GIPSY User Guide

(Version 2.0 of July 1, 1992)

KAPTEYN ASTRONOMICAL INSTITUTE

SPACE RESEARCH ORGANIZATION of the NETHERLANDS

EXPERTISE CENTRE ASTRONOMICAL IMAGE PROCESSING

Contents

Chapter 1

Introduction

What is GIPSY?

GIPSY is an acronym of Groningen Image Processing SYstem. It is a highly interactive software system for the reduction and display of astronomical data. It was designed originally for the reduction of interferometric data from the Westerbork Synthesys Radio Telescope, but over the last 20 years it has grown to a system capable of handling data from many different instruments (e.g. TAURUS, IRAS etc.).
This guide is intended for astronomers who want to use GIPSY for the reduction of their data. Part I deals mainly with the use of GIPSY. This part is split up in several chapters. Chapter 2 explains how to get started with GIPSY. The subsequent chapters deal with general utilities within GIPSY (chapter 5), and instrument specific reduction programs e.g. radio (chapter 6) or IRAS (chapter 7). The last chapter (chapter 8) of part I deals with the more advanced features of GIPSY such as batch processing, the use of COLA files and even developing your own programs in GIPSY.
Part II deals in much more detail with the basic design and architecture of the GIPSY system. It is intended more as backround reading material for the interested astronomer. In principle part II need not be read to be able to use GIPSY, however it may well help you in speeding up your reduction if you understand the inner structure of GIPSY.
The appendices contain specific system information dealing with e.g. the hardware requirements for running GIPSY, and local definitions.

Conventions

This guide uses the following conventions on notation and terminology.

- Within a command line, uppercase letters and words in the `typewriter` font represent computer output. For example, in the command line

  `INSET=`AURORA

  the keyword `INSET=` is typed by the computer.

- Within a command line, letters and words in the sans serif font represent user input. Thus in the example above the user entered AURORA. Input is case sensitive.

- Outside command lines, the `typewriter` font is used for keywords (see section 9.9) and descriptor items (see section 9.7). Keywords have the equals sign as last character. For example, `DATAMAX` indicates a descriptor item and `BOX=` a keyword.

- Outside command lines, the font sans serif indicates program names, for example, DISK.

- In astronomy, the terms *image* and *map* often are interchangeable. This guide follows the convention to use the term *image* rather than *map*. Where replacement is not possible, you will still find the term *map*.

Part I

Using GIPSY

Chapter 2

Getting started

2.1 Introduction

In this chapter the basic commands needed to run a GIPSY session are described. It is by no means intended to be complete, however it should suffice to get you started. It is intended as a quick-start manual. The first section describes the preparation you need to do before using to use any GIPSY software. Section 2.3 gives all the basic commands for a GIPSY session: startup, running tasks, stopping etc. The next section (section 2.4) explains the use of GIPSY data sets, and the last section describes a real-live GIPSY session as an example.
For all site-specific information you should contact the local system manager.

2.2 Preparation

If you are lucky, somebody will have prepared a proper environment for you, and you can skip this section. If not, you'll have to prepare some things to run GIPSY.
Obviously first you will have to find a computer on which you are allowed to log onto. On this computer you should have at least several tens of Megabytes of disk space for your data files and log files (a typical 512 by 512 pixel image will occupy ~1 Mb of disk space).
If you want to use tape drives, acquaint yourself with their location and (logical) names. If you are going to use a workstation you may want to find some background material on using the X-window environment (see also sections 9.3 and 9.6).
To be able to run GIPSY you may have to setup a GIPSY environment on the computer you are going to work on. Setting up such an environment is described in the document `gipsy_local.dc0` (ask your local system manager).

2.3 The basic commands

To have a GIPSY session proceed as follows:

1. Log in on the computer

This is somewhat system dependent, but usually you need a user name and password. Enter it on the prompt, and you're in business.

On workstations you must set up the X-window environment. This is sometimes done automatically, on most systems you must give a command yourself, ask your local system manager how to do this.

Then go to the subdirectory where you want to have your data files etc. (on unix type e.g.: `cd work` to go to your own subdirectory `work`).

2. Starting a GIPSY session

   To start the session (on a VT-100 terminal or window) type `gipsy` <CR>.[1]

   This will set up the environment as needed for GIPSY and start the master control task HERMES (refchapter:hermes). HERMES is the master program which will check all your inputs before they get passed on to the operating system or to individual tasks. You really do need it!

   HERMES will set up a screen divided into three parts (see figure 2.1).

   - The top part of the screen contains the Common Output Area (COA). In this area a part of the log file, where all tasks put their results, is shown. You can skip back and forth through this log file using the `<tab>` and `<line feed>` keys. The page indicators in the Task Status Area show you which page of the log file is displayed and the page number of the last page of the log file.
   - The middle part of the screen is the Task Status Area (TSA). Here status messages of tasks are shown. Programs that require parameters will inform you of this in the TSA.
   - The bottom part of the screen is the User Command Area (UCA). Everything typed by the user ends up on these two lines (they function as one long line). Thus all commands go through here. Also tasks, when requiring input, will prompt with a keyword in the UCA.

3. Starting a task

   To start a task (also called an application), its name must be entered in the UCA followed by <CR>(e.g. `DISK<CR>` to start disk). The program will then start (indicated by the status "WAITING TO BE RUN", followed by "RUNNING" in the TSA), and subsequently prompt the user for input using keywords.

4. Supplying a task with parameters

   Giving a task parameters is done through keywords. They are asked by a task when the task needs information. A keyword is a character string followed by an equals sign, e.g. `INSET=` Following the equals sign values can be given. Depending on the function of the keyword

[1] Under X-windows you can try `xgipsy` <CR>to start a session using the X-window version of HERMES. Use this option with some caution, XHermes is still under developement.

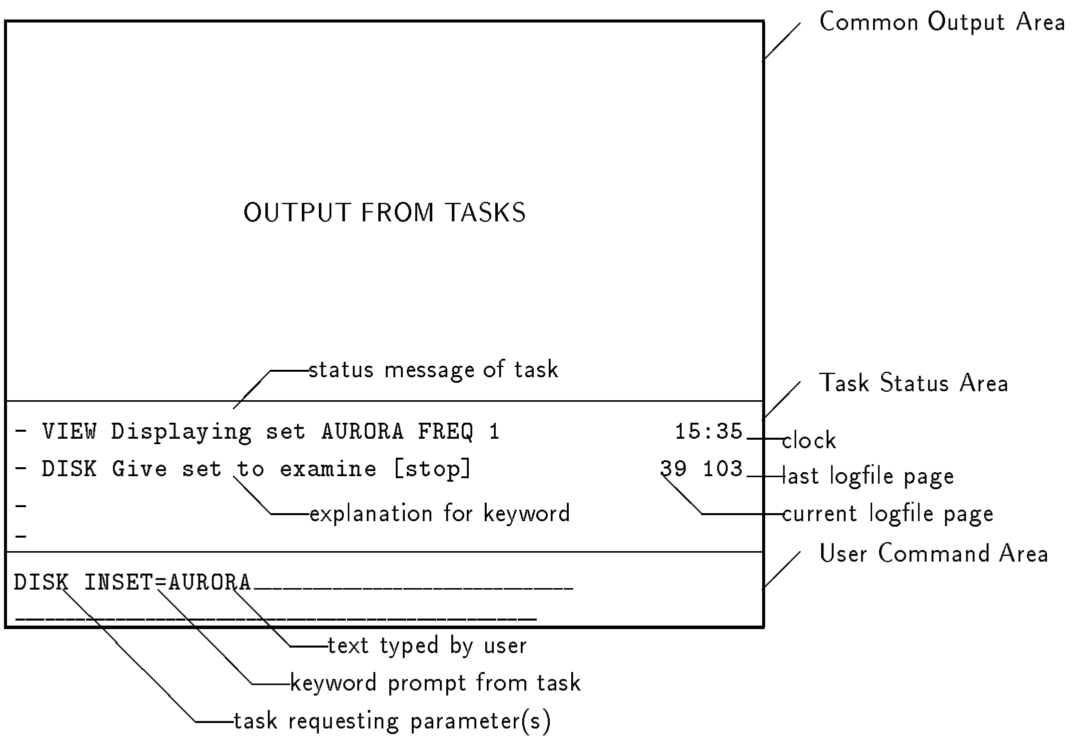

Figure 2.1: The layout of the GIPSY screen as setup by HERMES

values can be floating point numbers, integers, character strings and even expressions. If input given by the user is not as requested by a program (e.g. the program asks for an integer and the user enters text), the keyword will bounce, and you will be prompted again with the same keyword. All keywords asked by a task are listed and explained in the documentation of that task. When a keyword is asked by a program, a one-line message will appear in the TSA.

