

Note that before a call to one of the modification routines a call to *GDSASN* must be executed.

4.4 Output set and subsets

GDSOUT is intended for obtaining the set and subsets from the user where your program should put its modified data. *GDSOUT* will perform the same checks as *GDSINP* except that if the output set specified does not exist it will create a new one, and if it does exist it will check whether the input subset does exactly fit in the already existing output set. If it does fit in, *GDSOUT* will ask the user permission to overwrite the old data.

```
Use:          INTEGER GDSOUT( SET      ,  Output     CHARACTER*(*)
                              SUBSETS  ,  Output     INTEGER ARRAY
                              MAXSUBS  ,  Input      INTEGER
                              DEFAULT  ,  Input      INTEGER
                              KEYWORD  ,  Input      CHARACTER*(*)
                              MESSAGE  ,  Input      CHARACTER*(*)
                              SHOWDEV  ,  Input      INTEGER
                              AXPERM   ,  Output     INTEGER ARRAY
                              AXCOUNT  ,  Output     INTEGER ARRAY
                              MAXAXES  )  Input      INTEGER

              GDSOUT    Returns the number of subsets entered by user.
              SET       Name of GDS set selected by user.
              SUBSETS   Subset coordinate words of the subsets selected
                        by the user.
              MAXSUBS   Maximum number of subsets the application
                        allows.
              DEFAULT   Default level as in USERxxx.
              KEYWORD   Keyword for user prompting. Usually 'SETOUT='.
              MESSAGE   Message for the user.
              SHOWDEV   Device as in ANYOUT to which some information
                        about the selected set is send.
              AXPERM    Contains the axes numbers in the order as
                        specified by user.
              AXCOUNT   Contains the number of grids along the axes
                        as specified by user in the order as specified
                        by user.
```

```
MAXAXES   Maximum number of axes the application allows.
```

Note that for `CLASS=1` applications *GDSOUT* automatically checks whether the number of input subsets is equal to the number of output subsets. Independent of the `CLASS` of the application, *GDSOUT* will assume the same subset dimensions as was input to the associated *GDSINP* call.

4.5 Positions in a subset

For obtaining positions in a subset the function *GDSPOS* is used in GIPSY. It decodes the user input into grid positions, which are returned in a double precision array.

```
Use:          INTEGER GDSPOS( POS     ,  Output  DOUBLE PRECISION ARRAY
                              MAXPOS  ,  Input   INTEGER
                              DEFAULT ,  Input   INTEGER
                              KEYWORD ,  Input   CHARACTER*(*)
                              MESSAGE ,  Input   CHARACTER*(*)
                              SET     ,  Input   CHARACTER*(*)
                              SUBSET  )  Input   INTEGER

              GDSPOS    Returns the number of positions entered.
              POS       Array which contains the positions entered
                        by the user. Each position occupies N items
                        in POS, where N is the dimension of the subset.
              MAXPOS    Maximum number of positions to enter. Note that
                        the size of POS must not be smaller than
                        MAXPOS * N, where N is the dimension of the
                        subset.
              DEFAULT   Default code as in USERxxx.
              KEY       Keyword prompts the user to enter positions.
                        If KEY is blank then GDSPOS will prompt with
                        keyword 'POS='.
              MESSAGE   Message for the user. If MESSAGE is blank then
                        GDSPOS will generate its own message.
              SET       Name of set for which a position is wanted.
              SUBSET    Coordinate word of the subset for which
                        positions should be defined.
```

Note that the positions returned by *GDSPOS* should be converted to integers before using it for calculating coordinate words etc. In this case the nearest integer must be taken, i.e. with `NINT`.

4.6 Subset frames

Most GIPSY applications do something on parts of subsets. They allow the user to specify a frame inside (or in some case outside) the subset to work on. This is done with subroutine *GDSBOX* which returns two arrays containing the grid coordinates of the frame.

```
Use:          CALL GDSBOX( BLO     ,    In/Output  INTEGER ARRAY
                           BHI     ,    In/Output  INTEGER ARRAY
                           SET     ,      Input    CHARACTER*(*)
                           SUBSET  ,      Input    INTEGER
                           DEFAULT ,      Input    INTEGER
                           KEYWORD ,    In/Output  CHARACTER*(*)
                           MESSAGE ,      Input    CHARACTER*(*)
                           SHOWDEV ,      Input    INTEGER
                           OPTION  )      Input    INTEGER

              BLO       On input this array may contain the default
                        centre or the default lower left edge of the
                        box (depends on OPTION).
                        On output it contains the user defined lower
                        left grid coordinates of the box.
              BHI       On input this array may contain the default
                        size or the default upper right edge of the
                        box (depends on OPTION).
                        On output it contains the user defined upper
                        right grid coordinates of the box.
              SET       Name of set for which a box should be defined.
              SUBSET    Coordinate word of the subset for which a box
                        should be defined.
              DEFAULT   Default code as in USERxxx.
              KEYWORD   Keyword prompts the user to enter a box. If
                        KEYWORD is blank the GDSBOX will prompt with
                        'BOX='.
              MESS      Message for the user. If MESS is blank then
                        GDSBOX will generate its own message.
              SHOWDEV   Device (as in ANYOUT) to which some info
                        about the chosen box is displayed.
              OPTION    The different options are:
                         1 box may exceed subset size
                         2 default is in BLO
                         4 default is in BHI
```

```
 8 box restricted to size defined in BHI
These codes work additive. So when OPTION = 0
or 1 then the default is entire subset.
```

The arrays containing the grid coordinates can easily be converted to coordinate words with the function *GDSC_FILL*. If an applications program produces output data, it should write BLANKS (better known as undefined values) outside the frame returned by *GDSBOX*. BLANKS are discussed in more detail in the next section.