Many keywords have defaults; the value that is used when the user presses <CR>**without** a value. If a keyword has a default, the default value is given between square brackets in the TSA message.

Besides waiting for a task to prompt you with keywords there are two other ways to specify them. Firstly, when starting up a task you can pre-specify keywords on the same command line (e.g. DISK INSET=AURORA<CR> to start disk for set AURORA). Secondly, when a task is running you can also set a keyword by typing **taskname keyword=value**. When a keyword is pre-specified like this, it will not be asked anymore.

Hidden keywords are keywords which are never prompted, and which always have a sensible default value. You can change their value by pre-specifying them when starting up a task, or by specifying them together with the taskname. Hidden keywords should be considered as fine-tuning parameters; for many general applications you do not need to use them.

5. Aborting a task

   Any task can be aborted by typing its name in the UCA and then typing <Ctrl>C. If there is only one task running, you need not type its name, <Ctrl>C will kill it.

6. Getting help

   When under HERMES you can always get help by typing <Ctrl>H.

   If there is a task name with a keyword in the UCA, the help will be the part of the task documentation relevant for that keyword. If there is only a task name in the UCA, you will get the first page of the task documentation. And if there is nothing in the UCA, you will get help on HERMES.

   You can browse through the documentations using the <tab> and <line feed> (or <Ctrl>I and <Ctrl>J) keys. By typing <Ctrl>H again you will go back to your session.

7. Ending a GIPSY session

   HERMES can be stopped by typing <Ctrl>Q, you are then asked if you to confirm that you really want to stop. You are then returned to at system level.

8. Keeping track of things

   GIPSY keeps a log file. An ASCII version of this log file (named GIPSY.LOG) will be present in the working subdirectory after the session. You can print this file with any standard printing command.

2.4 GIPSY data sets

Data in GIPSY are stored in so called sets. These are n-dimensional arrays of floating point numbers together with a header describing the data. Every axis of a data set has a name, a length in pixels and physical coordinate system associated with it (all specified in the header). Elements of the data set can be identified using an n-dimensional grid coordinate. The user gives such a set whatever name he likes.
A typical radio dataset might be a 3-D "cube" named AURORA, with axes RA, DEC and FREQ. Pixel, or grid, coordinates in such a set can be seen as triplets; (RA,DEC,FREQ).

2.4.1 Sets and subsets

Within sets there are so called subsets. These parts of the n-dimensional set for which one or more of the coordinates are fixed. For example an RA-DEC plane at FREQ grid 32 is a 2-D subset of the cube AURORA above. And, in the most extreme case, the pixel at say (RA,DEC,FREQ) = (2,34,11) is a 0-D subset of AURORA.
When programs ask for an input set, they usually allow you to specify a subset of the data. Specifying one or more subsets is done by specifing the axis name(s) and grids you want for

Input string	function
$*$ hh mm ss	Right Ascention hh mm ss for epoch of set
$*$ dd mm ss	Declination dd mm ss for epoch of set
$*$1950 hh mm ss	Right Ascention hh mm ss for epoch 1950
	.....
	See chapter 3

Table 2.1: Some GIPSY position input formats

each subset. E.g. SET=AURORA FREQ 3, specifies the subset FREQ=3 (i.e. the RA-DEC plane at FREQ grid 3) of the set AURORA. Similarly AURORA FREQ 1 2 3 4 8 (or shorthand AURORA FREQ 1:4 8) specifies 5 subsets: 5 RA-DEC planes in AURORA. Specifying only the axis name, and no grids, selects *all* subsets along that axis (AURORA FREQ selects all RA-DEC planes in AURORA). Again you could specify INSET=AURORA RA DEC FREQ.... guess what happens.
For more details on sets and subsets see chapter 3.

2.4.2 Specifying a position in a (sub)set

Many programs will ask for a position in a (sub)set. This is usually done using the keyword `POS=`. When this keyword is asked programs will tell you what coordinates you must specify and in what ranges (in pixels) you can specify them.
For this keyword you can then enter a grid position. Also you can enter some number corresponding to the physical units along that axis if you postfix that number with the units in which it is. For example in the AURORA FREQ 10 subset you will need to specify RA and DEC for `POS=`. This could be `POS=3 5`(in grids) or `POS=* 14 26 9 * -12 3 5.4`.
For an n-dimensional subsets you will have to specify an n-tuple coordinate; e.g. `POS=3 5 10` specifies the same grid point as above but now starting from the full set AURORA.
Chapter 3 lists all available input formats, pre- and postfixes.

2.4.3 Specifying an area

In many programs you want operations to be performed on specified areas of (sub)sets. More general, operations need to be performed on an n-dimensional section of an m-dimensional set ($n \leq m$). Such a section is called a frame (for more details on frames see section 9.7).
Programs will prompt for a frame usually using the keyword `BOX=`. Similar to the `POS=` keyword, the programs will inform the user of which ranges in which coordinates can be specified. Note that in this definition of frame, it need not be 2-dimensional. To specify the frame yow nou have to give *two* positions one for the lower left, and one for the upper right corner of the frame. For entering these positions the same formats can be used as for the `POS=` keyword (see also Table 2.4.2.

2.5 A sample GIPSY session

At the unix prompt type: gipsy Wait until HERMES is installed. Now you can start GIPSY applications. In this sample GIPSY session, the progams DISK, STAT, and VIEW were used. The results displayed in the log file are:

```
<USER> disk
                                    ***DISK***
 ------------------------------------------------------------------------------
   nr     SETNAME         descr    image    Axis name(s) and size(s)
                           (kb)     (kb)
 ------------------------------------------------------------------------------
    1  cg1517               48    10240    (RA,DEC,VEL) = (256,256,40)
    2  n6946                48             (RA,DEC,DEC) = (512,512,63)
    3  rosimfb1             48      225    (RA,DEC) = (240,240)
    4  total                48     1024    (RA,DEC) = (512,512)
 ------------------------------------------------------------------------------

 Descriptors:     192 kb
 Images     :   11489 kb
 Directory  :   13072 kb
 ------------------------------------------------------------------------------
<USER> DISK INSET=
<STATUS>  DISK   +++ FINISHED +++
<USER> stat
<USER> STAT INSET=rosimfb1
Set rosimfb1 has 2 axes
RA---TAN           from  -119 to   120
DEC--TAN           from  -119 to   120
<USER> STAT BOX01=
BOX range for set rosimfb1 :
RA---TAN           from  -119 to   120
DEC--TAN           from  -119 to   120

subnr  pixels       sum       mean         rms        min         max    blanks
==============================================================================
   1   57447        3.01  5.245E-05       .263      -.628        1.67     153
==============================================================================

<STATUS>  STAT   +++ FINISHED +++

<USER> view box=-20 -20 20 20
<USER> VIEW INSET=rosimfb1
Set rosimfb1 has 2 axes
```

```
RA---TAN           from  -119 to   120
DEC--TAN           from  -119 to   120
BOX range for set rosimfb1 :
RA---TAN           from   -20 to    20
DEC--TAN           from   -20 to    20
<USER> VIEW CLIP=
<STATUS>  VIEW   +++ FINISHED +++
```

The application DISK displays the GIPSY sets that are on disk. The second file in the table has a descriptor part but not an image part. Displayed are the total amount in kilobytes of all descriptors, all images and all files in the current directory. STAT gives an impression of the properties of the data. VIEW displays part of the image. The size of the box is prespecified. VIEW opens a new window where GIDS (Groningen Image Display Server) is started. With your mouse you can click several buttons to start actions like zooming, scaling etc.

2.6 Bug handling in Gipsy

To structure the registration of bugs in GIPSY applications, the program BUG is written. A bug is associated with a subject, which is usually the name of the application. It is possible that you want to report a bug that is already reported by somebody else. To avoid this situation, use the keyword `SUBJECT=` to obtain a list with reported bugs. If you select a subject, you can read all the reports about this subject by giving the report number to `REPORT=`. The keyword is asked in a loop. If you want to continue, press carriage return. With keyword `SUMMARY=` a short description of the bug must be given. The text containing the complaint or description of the bug, can be given interactively or from an input ASCII file. Pressing $< CR >$ at the `INPUT=` prompt will put you in the interactive mode. Your report will be send to the mailbox of the Gipsy manager (i.e. user "Gipsy"). The program BUG is also used by programmers to mail you back if your bug is fixed.

2.7 News

Latest news concerning GIPSY is found with the application NEWS. By default it will display only the news that you did not read before, but `MODE=ALL` displays all news items, including old ones. A selection on items can be made with `MODE=SELECT` which will prompt the user with `ITEMS=`.

Chapter 3

Input syntax

3.1 Introduction

Giving a task parameters is done through keywords. Keywords are asked by a task when the task needs information. A keyword is a case-insensitive character string followed by an equals sign (f.i. `INSET=`). Following the equals sign, values can be given. Depending on the function of the keyword these values can be floating point numbers, integers, character strings and even expressions. Context-sensitive help about keywords is provided at a single keystroke <`Ctrl`>`Q`, (or button-press when working in xHermes). Parameters for a task can be specified in the command that starts the task and thereafter at any moment while the task is active. A task requiring input from the user causes Hermes to find out whether it has already been specified and if not, prompt the user. Before a parameter value is passed to a task, Hermes first checks whether it meets the task's request and if not, Hermes rejects that value and prompts again.

3.2 Input of structures and substructures

A set is an n-dimensional data structure with n different axis names. The contents of the set are floating point numbers stored in a file with the extension "image". The description of the structure is stored in a file with extension "descr". Substructures (subsets) have dimensions in the range 0..n. A subset of dimension 0 is a pixel and a subset of dimension n is the whole data set. If the dimension is less than n, there are multiple subsets possible in a set. This number depends on the number of axes (and their sizes in pixels) that are not part of the subset. The structure is specified by a character string i.e. the name of the set (less than 80 characters including a path) and the subsets by the name of one or more axes outside the subset and numbers indicating pixel positions along those axes. In short, the following format applies:

$$< structure >< sub.axis1 >< lo >:< hi >< sub.axis2 >< lo >:< hi > ...$$

The substructure axes can be found under the header item `CTYPEi` where i=1..n. CTYPE's can be displayed with f.i. the program FIXHED (run: FIXHED ITEM=HEAD).

CTYPE	Meaning
RA	Equatorial, right ascension
DEC	Equatorial, declination
GLON	Galactic longitude
GLAT	Galactic latitude
ELON	Ecliptic longitude
ELAT	Ecliptic latitude
SLON	Supergalactic longitude
SLAT	Supergalactic latitude
FREQ	Frequency
VELO	Velocity axis
LAMBDA	Wavelength
INVLAM	Inverse Wavelength
LOGLAM	Log(wavelength)
TIME	Time
POLN	Polarisation
PARAM	Parameter axis
SAMPL	IRDS sample axis
TICK	IRDS tick axis
SDET	IRDS detector axis
SNIP	IRDS snip axis

Table 3.1: Possible GIPSY axis names

Axis names from table 3.2 can be abbreviated because minimum matching is used. By default the last axis is used and lo and hi indicate pixel positions along the axis following the integer input rules. If an axis name is given but not a number, all subsets along that axis will be generated. Usually sets are asked by the keyword `INSET=` (or variants).

- Examples: the structure AURORA is 3-d and has axes RA,DEC and FREQ with sizes:

```
RA           from   -63 to    64
DEC          from   -63 to    64
FREQ         from     1 to    32
```

`INSET=`AURORA FREQ 10:18 24

`INSET=`AURORA F 10:18 24

`INSET=`AURORA 10:18 24

give 2-d planes (RA,DEC) for values of FREQ 10 to 18 and 24.
Following the integer input rules (section 3.3), you are
also allowed to give weird expressions like:

`INSET=`AURORA F 11:2:−1 12

which generates 2-d planes (RA,DEC) for values of FREQ 11 to 2 and 12

`INSET=`AURORA F

`INSET=`AURORA 1:32

give 2-d planes (RA,DEC) for values of FREQ 1 to 32

`INSET=`AURORA DEC 5:10 FREQ 10

gives 1-d lines (RA) for values of DEC 5 to 10 at FREQ 10

`INSET=`AURORA

gives the whole 3-d cube

`INSET=`AURORA DEC 1

gives 2-d plane (RA,FREQ) for a value of DEC 1

`INSET=`AURORA RA 0 DEC 0 FREQ 1

gives a pixel at (RA, DEC, FREQ) = (0, 0, 1)

Most applications repeat their operations for the specified axes (class 1 programs). Therefore you could call these axes 'repeat' axes. Some applications (class 2) however (f.i. MOMENTS, GAUFIT, MEAN, SUM) need the specification of a so called 'operation' axis. For example MOMENTS requests one operation axis, this is the integration direction. The operation is carried out but not repeated in this direction.

- Examples:

  MOMENTS INSET=AURORA FREQ 2:20 BOX=-63 -63 64 64

  gives a velocity field of the set AURORA for all RA and DEC.

  MEAN INSET=AURORA F BOX=-63 -63 64 64

  gives for all pixels in the RA-DEC plane the average over all frequencies.

  MEAN INSET=AURORA RA -10:10 DEC -10:10 BOX=1 32

  gives an average spectrum (averaged over RA from -10 to 10 and DEC from -10 to 10) for frequencies 1 to 32.

The documentation should state whether class 1 input or class 2 input is necessary. Also the structure of your output is explained in the application documentation.

3.3 Input of floating point or integer numbers

Floating point or integers numbers can be typed as numbers and/or expressions.The numbers/expressions can be typed on one line, separated by blanks and/or a comma. There is a loop-facility implemented, which has the format s:e[:i], where s = start value, e = end value and i = increment value (default: 1), and a repeat-facility, which has the format v::n, where v = value to be repeated and n the repeat-count of v. There is also a list-facility, which is a string of numbers/expressions enclosed between [square brackets]. A list is treated in expressions as a kind of array. The expression is evaluated for each list element (from left to right) separately. Loops, repeats and lists cannot be nested!

- Examples:

```
1 2 3/3  sin(pi)      yields       1.0 2.0 1.0 0.0
log(10)::4            yields       1.0 1.0 1.0 1.0
log(10):log(100):2/4  yields       1.0 1.5 2.0
10**[0 1 5]           yields       1 10 100000
10**[0:3]             yields       1 10 100 1000
```

  Suppose an inclination at keyword INCL= must be given in degrees, but you want the input in axis ratio.

  INCL=DEG(ACOS([0.3 0.5]))

Between square brackets (the list facility) are the arguments 0.3 and 0.5. The expression evaluates the two values for the ACOS and converts these values to degrees. The result is that the values 0.3 and 0.5 are converted to the angles 72.5424 and 60.0 deg.