There is a special routine to keep track automatically of whether a chunk of data just read in lies inside or outside the frame specified with *GDSBOX*. This routine *INITPTR* is used in combination with *GDSI_READ* (or *GDSI_WRITE*). *GDSI_READ* returns the number of pixels just read, which is input to *INITPTR* in order to initialize the logical functions *INSIDEPTR* and *OUTSIDEPTR*. Further input to *INITPTR* are the dimension of the subset, the grid coordinates of the edges of the subsets and the grid coordinates returned from *GDSBOX*. After each call to *GDSI_READ* and *INITPTR* the functions *INSIDEPTR* and *OUTSIDEPTR* return pointers to pixels (in the buffer filled by *GDSI_READ*) which are inside or outside the frame returned by *GDSBOX*. *INSIDEPTR* and *OUTSIDEPTR* return `.FALSE.` if all positions in the readbuffer have been scanned. These routines provide the applications programmer with a simple tool to handle data in subframes (i.e. frames returned from *GDSBOX*) in a different way than the data outside the subframe.

4.7 BLANKs (undefined values)

GIPSY applications programmers should be aware of the fact that the so called undefined value (or BLANK as it is called in FITS terminology) is a fixed value which can only change across different computer operating systems. On some computers any arithmetic operation on such a BLANK might cause a crash of your program. So it is absolutely necessary to check whether a certain pixel value is a BLANK before doing any operation on it. This is done with the function *FBLANK*. This function returns `.TRUE.` if the argument is BLANK, otherwise it will return `.FALSE.` With the subroutine *SETFBLANK* you can set one value to BLANK, and with subroutine *SETNFBLANK* you can set a whole array to BLANK. For the sake of speed it is advisable to compare data values directly with the system defined BLANK instead of using the slow function *FBLANK*. This is done in the following way: declare `REAL BLANK` in your application and set it to BLANK by `CALL SETFBLANK( BLANK )`.

Note that also certain arithmetic operations are illegal like taking the square root of a negative value or dividing by zero. It is clear that your applications program should also check on this prior to the execution of the operation. If the operation is illegal, then the result should be set to BLANK.