In numerical expressions the following operators are available:

`+`	addition
`-`	subtraction
`*`	multiplication
`/`	division
`**`	power

The following predefined constants are available:

`PI`	π (3.14159....)
`C`	speed of light (SI)
`H`	Planck (SI)
`K`	Boltzmann (SI)
`G`	gravitation (SI)
`S`	Stefan-Boltzmann (SI)
`M`	mass of sun (SI)
`P`	parsec (SI)
`BLANK`	universal undefined value

The following functions can be used:

abs(x)	absolute value of x	max(x,y)	maximum of x and y
sqrt(x)	square root of x	min(x,y)	minimum of x and y
sin(x)	sine of x	sinc(x)	$\sin(x)/x$ (sinc function)
asin(x)	inverse sine of x	sign(x)	sign of x (-1,0,1)
cos(x)	cosine of x	mod(x,y)	gives remainder of x/y
acos(x)	inverse cosine of x	int(x)	truncates to integer
tan(x)	tangent of x	nint(x)	nearest integer
atan(x)	inverse tan of x	ranu(x,y)	generates uniform noise between
atan2(x,y)	inverse tan (mod 2π)		x and y
	$x = \sin$, $y = \cos$	rang(x,y)	generates gaussian noise with
exp(x)	exponential of x		mean x and dispersion y
ln(x)	natural log of x	ranp(x)	generates poisson noise with mean x
log(x)	log (base 10) of x		
sinh(x)	hyperbolic sine of x	ifeq(x,y,a,b)	returns a if $x = y$, else b
cosh(x)	hyperbolic cosine of x	ifne(x,y,a,b)	returns a if $x \neq y$, else b
tanh(x)	hyperbolic tangent of x	ifgt(x,y,a,b)	returns a if $x > y$, else b
rad(x)	convert x to radians	ifge(x,y,a,b)	returns a if $x \geq y$, else b
deg(x)	convert x to degrees	iflt(x,y,a,b)	returns a if $x < y$, else b
erf(x)	error function of x	ifle(x,y,a,b)	returns a if $x \leq y$, else b
erfc(x)	1-error function	ifblank(x,a,b)	returns a if $x =$ BLANK, else b

3.4 Input of logicals

Logicals are decoded in the following way: YES, JA and TRUE result in a logical which is true, NO, NEE and FALSE give a logical which is false. It is sufficient to give the first letter of the possible affirmative and negative replies. Any other answer will result in a syntax error.

- Examples:
  PLOTGRIDS=N
  OK=YES

3.5 Input of character strings

Normally character strings are case-sensitive, but individual tasks can deviate from this or convert to either uppercase or lowercase. Some programs can pass an array of characters. The characters are then separated by blanks and/or comma's. Sometimes spaces however can be part of the string. The input is called text then and carriage return closes the text.

- Examples:
  `INSTRUME=WSRT` (character string)

  `COMMENT=`Set smoothed on 2-2-92 (text)

3.6 Input of coordinates

You can specify the sizes of the substructures, i.e. the sizes of the subset axes. Usually you are asked to do so with the keyword `BOX=`. Positions can be entered either as pixel positions or as physical coordinates (provided that the relevant header items are correct).
To define a box, there are two general input possibilities:

$$< lower\ position >< upper\ position >$$

$< lower\ position >$ and $< upper\ position >$ indicate the lower- and upper corner of the substructure and have the same dimension as the subset.

$$< centre\ position > D < size >$$

with $< size >$ indicating the rectangular size of a box centered on $< centreposition >$. The D stands for Delta. If only the size is given, the box input routine will prompt you to specify the central position in `CPOS=`
The general format for a position in an n-dimensional subset is (prompted with `POS=` or `POSITION=`):

$$< n - tuple\ value >$$

There are two symbols that denote one position. The first one is **PC**. This stands for Projection Centre and is effectively grid (0,0,...). The second is **AC** which means Axis Centre. If the length of axis i is `NAXISi` and the reference pixel is pixel number `CRPIXi` then the i-th coordinate of AC is given by the expression `NAXISi`/2 − `CRPIXi`. In most cases this will be [(hi+lo+1)/2, ...]. If the specification for `BOX=` is a size only, the application will prompt the user with `CPOS=` complete the input.

- Examples:

  `STAT INSET=AURORA` DEC 0 `BOX=`-30 1 20 32

  gives a RA-FREQ plane with RA from -30 to 20 and FREQ from 1 to 32

  `STAT INSET=AURORA` F 1 `BOX=`0 0 D 7 6

  is a RA-DEC plane with RA from -3 to 3 and dec from -2 to 3 (Central position is 0,0, size of the box is 7x6).

  `STAT INSET=AURORA` F 1 `BOX=`PC D 10 10

  results in a RA-DEC plane with sizes 10x10 centered around the projection centre.

  `COORDS INSET=AURORA` F 1 `POS=`PC

  determines the physical coordinates of the projection centre in set AURORA.

If the correct header items and the correct axes names are entered, you can replace the grid coordinates with physical coordinates. If a coordinate is not followed by a unit, it is assumed to be a grid otherwise it will be converted. A position on a FREQ axis might be given then as:

`POS=`32	just a grid
`POS=`1418.6 MHZ	a frequency
`POS=`300000 M/S	a velocity

If you want to enter physical coordinates in the units corresponding to the axis units, it is possible to prefix the values with 'U'. If the axis has secondary units, these units are used instead of the primary units. For example if `CTYPE3` is 'FREQ' (primary units MHZ) and `DTYPE3` is 'VEL' (secondary units M/S), than:
`POS=`U 300000 is equivalent to `POS=`300000 M/S

For spatial axes there are a number of prefixes:

- Examples:

  `POS=`* 10 12 8 * -67 8 9.6

  RA = 10 hour, 12 min, 8 sec, DEC = -67 deg, 8 min, 9.6 sec., in a 2-d area and in the epoch as found in the header of the set.

DEGREE	ARCSEC	ARCMIN	RADIAN
CIRCLE	METER	ANGSTROM	MICRON
MM	CM	INCH	FOOT
YARD	M	KM	MILE
PC	KPC	MPC	TICK
SECOND	MINUTE	HOUR	DAY
YEAR	HZ	KHZ	MHZ
GHZ	M/S	MM/S	CM/S
KM/S	K	MK	JY
MJY	TAU		

Table 3.2: Units recognized by GIPSY

Input string	function
*	for RA or DEC in resp. HMS and DMS in EPOCH of set.
*1950	for RA or DEC in resp. HMS and DMS in EPOCH 1950.0
*xxxx.x	for RA or DEC in resp. HMS and DMS in EPOCH xxxx.x
G	Galactic longitude or latitude in degrees
E	Ecliptic longitude or latitude in degrees
S	Supergalactic longitude or latitude in degrees

Table 3.3: Possible GIPSY position input formats

`POS=`$*$2000.0 3 14 38.02 $*$2000.0 41 13 54.84

Input of RA 3 h 14 m 38.02 s, DEC 41 d 13 m 54.84 s in epoch 2000.0

`BOX=`G 56.7 G 9.89 D 12 ARCSEC 10 ARCSEC
Input of a box centered at 56.7 degrees galactic longitude and 9.89 degrees latitude with size 12x10 seconds of arc.

Some applications use conversion routines independent of the set header, for example to convert angles given in HMS or DMS to a real value. The documentation (for example: userangle.dc2) will explain the input syntax. Also some applications use a special input routine based on the input routines for reals. This routine allows input of the values -INF and INF (typed as -inf and inf) as minimum and maximum of the image. You can expect to be able to use these values if you encounter the **`RANGE=`** keyword. Again, the application will contain documentation about this.

Chapter 4

Hermes

This chapter describes Hermes, the user interface of the Groningen Image Processing SYstem (GIPSY). It is meant as both an introduction for new users and a reference document for more experienced users.

4.1 What is Hermes?

Hermes is the user interface of GIPSY. It enables users to do multi-tasking in an organised fashion. Users have direct control over the application programs (*tasks*) they run: a running task can be suspended or aborted with a simple command; various settings can be changed while a task is running.
Parameters for a task can be specified at any time; if a task needs information that has not been specified yet, the user is prompted.
Context-sensitive help about tasks is provided at a single keystroke or button-press.
Tasks keep the user informed in two ways: they write in a log file in which the user can page and search and they can provide a one-line status message, which can be frequently updated.
At the moment of this writing three versions of Hermes exist:

- tHermes, which runs on standard character display terminals. this is currently the standard version for interactive work.
- xHermes, an experimental version based on the X Window system. This version is in a number of respects incompatible with other versions of Hermes.
- nHermes, a non-interactive version intended for batch work.

Normally Hermes is not started directly by the user, but rather by a shell script which prepares a number of settings before starting Hermes. Refer to the GIPSY users guide.

4.2 Tasks

Tasks in GIPSY are the application programs which do useful work for the user. They consist of one or more processes (usually one) which communicate with the user through a formalized collection of interface routines. Because of the use of these interface routines, tasks can be run from any version of Hermes.

> Though tasks can be run from all versions of Hermes, not all tasks can be run *successfully* from all versions because a version of Hermes might not implement all interface functions. xHermes is currently the only version which does not implement all functions: it does not support the `DEPUTY` function which is used by tasks to start another task. The most important consequence of this is that the task `COLA` cannot be used under xHermes.
>
> `COLA`-scripts can be started as tasks. For information about the use of `COLA` refer to the document `cola.dc1` and chapter 8.4)

4.3 User input

4.3.1 Introduction

Parameters for a task are passed to it via keywords. A keyword is a character string followed by an equals sign, e.g. `INSET=`. Keyword names are case-insensitive. Parameters for a task can be specified in the command that starts the task and thereafter at any moment while the task is active. A task requiring input from the user causes Hermes to find out whether it has already been specified and if not, prompt the user. Before a parameter value is passed to a task, Hermes first checks whether it meets the task's request and if not, Hermes rejects that value and prompts again.
As the execution of the task proceeds, Hermes builds up a table of the task's keywords and associated values. At task termination this table is saved in order to allow the user to run the task again with (partly) the same parameters.
Three classes of keywords are recognized:

1. *Forced*, which are keywords for which no task default is allowed. The user must respond to the prompt with a correct value.

2. *Defaulted*, which are keywords which have a default value defined by the task. If the keyword has not already been specified then the user will be prompted. If on this occasion the user types only a `<RETURN>`, this signifies that the task default is acceptable and should be used. Any other input will override the default.

3. *Hidden*, which are keywords for which the user is never prompted and which have a default value defined by the task. They can be specified by unprompted input by the user. The documentation of a task must be read to find out if it uses hidden keywords.

   > The user can set a parameter in the task context to change requests for hidden keywords into requests for defaulted keywords. ("Unhide" hidden keywords: `CTRL-S`)
   >
   > Hidden keywords are sometimes used to control looping in a task. Each cycle round the loop, the task requests the value of a hidden keyword. If it has not been specified, the task takes the default and looping continues. If the user defines the keyword then that value is sent to the task which could cause the task to terminate the loop.

4.3.2 Parameter syntax

Basically Hermes is capable of providing tasks with the following types of parameters: integer numbers, real numbers, double precision reals, logicals and character strings. Tasks may however request input in the form of character strings and perform their own decoding to obtain numerical information. One parameter value can contain one or more elements.
Normally character strings are case-sensitive, but individual tasks can deviate from this or convert to either uppercase or lowercase.
In the case of integer or real numbers, the characters typed are converted to the required type. The Fortran syntax for numbers applies. It is also possible to supply an expression, including function references and predefined constants. Lists of numbers can be specified using the '*start*:*end*:*increment*' notation, where the ':*increment*' part is optional and defaults to one. In the following example a list of 8 elements is specified:
`CONT= -1.5:2:0.5`
For logicals the inputs `YES`, `JA` or `TRUE` are affirmative (the first letter is sufficient). Other inputs result in `NO`.

> If a task requests a keyword repeatedly, the user can pre-specify inputs in two ways. The first method is separating the inputs with semicolons, e.g.
>
> `POS= 10 20; 15 25; 30 30`
>
> Please note that a trailing semicolon designates an empty input for which a default is to be taken. The second method consists of specifying a file ("*recall file*") as input to the keyword. This must be a text file with a name like "*name*`.rcl`". Every line in this file is a separate input. Either the whole file or a part of it can be specified:

`<`*name*	use the whole file;
`<`*name n*:*m*	use line numbers n to m;
`<`*name* :*n*	use line numbers 1 to n;
`<`*name n*:	use line numbers n to the end of the file.

> It is possible to specify a recall file in a semicolon-separated list, but within a recall file semicolons cannot be used. Recall files cannot be used recursively, i.e. a recall file can not contain a reference to another recall file.
>
> When the task has "consumed" all pre-specified input on a keyword, the next request for that keyword will cause the user to be prompted in the normal way. Inputs given on repeatedly requested keywords are stored in a semicolon-separated list, so if a task is re-run with the previous set of parameters, *all* inputs are available again.
>
> If unprompted input is given on a keyword of which all pre-specified input has not been consumed yet, the new input will supersede the *complete* previous set of inputs.

Keywords can be prespecified in a text file with a name like "*name*`.def`". This ("*default file*") can contain any number of keywords=values combinations including the use of recall files. User supplied parameters are higher in the parameter hierarchy than the parameters specified in a default file. In tHermes the syntax to use a default file is:

$$< name\ of\ default\ file\ without\ extension >< (program\ name) >$$

example: if myplot.def is a default file for CPLOT start with: MYPLOT(CPLOT)

4.3.3 Numerical input

In numerical expressions operators, constants and functions can be used (See chapter 3)

4.4 Task context

Tasks all run in a *context*. This context consists of a number of user-settable parameters:

- Error level. If a task generates an error at or above the current error level, Hermes will abort the task.
- Message level. If a task generates an error at or above the current error level, Hermes will display the associated error message.
- Output mode. This parameter determines whether test messages from tasks will be displayed and whether messages unnecessary for experienced users will be suppressed.
- Device status. Two parameters indicating whether messages sent to either the screen or the log file actually be written.
- Hide status. This parameter determines whether "hidden" user input requests from the task will cause the user to be prompted anyway.
- Finally there are four parameters which determine what kind of status messages will be logged.

Tasks normally inherit their context from a template context. If a task is started by another task, it inherits the context of that task. Both the template context and the context of any running task can be changed by the user. The way in which this can be done depends on the specific implementation of Hermes. When Hermes is started, the template context contains sensible defaults.

4.5 tHermes

4.5.1 Introduction

tHermes is the version of Hermes which communicates with the user through standard character display terminals, including X11 terminal windows. It organises the terminal screen in a special window-like way. The most important permanent components (windows) of the screen lay-out are:

- User Command Area (UCA). This occupies the lower two lines of the terminal screen and is the focus of user-to-task communication.
- Task Status Area (TSA). This consists of four lines above the UCA which can each contain information reflecting the current status of a task.
- Common Output Area (COA), the remaining top part of the screen which shows a part of the log file. Here user commands are logged and tasks can write text.