4.8 Example

The following example program shows how to manipulate data in GIPSY.

```
      PROGRAM EXAMPLE2
C
C     Demo program which multiplies the input data by two inside and
C     by three outside the frame defined by the user.
C
N     Maximum number of axes in a GDS set
      INTEGER MAXAXES
      PARAMETER (MAXAXES = 10)
N     Maximum number of subsets
      INTEGER MAXSUBS
      PARAMETER (MAXSUBS = 2048)
N     Size of internal data buffer
      INTEGER MAXDATA
      PARAMETER (MAXDATA = 4096)
N     Names of input and output set
      CHARACTER*80 SET1, SET2
N     Dimension of subsets
      INTEGER      SUBDIM
N     Number of subsets
      INTEGER      NSUBS
N     Array containing axis permutation
      INTEGER      AXPERM1(MAXAXES), AXPERM2(MAXAXES)
N     Array containing axis counts
      INTEGER      AXCOUNT1(MAXAXES), AXCOUNT2(MAXAXES)
N     Arrays for input and output subsets
      INTEGER      SUBSET1(MAXSUBS), SUBSET2(MAXSUBS)
N     Arrays containing lower and upper grids of frame
      INTEGER      FLO(MAXAXES), FHI(MAXAXES)
N     Arrays containing lower and upper grids of subframe
      INTEGER      BLO(MAXAXES), BHI(MAXAXES)
N     Coordinate words
      INTEGER      CWLO, CWUP
N     GDS error returns
      INTEGER      GERROR
N     Transfer identifiers
      INTEGER      TID1, TID2
N     Number of pixels just read in buffer
      INTEGER      NREAD
```

```
N     Number of pixels just written to disk
      INTEGER      NWRITE
N     Counter for INITPTR
      INTEGER      NT
N     Various
      INTEGER      N, NS, IP, NP
N     System defined BLANK
      REAL         BLANK
N     Internal data buffer
      REAL         DATR(MAXDATA)
N     Function which gets input set and subsets
      INTEGER      GDSINP
N     Function which gets output set and subsets
      INTEGER      GDSOUT
N     Function which fills coordinate word
      INTEGER      GDSC_FILL
N     Extracts grid from coordinate word
      INTEGER      GDSC_GRID
N     Pointer functions
      LOGICAL      INSIDEPTR, OUTSIDEPTR
C     Executable code:
      CALL INIT
N     Zero GDS error return code
      GERROR = 0
N     No required dimension of subset
      SUBDIM = 0
N     Get input set and subsets from user
      NSUBS = GDSINP( SET1, SUBSET1, MAXSUBS, 0, 'SET=',
     #                'Give input set (and subsets)',
     #                11, AXPERM1, AXCOUNT1, MAXAXES, 1, SUBDIM )
N     Now get subframe of subset
      CALL GDSBOX( BLO, BHI, SET1, SUBSET1(1), 1, ' ', ' ', 11, 0 )
N     Prepare for GDSOUT
      CALL GDSASN( 'SET=', 'SETOUT=', 1 )
N     Get output set and subsets from user
      NSUBS = GDSOUT( SET2, SUBSET2, NSUBS, 0, 'SETOUT=',
     #                'Give output set (and subsets)',
     #                11, AXPERM2, AXCOUNT2, MAXAXES )
N     Fill subset coordinate word with whole range
      CALL GDSC_RANGE( SET1, SUBSET1(1), CWLO, CWUP, GERROR )
N     Now we need the grid coordinates of the subset frame
```

```
      FOR N = 1, SUBDIM
        FLO(N) = GDSC_GRID( SET1, AXPERM1(N), CWLO, GERROR )
        FHI(N) = GDSC_GRID( SET1, AXPERM1(N), CWUP, GERROR )
      CFOR
N     Get system defined BLANK
      CALL SETFBLANK( BLANK )
N     Integrate each subset
      FOR NS = 1, NSUBS
N       Show user what subsets we are working on
        CALL SHOWSUB2( SET1, SUBSET1(NS), AXPERM1,
     #                 SET2, SUBSET2(NS), AXPERM2 )
N       Get coordinate words of lower and upper edge of frame
        CWLO = GDSC_FILL( SET1, SUBSET1(NS), FLO )
N       GDSC_FILL fills in the undefined dimensions from grid array
        CHUP = GDSC_FILL( SET1, SUBSET1(NS), FHI )
N       Reset read transfer identifier
        TID1 = 0
N       Reset write transfer identifier
        TID2 = 0
N       Reset counter for INITPTR
        NT = 0
N       Loop until all data scaled
        REPEAT
N         Read data
          CALL GDSI_READ( SET1, CWLO, CWUP, DATR, MAXDATA, NREAD, TID1 )
N         Initialize INSIDEPTR/OUTSIDEPTR with INITPTR
          CALL INITPTR( FLO, FHI, BLO, BHI, SUBDIM, NREAD, NT )
N         IP is a pointer to pixel inside subframe
N         NP is number of pixels inside subframe starting at IP
          WHILE (INSIDEPTR( IP, NP ))
            FOR N = 1, NP
              IF (BLANK .NE. DATR(N+IP))
              THEN
N               Inside subframe multiply by 2.0
                DATR(N+IP) = 2.0 * DATR(N+IP)
              CIF
            CFOR
          CWHILE
N         IP is a pointer to pixel outside subframe
N         NP is number of pixels outside subframe starting at IP
          WHILE (OUTSIDEPTR( IP, NP ))
```

```
          FOR N = 1, NP
            IF (BLANK .NE. DATR(N+IP))
            THEN
N             Outside subframe multiply by 3.0
              DATR(N+IP) = 3.0 * DATR(N+IP)
            CIF
          CFOR
        CWHILE
N       Write back scaled data
        CALL GDSI_WRITE( SET2, CWLO, CWUP, DATR, NREAD, NWRITE, TID2 )
      UNTIL (TID1 .EQ. 0)
    CFOR
    CALL FINIS
    STOP
    END
```

Chapter 5

GIPSY Coordinate System

5.1 Introduction

Similar to data stored on a FITS tape, the GIPSY coordinate system uses the following descriptors for each each axis to define the coordinate system:

- $NAXIS\%$ holds the number of pixels along an axis.
- $CTYPE\%$ holds the name of the axis. The axis name is uniquely related to the physical coordinates of the pixels along the axis. The axis name is devided into several parts, separated by hyphens. The first part denotes the real axis name, so the physical coordinate system, the second part the type of projection (meaning that there are more ways to go from grid position to physical coordinates). At the moment only projections for spatial axes (like RA and DEC) and spectral axes (FREQ) are defined.
- $CRPIX\%$ holds the location of the reference pixel on the axis in file coordinates. The grid position of the reference pixel is always zero. For this position the physical coordinate is precisely defined in $CRVAL\%$, and therefore we call this position the projection centre.
- $CRVAL\%$ holds the physical coordinate at the reference pixel location in some units. The units are defined in the non-standard FITS descriptor $CUNIT\%$.
- $CUNIT\%$ holds the units of the physical coordinate at the reference pixel location, i.e. the units of $CRVAL\%$. When this descriptor item is not defined, the standard FITS units will be assumed.
- $CDELT\%$ holds the pixel separation in units specified in $CUNIT\%$ along the axis. Note that $CDELT\%$ may be negative.