Other permanent components are a clock display and numbers which indicate which part of the log file is mapped to the COA.
Components of the screen lay-out which are not permanent are called transient windows. These overlay the permanent screen components or other transient windows. For example the context-sensitive help display overlays (part of) the Common Output Area. Examples of transient windows are the already mentioned help display, a menu window allowing the user to change specific task settings, prompter windows in which for instance a search string can be specified, a Unix shell window, etc.
Many functions of tHermes can be invoked by typing a control key (designated as `CTRL-`*c*, where *c* is a letter), or by typing an escape sequence (the `ESC` character followed by an other character). Important control keys are: `CTRL-Q` to quit from Hermes, `CTRL-C` to abort a task or to break out of a transient window, `CTRL-O` to switch the "keyboard focus" between terminal windows waiting for keyboard input, and `CTRL-L` to repair a damaged screen lay-out.

4.5.2 User Command Area

The User Command Area (UCA) consists of the bottom two lines of the terminal screen where most interaction with the tasks takes place. The user can type task-related commands in the UCA and Hermes can put there task-related prompts. An important aspect of the prompts is that they are treated in exactly the same way as information typed by the user, i.e. the user can erase or modify a prompt supplied by Hermes as if he typed it himself. Prompts in the UCA are always in the form of a legal complete or partial command.
The information in the UCA is used as the context for a number of functions of Hermes. E.g. `CTRL-H` causes the user document of any taskname present in the UCA to be displayed and `CTRL-C` aborts the task if it is active.
The two lines of the UCA are one logical long line which can be edited using the following control keys of which most are EMACS-compatible.

`CTRL-A`	move to start of line
`CTRL-B`	move one position back
`CTRL-D`	forward delete one character
`CTRL-E`	move to end of line
`CTRL-F`	move one position forward
`CTRL-K`	forward delete rest of line
`CTRL-T`	replace complete line with next active task name (see below)
`CTRL-U`	delete complete line
`DEL`	backward delete one character
`space`	replace complete line with next prompt for user input (see below)

Characters `CTRL-T` and `space` are somewhat special. `CTRL-T` erases the UCA and then prompts with the name of an active task. It does this in a circular fashion so if more than one task is active, another taskname is displayed when `CTRL-T` is pressed again.
If the UCA is "free" (a subtle concept explained below), pressing the space bar does a similar thing for tasks waiting for input. This makes it easier for the user to do something else before he supplies the requested input, e.g. run another task which supplies information that helps to give the input.

The UCA is "free" when it is empty, or when it only contains an unmodified prompt supplied by Hermes.

4.5.3 Task Status Area

The Task Status Area (TSA) is the middle section of the terminal screen. It consists of four lines which each can contain status information of an active task. The format of a task status entry is one of the following:

- When a task has been activated, but is not running yet:
  taskname `WAITING TO BE RUN`
- When a task is running:
  taskname `RUNNING`
  or
  taskname status message supplied by servant task
- When a task is prompting for user input:
  taskname prompt message supplied by servant task
- When a task is suspended:
  taskname `PAUSING`
  or
  taskname `PAUSING` *message supplied by servant task*
- When a task is waiting for another task to finish:
  taskname `WAITING FOR` *other taskname*
- When a task has finished processing normally:
  taskname `+++ FINISHED +++`
- When a task has been aborted by the user:
  taskname `USER ABORT`
- When a task has encountered an error fatal to execution:
  taskname error message `-FATAL`
- When a task has crashed (i.e. exited without notification to Hermes):
  taskname `CRASHED`

4.5.4 Common Output Area and Log File

The Common Output Area (COA) on the terminal screen is the remaining top area of the terminal screen not used for the UCA or TSA. It is a viewport on the log file. Hermes treats the log file as a "book" with numbered pages. Normally Hermes shows the current page of the book, that is the page on which output is currently being written.
The COA has two modes of operation: page mode and non-page mode. In non-page mode the page on the screen is always the current page. If necessary, Hermes will flip the page. The

current page number is displayed at the extreme right of the Task Status Area. In page mode the information on the screen will stay there until the user instructs Hermes to change the page. Though the user does not see it, output can still be written to the log file. In page mode the number of the displayed page is shown to the left of the current page number. Page mode can be switched on and off by pressing `CTRL-P`.

The log file can be traversed in backwards direction by pressing the `CTRL-Z` key (or on ANSI keyboards the `PageUP` key). Paging forward is done with the `CTRL-N` key (or ANSI `PageDown`).

Text search in the log file is done by pressing `CTRL-R` (reverse) or `CTRL-S` (forward). Hermes then prompts the user for a search string.

A specific page number can be brought to the screen by typing `ESC G`, whereafter Hermes prompts for a page number. Signed numbers are treated as relative page numbers.

Sections of the log file can be printed by typing the `ESC H` sequence. Hermes prompts then for a range of pages to be printed. The default range is the page currently displayed and the preceding two pages. By default the command to actually do the printing is "`lpr`". This can be changed by typing `ESC P`, whereafter a new print command can be specified.

> The statement that the COA is a viewport on the log file is not quite true. In fact the log file and the COA are different output streams to which tasks can send information selectively. Most tasks however send the same information to both the log file and the COA.

COA command summary

`CTRL-P`	enter or leave page mode
`CTRL-Z`	display previous page and enter page mode
`CTRL-N`	display next page
`CTRL-R`	search forward for text string
`CTRL-S`	search backwards for text string
`ESC G`	go to specified page number
`ESC H`	make hardcopy of specified page numbers
`ESC P`	change hardcopy print command

4.5.5 Running Tasks

Starting tasks and supplying parameters

Tasks can be started with or without specifying parameters. To start a task without parameters, the name of the task must be typed in the UCA, followed by a `RETURN`:

taskname

To specify parameters, any number of keyword-value combinations separated by blanks can be added to the taskname:

taskname $keyword_1$=$values_1$ $keyword_2$=$values_2$ ...

This format is also used to supply parameters to an already running task.

If there is something wrong with the command, an error message is displayed in the lower right corner of the screen.

Tasknames may be preceded by an explicit path. In this case, the task must be present in the specified directory; the task path is not searched.

Tasks can be run again with (partly) the same parameters by prefixing the task start command with an exclamation mark (`!`). Any explicitly specified parameters supersede the corresponding old parameters. For historical reasons, the previous parameters of a task are called the task's *macro*.

Tasks can be run under a different name ("alias"). The command to do this is:

aliasname(*filename*) ...

Aliasname is the taskname under which the task will run and *filename* is the name of the task's executable file, specified in the correct case (lowercase for standard GIPSY tasks).

Aborting, suspending and resuming

Tasks can be aborted by typing `CTRL-C`. They can be suspended and resumed by typing `CTRL-W` or `CTRL-G`. If more than one task is active, the target task's name must be present in the UCA.

Changing the task context

The context of a task can be changed using a pop-up menu that can be activated with the sequence `ESC S`. The following context parameters can be changed with this menu:

- error level
- message level
- output mode: normal, expert or test
- screen output: on or off
- log file output: on or off
- hide "hidden" keywords: on or off

To change the context of a running task, its name must be present in the UCA. If the UCA is empty, the template context will be changed.

Directories and paths

The working directory can be changed by typing `ESC D`. Hermes then prompts the user for a new directory. This directory will then be used for tasks started after the change. Running tasks and Hermes itself are not affected.
The task path can be changed with `ESC T`. Upon the prompt that follows one or more directories or environment variables (with a leading `$`-character) separated by blanks can be specified. Whenever Hermes starts a task without an explicit path, this list is scanned to find the task. If it is not in one of the specified directories, Hermes tries to find the task in `$gip_exe`. The default path is '`.`' (the current working directory).

Inspecting keywords

The current set of keyword-value combinations of a task can be inspected by typing `ESC K` while the name of the task is present in the UCA. This causes (part of) the COA to be overlaid with

a display containing the current user input parameters of the task. If the amount of space on the screen is not sufficient to show all parameters, the user can page through the display just like paging through the log file. `CTRL-N` pages forward; `CTRL-Z` pages backwards. To inspect its keywords, the task need not be active.
Typing `ESC K` again will remove the overlay screen.