- $CROTA\%$ holds the rotation angle in degrees of the actual axis from the stated coordinate type in $CTYPE\%$. This descriptor makes only sense for spatial axes. In GIPSY it is assumed that the spatial latitude axis carries the rotation information.

The descriptors above are called the primary axis descriptors, and they relate grid positions to physical coordinates as defined in $CTYPE\%$. The GIPSY coordinate system also recognizes secondary axis descriptors, which relate grid positions to a different type of physical units. At the moment a secondary axis is only used for the FREQ primary axis to relate frequencies to velocities. Secondary descriptors are similar to primary axis descriptor, except that they all start with D instead of C.

In the following section we discuss the supported coordinate types in GIPSY.

5.2 Supported coordinate types

In this section we discuss the implemented coordinate types. The recognized axis names are the following:

- RA and DEC for equatorial sky coordinates.
- GLON and GLAT for galactic sky coordinates.
- ELON and ELAT for ecliptic sky coordinates.
- SLON and SLAT for supergalactic sky coordinates.
- FREQ for spectral frequency coordinates, equidistant in frequency.
- VELO for spectral velocity coordinates, equidistant in velocity.

5.2.1 Sky coordinates

At the moment there are five different sky systems implemented. Each sky system is represented by a number `SKYSYS`. The relation between `SKYSYS` and the sky coordinates is the following:

Implemented Sky Systems			
`SKYSYS`	$CTYPEm$	$CTYPEn$	Meaning
1	RA	DEC	Equatorial (EPOCH 1950.0)
2	GLON	GLAT	Galactic
3	ELON	ELAT	Ecliptic (EPOCH 1950.0)
4	SLON	SLAT	Supergalactic
5	RA	DEC	Equatorial (EPOCH 2000.0)

The routine which transforms from one sky system to another is called *SKYCO*.

```
Use:          INTEGER SKYCO ( XIN    ,  Input    DOUBLE PRECISION
                              YIN    ,  Input    DOUBLE PRECISION
                              SKYIN  ,  Input    INTEGER
                              XOUT   ,  Output   DOUBLE PRECISION
                              YOUT   ,  Output   DOUBLE PRECISION
                              SKYOUT )  Input    INTEGER

              SKYCO     Returns zero if transformation was successful,
                        otherwise a non-zero value is returned.
              XIN       Input longitude in degrees.
              YIN       Input latitude in degrees.
              SKYIN     Sky system of input coordinates
              XOUT      Output X coordinate in degrees.
              YOUT      Output Y coordinate in degrees.
              SKYOUT    Sky system of output coordinates.
```

For sky systems the rotation angle of the sky plane with respect to the axes is stored in *CROTAn*, so it is connected with the latitude axis. The epoch of the coordinate system in stored in the FITS descriptor *EPOCH*.

The different types of sky projection systems are associated with number PROSYS. The following projections are implemented:

Implemented Projection Systems		
PROSYS	Code	Meaning
1	AIT	Aitoff Equal Area projection
2	CYL	Equivalent Cylindrical projection
3	FLT	Flat projection
4	TAN	Gnomonic projection
5	SIN	Orthographic projection
6	ARC	Rectangular projection
7	GLS	Transversal projection
8	NCP	North Celestial Pole projection
9	STG	Stereographic projection
10	MER	Mercator projection

The projection of sky coordinates onto grids is done with function *PROCO*.

```
Use:          INTEGER PROCO( XIN   ,     Input      double precision
                             YIN   ,     Input      double precision
                             XOUT  ,     Output     double precision
```

```
                 YOUT  ,      Output     double precision
                 CRVAL1,      Input      double precision
                 CRVAL2,      Input      double precision
                 CDELT1,      Input      double precision
                 CDELT2,      Input      double precision
                 CROTA2,      Input      double precision
                 PROSYS,      Input      integer
                 MODE  )      Input      integer

PROCO    Returns zero if conversion was successful,
         otherwise a non-zero value is returned.
XIN      Input X coordinate (Longitude in degrees or
         X grids).
YIN      Input Y coordinate (Latitude in degrees or
         Y grids).
XOU      Output X coordinate (X grids or Longitude
         in degrees).
YOU      Output Y coordinates (Y grids or Latitude
         in degrees).
CRVAL1   Projection centre longitude (Longitude in
         degrees).
CRVAL2   Projection centre latitude (Latitude in
         degrees).
CDELT1   X grid separation in degrees.
CDELT2   Y grid separation in degrees.
CROTA2   Rotation angle in degrees.
PROSYS   Type of projection.
MODE     Mode determines what type of input/output
         coordinates are given/wanted.
         MODE       XIN     YIN   XOUT   YOUT
           0        LON     LAT      X      Y
           1          X     LAT    LON      Y
           2        LON       Y      X    LAT
           3          X       Y    LON    LAT
```