Getting help

To obtain help information about a task or about Hermes, `CTRL-H` can be pressed. This causes (part of) the COA to be overlaid with a display containing the user document of the task of which the name is present in the UCA. If there is also a keyword present (e.g. as the result of a prompt), Hermes positions the document at the first occurrence of the keyword. If the amount of space on the screen is not sufficient to show the whole document, the user can page through the display just like paging through the log file. `CTRL-N` pages forward; `CTRL-Z` pages backwards. If the COA is empty, pressing `CTRL-H` will display a summary of Hermes' commands.
Pressing `CTRL-H` again will remove the overlay screen.

4.5.6 Unix shell

Typing `ESC U` causes the COA to be overlaid by an Unix shell window or, if the Unix shell is already present, causes the window to be popped down. The shell used is `/bin/csh`. Because this shell is not connected to a real terminal, functionality is somewhat limited: e.g. a screen editor would not work. Also some commands behave differently, e.g. `ls` produces a one-column list. (A multi-column list can be obtained with `ls -C`.)
By default information displayed in the Unix shell window is not logged. Logging to the COA and log file can be enabled (and disabled) by typing `ESC L`.
The Unix shell can be completely terminated by typing `exit`.

4.6 xHermes

xHermes (X Window Hermes) is in a number of respects incompatible with tHermes. E.g. screen log files have different formats, task keywords (for the automatic "macro") are saved in different ways. The current version of xhermes should be considered as the result of an exercise in programming with the X Toolkit and the Athena widgets and as such should eventually be discarded when a new xhermes has been developed. For this reason no major improvements of this version will be carried out. Suggestions are however welcome; they will be useful when the new xhermes will be developed.

4.7 nHermes

4.7.1 Introduction

nHermes (non-interactive Hermes) is the GIPSY master control program for a non interactive environment. xHermes is based on the interactive tHermes. The user commands will be considered equivalent as if they were typed in the User Command Area of the tHermes. Default values for unspecified keywords will be taken if available. If there is no default available, an error message will be generated in a log file and nHermes will stop with the current user command. When a user command is done, nHermes starts the next user command if given by the user. If a time out option is set, a time out will occur after the user supplied number of minutes and nHermes stops with the user command it is dealing with at that time. The time out works separately for each user command. All output will be kept in a log file.

4.7.2 Syntax

The syntax for starting nHermes is:

$$\text{nhermes } [-l < log\ file\ name >][-t < minutes >]< user\ command >$$

The options may appear in any order and may be intermixed with the user commands. The -l and -t flags are optional.
The log file will always have the extention .LOG and it contains a history of all operations done and the information generated by servant processes. The information in the log file which is generated by servant processes depends on the settings of those processes. If you don't give the extention in uppercase or not at all, it will be added to $< log\ file\ name >$. The default log file name is GIPSY.LOG.
The time out option provides termination of the user command when it is stucked in an endless loop. $< user\ command >$ can be a cola script or a name of a servant task with their parameters.

> COLA is a small (but powerful) high-level control language for use with Hermes. Using COLA you can write a program to execute a sequence of commands.

The name of the servant task or the cola script with the parameters should be suitably quoted or escaped to keep it together (e.g. between double quotes). If the given servant task or cola script with their parameters are not suitably quoted or escaped to keep it together, nHermes handles them as if they were separate user commands.
A cola script file must have the extention .col as specified in the documentation of the application program COLA. You may give the name of the cola script with or without the extention .col to nHermes.

- Example:

  nhermes –lMylog "FLUX, INSET=cg1517 v "

  Let 'nhermes' start task FLUX on all subsets of set cg1517 and put results in file Mylog.LOG.

CONTROL KEY DEFINITIONS	
HELP	
CTRL-H	- provide help display. If a task name is present in the user command area, the dc1 document of that task is displayed. If there is also a keyword present, then the display is positioned at the first occurrence of the keyword in the document. When CTRL-H is pressed for a second time, the help display will disappear.
CTRL-N	- page forward in the help display.
CTRL-Z	- page backwards in the help display.
CTRL-R	- reverse search for a text string in the document.
CTRL-S	- forward search for a text string in the document.
SESSION CONTROL	
CTRL-L	- refresh the terminal screen.
CTRL-O	- circulates between terminal windows waiting for keyboard input.
CTRL-Q	- prompts for user's approval to quit. 'Y' terminates the GIPSY session. If any tasks are still active, THermes waits until they are finished. Any other character causes the session to be continued.
TASK CONTROL	
CTRL-C	- abort a task; the task name must be present in user command area (UCA).
CTRL-W	- suspend or resume a task (name must be present in the UCA).
Esc S	- change running parameters, e.g. output mode. If a task name is present in the UCA, then this task is affected, otherwise the default settings for new tasks are changed.
Esc K	- display the current user input parameters of the task of which the name is present in the UCA. The task need not be active. Pressing Esc K again removes this display.
Esc D	- change working directory.
Esc T	- change 'path' for tasks. A path consists of one or more directory specifications, separated by blanks.

Table 4.1: HERMES commands

COMMON OUTPUT AREA	
CTRL-N	- page forward in the common output area (COA).
CTRL-Z	- page backwards in the COA.
CTRL-R	- search backwards for text in the COA.
CTRL-S	- search forward for text in the COA.
CTRL-P	- switch page mode on or off.
Esc G	- go to absolute or relative (+/-) page number.
Esc H	- make hardcopy of screen pages to printer (default: current page)
Esc P	- change printer command for Esc H (default: "lpr")
COMMAND LINE EDITING	
CTRL-A	- move to start of user command area (UCA).
CTRL-B	- move one position backwards.
CTRL-D	- forward delete one character.
CTRL-E	- move to end of UCA.
CTRL-F	- move one position forward.
CTRL-K	- forward delete rest of command.
CTRL-U	- clear UCA.
DEL	- backward delete one character.
CTRL-T	- replace UCA contents with the next active task name.
Space	- if the UCA is 'free', replace it with the next prompt for user input.
UNIX SHELL	
ESC U	- pop up and down UNIX shell (terminate with `exit`.
ESC L	- logging on/off to COA and log file.
MISCELLANEOUS	
ANSI escape sequences recognized by Hermes:	
Key	- Equivalent
left arrow	- CTRL-B
right arrow	- CTRL-F
up arrow	- CTRL-T
page up	- CTRL-Z
page down	- CTRL-N

Table 4.2: HERMES commands (cont.)

Chapter 5

General GIPSY programs

5.1 Introduction

This chapter describes the most often used general utility programs available within GIPSY. For each program a short description is given of the most important keywords and their use in standard cases. More information can be found in the standard GIPSY documentation files. You can print these files using standard print commands. You can always get help from within GIPSY using the HERMES help command (<Ctrl>H).
The programs are treated more or less in the order in which they are needed; getting data (i.e. listing and reading tapes, making maps), finding your way on disk (listing disk contents, looking at headers), basic set manipulation, plotting and printing etc.etc.

5.2 Getting data into GIPSY

LFITS

The program LFITS lists FITS (Flexible Image Transport System) files on magnetic tape or disk and gives one-line descriptions of their contents. The program first asks to specify the input device with `TAPE=`. Here the name of a tape unit can be entered. For example,

`TAPE=MAG0`

Also, the name of a subdirectory on disk can be given. For example,

`TAPE=../Gipsy/FTSfiles`

where ../Gipsy/FTSfiles is the name of a subdirectory owned by the user. In this subdirectory all files with the names `filexxxxxx.mt` (where `x` can be any digit) are assumed to be FITS files. Next, LFITS selects the numbers of the files to list with `FILES=`. As a default, all FITS files are listed.