Transformation of equatorial coordinates from one epoch to another are done with subroutine *EPOCO*.

```
Use:          CALL EPOCO( RA1    ,   Input    DOUBLE PRECISION
                          DEC1   ,   Input    DOUBLE PRECISION
                          EPOCH1 ,   Input    DOUBLE PRECISION
                          RA2    ,   Output   DOUBLE PRECISION
```

```
                DEC2   ,   Output   DOUBLE PRECISION
                EPOCH2 )   Input    DOUBLE PRECISION

RA1             Input R.A. in degrees at EPOCH1.
DEC1            Input Dec. in degrees at EPOCH1.
EPOCH1          EPOCH1 in years for RA1 and DEC1.
RA2             Output R.A. in degrees at EPOCH2.
DEC2            Output Dec. in degrees at EPOCH2
EPOCH2          EPOCH2 in years for RA2 and DEC2.
```

5.2.2 Spectral coordinates

Spectral coordinates have axis names $FREQ$ or $VELO$. The transformation from frequency to velocity is not straightforward, since the frequencies are usually the observed frequencies. For this reason a secondary axis is introduced in GIPSY. This axis holds the velocity at the reference pixel in descriptor $DRVAL\%$. The velocity of a certain grid position g along the $FREQ$ axis can now be obtained with the following formulae:

$$V_{radio} = DRVAL\% - \frac{gCDELT\%c}{\nu_0}$$

$$V_{optical} = DRVAL\% + (\frac{1}{CRVAL\% + gCDELT\%} - \frac{1}{CRVAL\%})\nu_0 c$$

where c is the speed of light in vacuum and ν_0 the rest frequency of the spectral line. GIPSY uses for the grid separation in velocity the following:

$$DDELT\%_{radio} = -CDELT\%\frac{c}{\nu_o}$$

$$DDELT\%_{optical} \approx -CDELT\%\frac{c}{\nu_o}\frac{\nu_0^2}{CRVAL\%^2}$$

The rest frequency is stored in descriptor $FREQ0$ in units of Hz.

The different velocity systems (indicated by number `VELSYS`) are encoded in the following way:

Velocity Systems		
`VELSYS`	$CTYPEn$	Meaning
1	FREQ-OHEL	Optical definition, w.r.t. SUN
1	FREQ-OLSR	Optical definition, w.r.t. LSR
2	FREQ-RHEL	Radio definition, w.r.t. SUN
2	FREQ-RLSR	Radio definition, w.r.t. LSR

The function *VELPRO* does the velocity - grid transformations.

```
Use:           INTEGER VELPRO( COORD1,    Input    DOUBLE PRECISION
                               COORD2,    Output   DOUBLE PRECISION
                               CRVAL ,    Input    DOUBLE PRECISION
                               CDELT ,    Input    DOUBLE PRECISION
                               DRVAL ,    Input    DOUBLE PRECISION
                               FREQ0 ,    Input    DOUBLE PRECISION
                               VELSYS,    Input    INTEGER
                               DIR   )    Input    INTEGER

               VELPRO    Returns zero if transformation was successful,
                         otherwise a non-zero value is returned.
               COORD1    Input velocity in m/s or grid.
               COORD2    Output grid coordinate or velocity in m/s.
               CRVAL     Observed frequency in Hz at channel reference
                         pixel.
               CDELT     Grid separation in Hz.
               DRVAL     Velocity at reference frequency in m/s.
               FREQ0     Rest frequency in Hz.
               VELSYS    Velocity system of input coordinates.
               DIR       Direction of transform:
                         DIR == 0:        velocity -> grid
                         DIR != 0:        grid -> velocity
```

5.2.3 Other Coordinates

Other coordinate axes which are recognized are listed in the table below.

Other $CTYPE$s	
$CTYPE\%$	Meaning
INVLAM	Inverse Wavelength
LAMBDA	Wavelength
LOGLAM	Logarithmic Wavelength
PARAM	Parameter axis
POLN	Polarisation
TIME	Time

For these axes there are no special routines available yet, so conversion from grid g to physical coordinate p goes in the standard FITS way:

$$p = CRVAL\% + gCDELT\%$$

$$g = \frac{p - CRVAL\%}{CDELT\%}$$

5.3 General Coordinate transformations

The routine which does the general coordinate transformations is called *COTRANS*. This routine converts grids into the physical coordinates as stated by the *CTYPEs* and vice versa. The function *AXUNITS* returns the units of the physical values input/output to/by *COTRANS*.

```
Use:          INTEGER COTRANS ( SET    ,  Input   CHARACTER*(*)
                                SUBSET ,  Input   INTEGER
                                COORD1 ,  Input   DOUBLE ARRAY
                                COORD2 ,  Output  DOUBLE ARRAY
                                DIR    )  Input   INTEGER

              COTRANS    If transformation was successful, zero is
                         returned, otherwize a non zero value is
                         returned.
              SET        Name of data set.
              SUBSET     Subset coordinate word.
              COORD1     Array containing grid coordinates or physical
                         coordinates in substructure defined by SUBSET.
                         The number of coordinates must be equal to
                         the dimension of SUBSET.
              COORD2     Array containing the corresponding physical
                         coordinates or grid coordinates of the
                         coordinates specified in COORD1 and by SUBSET.
                         The units of the physical coordinates are
                         converted to the units as defined by CUNIT
                         and DUNIT in the set header. The dimension of
                         COORD2 must at least be equal to the number
                         of dimensions of SET.
              DIR        Direction of transformation:
                         DIR == 0:  physical coords -> grid coords
                         DIR != 0:  grid coords -> physical coords
```

For determining the axis type, function *AXTYPE* can be used.

```
Use:          INTEGER AXTYPE( CTYPE,      Input      CHARACTER*(*)
                              CUNIT,      Output     CHARACTER*(*)
```

```
                DUNIT,      Output     CHARACTER*(*)
                SKYSYS,     Output     INTEGER
                PROSYS,     Output     INTEGER
                VELSYS )    Output     INTEGER

AXTYPE  Returns:
         0: unknown type of axis.
         1: spatial axis longitude.
         2: spatial axis latitude.
         3: spectral axis frequency.
         4: spectral axis velocity.
         5: spectral axis wavelength.
         6: spectral axis inverse wavelength.
         7: spectral axis log(wavelength).
         8: time axis.
         9: polarisation axis.
        10: parameter axis.
CUNIT   Natural axis units of primary axis.
DUNIT   Natural axis units of secondary axis.
SKYSYS  If AXTYPE equals 1 or 2, the sky system code.
PROSYS  if AXTYPE equals 1 or 2, the sky projection code.
VELSYS  If AXTYPE equals 3 the velocity system code.
```

Note that natural units are usually S.I. Units. In GIPSY the routines involved in coordinate transformations use these units! In case other units are stated in *CUNIT%* or *BUNIT%* the conversion factor can be obtained with function *FACTOR*.

In case the application programmer wants to display the physical coordinates calculated by *COTRANS*, he might want to use routine *ASC_COORD*. This routine returns the physical coordinates in nicely formated character string. A similar routine will be available for formatted coordinates on plots.

```
Use:          INTEGER ASC_COORD( SET    ,  Input  CHARACTER*(*)
                                 COORDS ,  Input  DOUBLE ARRAY
                                 CTYPES ,  Output CHARACTER*(*) ARRAY
                                 VALUES ,  Output CHARACTER*(*) ARRAY
                                 CUNITS )  Output CHARACTER*(*) ARRAY

              ASC_COORD   If successfull zero is returned, else a
                          non-zero value.
              COORDS      Array with physical coordinates as returned
                          by COTRANS.
              CTYPES      Array with left adjusted axis names.
```

`VALUES`	`Array with right adjusted physical values.`
`CUNITS`	`Array with left adjusted units.`

5.4 Hidden Axes

Still to come.

Chapter 6

Tables in GIPSY

A GIPSY table consists of a number of columns which are identified by column names. Each column can have a number of items which are all of the same data type and in the same (physical) units. The items in a column are stored sequentially in a record-like GDS descriptor. The following conventions are used:

- a table name has a maximum length of eight characters.
- a column name has a maximum length of eight characters.
- each column has a column header which holds the following information:
  - type of data stored in column.
  - physical units of data stored in column.
  - comments for the column.

The following data types are supported:

- characters, designated in column header by CHARn, n being the number of characters per item. n cannot be greater then 132.
- integers, designated by INT
- logicals, designated by LOG
- single precision floating point numbers, designated by REAL
- double precision floating point numbers, designated by DBLE

The following routines are available:

GDSA_TABINQ	gives information about a GDS table
GDSA_TABLIS	list all tables present in a GDS descriptor file
GDSA_DELTAB	deletes a table
GDSA_COLINQ	gives information about columns in a table
GDSA_WRCOM	write comments to a GDS table
GDSA_RDCOM	reads comment from a GDS table
GDSA_CRECOL	creates a column in a GDS table
GDSA_DELCOL	deletes a column from a GDS table
GDSA_WCCHAR	writes character*n items to a column
GDSA_WCINT	writes integer items to a column
GDSA_WCLOG	writes logical items to a column
GDSA_WCREAL	writes real items to a column
GDSA_WCDBLE	writes double items to a column
GDSA_RCCHAR	reads character*n items from a column
GDSA_RCINT	reads integer items from a column
GDSA_RCLOG	reads logical items from a column
GDSA_RCREAL	reads real items from a column
GDSA_RCDBLE	reads double items from a column

Chapter 7

Plot Interface

Still to come.

Chapter 8

Display Interface

Still to come.

Chapter 9

Programming in C

9.1 Introduction

This chapter is dedicated to programmers who want to write applications (or service routines) in the C programming language. Since the GIPSY programming environment supports both Fortran and C as programming language, there are some restrictions on what type of arguments may be passed to servant routines. Also the module names have some restrictions since each operating system interfaces C and Fortran differently. These topics, which have to do with the Fortran to C interface, will be discussed in section 2.

The GIPSY programming standard only accepts code written in ANSI C and ANSI Fortran. Almost all operation systems have ANSI Fortran compilers (with extensions which are not portable, so may not be used by GIPSY programmers). However, most operation systems do not have an ANSI C compiler (yet). For most part of the code this will not make a difference, except for function declarations (ANSI C uses function templates), the standard C include files and connected with this the standard C library. These misfortunes are discussed in section 3 of this chapter.

9.2 Function Arguments and Names

Each operating system has different ideas about how to interface Fortran modules with C modules. Therefore a portable Fortran to C (and vice versa) interface was develloped at the Kapteyn Laboratorium. (Programmers who wish to know more about this interface should read document f2cvv.doc available under update.) This interface sets the following restraints on the types of arguments passed to other service routines and on the names of servant routines:

9.2.1 Module names

Almost any routine in the GIPSY library is both C callable and Fortran callable. C programmers append '_c' to the lower case Fortran name (name as stated in corresponding document), whether a routine is written in Fortran or in C. For example, a Fortran application would start with `CALL INIT()`, and a C application with `init_c();`.

9.2.2 Argument Types

The types of arguments which may be passed from one routine to the other, whether the called routines are written in C or Fortran, are:

Argument types	
Fortran	C
`CHARACTER`	`fchar`
`COMPLEX`	`complex`
`DOUBLE PRECISION`	`double`
`INTEGER`	`fint`
`LOGICAL`	`bool`
`REAL`	`float`

The non-C types are all defined in include file `"gipsyc.h"`, so each C programme or function must include this file (after the standard C include files but before the definition files (see next section)).

A `fint` is an integer which has the same number of bytes as the Fortran integer. Since C supports three integer types (`short, int, long`) and since the number of bytes occupied by these types is system dependent, C programmers must use `fint` for portability reasons.

```
/* gipsyc.h

fint will be one of the following, depending on the operating system:
*/
typedef short  fint;
typedef int    fint;
typedef long   fint;
```

A `bool` is also an integer which has the same number of bytes as the Fortran logical.

A `complex` is a struct consisting of two floats, one carrying the real part, the second carrying the imaginary part.

```
/* gipsyc.h
*/
typedef struct { float r; float i; } complex;
```

A `fchar` is a struct equivalent to a Fortran character variable or constant. `fchar` carries the address of the first element of the string and the length of the string. Note that this string is not null-terminated!

```
/* gipsyc.h
*/
typedef struct { char *a; fint l; } fchar;
```

Since Fortran passes all arguments by reference, all C functions must do this also. There is one exception made for type `fchar`, which is passed by value.

9.2.3 Function types

The Fortran routines which correspond with C functions are the following:

Corresponding Fortran/C functions			
in Fortran		in C	
`subroutine`	`ss(..)`	`void`	`aa_c(..)`
`character*(*)`	`cc(..)`	`void`	`cc_c(fchar r,..)`
`complex`	`cx(..)`	`complex`	`cx_c(..)`
`double precision`	`dd(..)`	`double`	`dd_c(..)`
`integer`	`ii(..)`	`fint`	`ii_c(..)`
`logical`	`ll(..)`	`bool`	`ll_c(..)`
`real`	`rr(..)`	`float`	`rr_c(..)`

Note that Fortran character functions become voids in C with an extra argument being the destination of the character function.

9.3 Standard C library

Although a few operating systems support ANSI C, none of them have the standard C include files and libraries. Therefore the eXtended C library was develloped at the Kapteyn Laboratorium to add the most important missing functions, or to replace those which are available on the operation systems but not working according to ANSI C specifications. We have tried to supply the C programmer with the most interesting of those functions. These functions are defined in the ANSI C include files `"stdio.h"`, `"string.h"`, `"ctype.h"`, `"stdlib.h"` and `"math.h"`. Note that these include files reside under update, and that if you want to use them you should use the `""` notation instead of `<>`. The following tables show which ANSI C functions are available for the C programmer.

"stdio.h"			
clearerr	fputc	getchar	rewind
fclose	fputs	gets	scanf
feof	fread	perror	setbuf
ferror	freopen	printf	sprintf
fflush	fscanf	putc	sscanf
fgetc	fseek	putchar	tmpfile
fgets	ftell	puts	tmpnam
fopen	fwrite	remove	ungetc

"ctype.h"			
isalnum	isgraph	isupper	tolower
isalpha	islower	isxdigit	toupper
isascii	isprint	_tolower	
iscntrl	ispunct	_toupper	
isdigit	isspace	toascii	

"string.h"			
memchr	strcat	strlen	strrchr
memcmp	strchr	strncat	strspn
memcpy	strcmp	strncmp	strstr
memmove	strcpy	strncpy	strtok
memset	strcspn	strpbrk	

"stdlib.h"			
abort	bsearch	getenv	rand
abs	calloc	labs	realloc
atof	div	ldiv	srand
atoi	exit	malloc	system
atol	free	qsort	

"math.h"			
acos	cosh	ldexp	sinh
asin	exp	log	sqrt
atan	fabs	log10	tan
atan2	floor	modf	tanh
ceil	fmod	pow	
cos	frexp	sin	

9.4 ANSI C Argument checking

ANSI C supports argument type checking. Portable C (the C which is almost portable and inplemented on most systems) does not have this handy feature. This is bad for portability, since ANSI C function definitions cannot be compiled with portable C compilers. To overcome this programmers are urged to use `#if defined` structures so that it will compile with both types of compilers. The example in the next chapter demonstrates how this can be done.

The Fortran to C interfacing program supplies the C programmer with function templates (both for ANSI C and Portable C, depending on the type of the compiler) which define each function callable from C. This means that C programmers just have the include `"functionname.h"`. If a programmer wants to use the `NELC` function (see next section), he may want to include `"nelc.h"`. This include file will contain `extern fint nelc_c( fchar );` or `extern fint nelc_c();` depending on the compiler.

9.5 Example

In this section we show two examples of two different implementation of the character function `NELC`, one Fortran implementation and one C implementation. First the Fortran (read Sheltran) implementation.

```
C     NELC.SHL
C
C     Integer function NELC returns the number of characters in a string
C     discarding trailing blanks.
C
      INTEGER FUNCTION NELC( STRING )
C
      CHARACTER*(*) STRING
C
      NELC = LEN( STRING )
      WHILE (STRING(NELC:NELC) .EQ. ' ')
         NELC = NELC - 1
         IF (NELC .EQ. 0)
         THEN
            XWHILE
         CIF
      CWHILE
      RETURN
      END
```

Now the C implementation.

```
/* nelc.c

fint function nelc returns the number of characters in a string
discarding trailing blanks.

*/

#include "gipsyc.h"

#if defined(__STDC__)
fint nelc_c( fchar string )   /*     ANSI C function definition */
#else
fint nelc_c( string )         /* Portable C function definition */
fchar string;
#endif
{
   while (string.l-- && string.a[string.l] == ' ');
   return( ++string.l );
}
```

Chapter 10

Documentation of Applications

Still to come.

Chapter 11

Writing Service Routines

11.1 Introduction

This section will explain how to document your service routine and how to create the Fortran to C or C to Fortran interface.

11.2 Documentation

```
Function:     (name of function in uppercase if it is callable from
              C and Fortran, in lower case if its only callable from C)

Purpose:      (a very short description of what the function does)

File:         (name of the source file with extension .c for C and
              .shl for S77 sources. File name should be in lowercase)

Author:       (Your name)

Use:          (Explains how to call the service routine. See examples
              in this document)

Description:  (If you cannot summarize in the purpose line what your
              function does, here is your space for a detailed
              description)

Warnings:     (This item is only for routines which are system
              dependent! Here you put for which systems it works!)
```

```
Related Docs: (If there are any other service routines connected
              with the one described in the document, name them
              here)

Updates:      (Date and name and why or what)
```

11.3 Fortran 2 C Vice Versa

Still to come. Read the f2cvv.doc document.

Appendix A

Descriptor Items

Name	Type	Meaning
BUNIT	CHAR	data units (WU,MJY/SR,...)
POL	CHAR	polarization (I,Q,U,V,XX,...)
OBSTYP	CHAR	type of observation (LINE,CONT)
FSCRA	DBLE	RA fringe stopping center (degrees)
FSCDEC	DBLE	DEC fringe stopping center (degrees)
BANDW	DBLE	total bandwidth of observation
EPOCH	DBLE	epoche of observation (years)
NINTF	INT	number of interferometers used
NPOL	INT	number of polarizations used (1,2,4)
NFREQ	INT	number of frequency points used
REDCODE	CHAR	LINEMAP reduction code
MAPCODE	CHAR	LINEMAP map code
UVGRID	CHAR	convolving function code
BLGRAD	CHAR	baseline grading function
NBLANK	INT	number of undefined values in map
INSTRUME	CHAR	source of data (WSRT,TAURUS,...)
MAPVSN	CHAR	tape volume of map archive
MAPLAB	CHAR	tape label of map archive
APVSN	CHAR	tape volume of Antenna Pattern
APLAB	CHAR	tape label of Antenna Pattern
PCRA	DBLE	pointing center RA (degrees)
PCDEC	DBLE	pointing center DEC (degrees)
MAXBASE	DBLE	maximum baseline (meter)
MINBASE	DBLE	minimum baseline (meter)
BMMIN	DBLE	minor axis (FWHM) of beam (arcsec)
BMMAJ	DBLE	major axis (FWHM) of beam (arcsec)

RESOL	DBLE	resolution of spectral axis
EQUINOX	DBLE	equinox of coordinate system (years) ?
DATE-OBS	CHAR	observation date (DD/MM/YY)
APSET	INT	set number of antenna pattern ?
APSSET	INT	subset number of antenna pattern ?
FILEID	CHAR	identification of file
FREQ0	DBLE	Rest frequency of spectral line (Hz)
UVFREQ	DBLE	reference freq. for UV coords. (MHz)
UVBANDW	DBLE	bandwidth of UV coordinates (MHz)
NOISE	DBLE	noise of map
NORM	DBLE	normalizing factor in FFT
DATAMAX	REAL	maximum value of map
DATAMIN	REAL	minimum value of map
OBSERVER	CHAR	observer name
OBJECT	CHAR	object name
TAPER	CHAR	type of frequency taper (HANNING,...)
GRIDTYPE	CHAR	type of grid
ORIGIN	CHAR	tape writing institute
DATE	CHAR	tape writing date (DD/MM/YY)
OBSTIME	CHAR	observation time (HH:MM:SS)
BMPA	DBLE	pos. angle of major axis of beam (N->E)
HISTORY	HIST	History records
COMMENT	HIST	Comment records