RFITS

RFITS transfers FITS files from tape to GIPSY data sets on magnetic disk. Like LFITS it first

asks for a tape unit from which to transfer the data, with INTAPE=.
Subsequently filenumbers which are to be loaded onto disk can be specified using INFILES=. The number of the first file found on tape is zero. Thus when the tape has been wound forward to e.g. the middle of its total length, RFITS will continue forward until it finds a begin of file, and then assume that that file is number zero. The numbers of all following files are relative to this one. To transfer more than one FITS file to one GIPSY set, supply INFILES= with a series of file numbers. For example,

INFILES=0 3 4

or

INFILES=5:10

To allow more than one transfer, RFITS repeats the keyword INFILES= until a carriage return is given. Since RFITS reads a tape in forward direction, the input numbers must be in ascending order. Each input to INFILES= is followed by a series of keywords to specify the output set.
To select the output set name, RFITS prompts with OUTSET=. This keyword allows an existing set name as input.
Subsequently, the program asks (hidden, unless AUTO=Y) information on the axes, repeating

a series of keywords for each axis. The last character of these keywords always indicates the number of the questioned axis (e.g. NAXIS2 contains the number of pixels on the second axis). In the following, this character is represented as an asterisk.

> In automatice mode RFITS will do its best to comprehend the FITS structure on tape. FITS items for which the program has some reasonable defaults will not be asked. In non-automatic mode (AUTO=N) all items will be asked.

CTYPE*= requests the axis name. If an existing set is supplied as output set, RFITS tries to provide by default the name of the corresponding axis in the set. For a not yet existing set, RFITS can provide by default the axis name in the FITS file, if this is a correct GIPSY axis name. Axes having an extension of one can be abandoned, by giving SKIP, or hidden, by giving HIDE. The only remainder of a hidden axis in the output set is a descriptor item telling that it is a hidden axis. SKIP and HIDE can appear as default values too.
If the units of the axis are not available in the FITS header, RFITS requests them with the keyword CUNIT*=. For example,

CUNIT1=m/s

Then the program asks, with the keyword CRVAL*=, to enter the value in physical units of the reference pixel on the axis. The reference pixel is the pixel with value zero. For example, for a FREQ axis,

CRVAL3=2000

Next, the distance in physical units between pixels on the axis has to be set using CDELT*=.

To specify the range in grids along the axis the keyword LIMITS*= can be used. Note that sets, once they are created can only be extended along their last axis, so make sure the area specified with LIMITS*= is large enough for your purposes.
The first grid position on the axis is set with GRID*=. The values given to GRID*= and LIMITS*= determine the reference pixels in the output set. E.g., if

LIMITS1=15 GRID1=−7

then the reference pixel for the first axis is 8.
If a data set has a frequency axis, the frequencies in principle can be converted to velocities if the relevant rest freqeuncy is known. RFITS requests the rest frequency of observation in Hz, with FREQ0= if it is not found in the FITS header. For example,

FREQ0=1400

If the FITS file was written by the old VMS GIPSY, the velocity of the first grid on the velocity axis is asked for with VEL=, e.g. VEL=20 .

CREATE

The program CREATE creates a set and fills it with user defined values. The program starts with prompting for a set name, with the keyword INSET=. If the name of an already existing set is entered, the program returns with the keyword DELETE=. Answering with YES results in deletion of the existing set. Giving a return lets CREATE prompt again with INSET=.
Next, the program requests the instrument of observation, with INSTRUME=. For example,

INSTRUME=VLA

CREATE asks then to specify the axes, prompting with a series of keywords for each axis (like in RFITS). The last character of these keywords indicates the axis number. Below, an asterisk represents this character.
First, an axis name has to be assigned to CTYPE*=. E.g.,

CTYPE1=RA

The next keyword NAXIS*= selects the size of the axis in grids. For example,

NAXIS1=15

Then, the reference pixel of the axis (having physical coordinate CRVAL*=) is asked for, with CRPIX*=. For example,

CRPIX1=8

Subsequently, supply the physical units of the axis to CUNIT*=. For example,

CUNIT1=m/s

and next give the value in physical units at the reference pixel with CRVAL*=. E.g., CRVAL3=2000

The keyword CDELT*= requests the increment in physical units per pixel. For example,

CDELT1=1

Only for a DEC axis, CREATE asks to specify the rotation angle of the axis in degrees, with CROTA=. Its default is zero.
For a FREQ axis, CREATE prompts with the keywords DUNIT*=, DRVAL*=, and FREQ0=. To specify the secondary units etc. of a FREQ axis. Answer FREQ0= with the rest frequency in Hz. This series of keywords is repeated until <CR>is given to CTYPE*=.
After entering all information on the axes, fill the set with values by assigning a mathematical expression to FUNCTION=. Parameters in the expression are the axes names assigned to CTYPE*=. For all functions, constants, and operators allowed in the expression, see the document create.dc1. A sample expression is

FUNCTION=RA + DEC/100

5.3 What is on disk?

DISK

The program DISK gives a list of the sets in the current directory. Furthermore, some information on particular sets can be obtained.
A set consists of astronomical data and their header information. Its structure compares to an n-dimensional array. A subset of a set is an m-dimensional part of it, m being smaller than or equal to n. A frame is a part of a subset, but it has for one or more of its dimensions not the full range of values of that dimension.
To start the program, enter its name; DISK<CR>. Then, DISK displays an overview of the sets in the current directory as shown in Table 5.1.
Note that DISK truncates names more than fifteen characters long. For example, AVERYVERY-LONGNAME becomes AVERYVERYLONGNA...
After giving the listing, DISK prompts with INSET= for the name of a set on which you want more detailed information. For example,

INSET=VELRES

For this set DISK displays more detailed information.
DISK repeats prompting with INSET=. A carriage return stops the program.
This program has a hidden keyword PRINTER=, allowing for sending its output to a printer. To use this option, start up DISK as follows.
DISK PRINTER=0
DISK shows a menu of available printers on the screen (see Table 5.2).
Then, a choice can be made. For example,

PRINTER=3

```
                             ***DISK***
-----------------------------------------------------------------------------
  nr     SETNAME          descr    image    Axis name(s) and size(s)
                           (kb)     (kb)
-----------------------------------------------------------------------------
   1  3DSET                 48       10     (RA,DEC) = (60,40)
   2  4DIMSET               48       40     (FREQ,RA,DEC,PARAM) = (10,10,10,10)
   3  ANTENNA               48       40     (RA,DEC) = (100,100)
   4  AS4                   48        1     (RA,DEC,FREQ,NEWPAR) = (5,5,2,3)
   5  AURORAMOMENTS         48       10     (RA,DEC,PARAM) = (20,20,6)
   6  BLANKSET              48       19     (RA,DEC,VEL) = (40,40,3)
   7  ASUPERLONGNAME...     48        1     (RA,DEC) = (10,10)
   8  FITDAT                48        1     (RA) = (16)
   9  GAUSS                 48       40     (RA,DEC,FREQ) = (100,100,1)
  10  GOODLOOK              48      767     (RA,DEC) = (415,473)
  11  HISTOGSET             48        1     (PARAM,PARAM) = (10,10)
  12  KGB2                  48       11     (DEC,PARAM) = (1014,3)
  13  VELRES                48      108     (RA,DEC) = (131,211)
  14  n1260s60              48      896     (RA,DEC,FREQ) = (128,128,14)
-----------------------------------------------------------------------------

Descriptors:     672 kb
Images     :    1938 kb
Directory  :    2614 kb
-----------------------------------------------------------------------------
```

Table 5.1: Sample output of DISK

HEADER

The program HEADER displays header information on a set and its subsets. The program prompts for an input set (and subsets) with the keyword `INSET=`. If only a set is given, HEADER returns header information on the set as a whole. If one or more subsets are specified, only header information pertaining to these subsets is provided. Information not available at subset level is searched for at higher levels. For example, typing

`INSET=VELRES`

will produce a header listing as given in Table 5.3.

To obtain a hard copy of the output you can use the hidden keyword `PRINTER=` as described under DISK (see also Table 5.2).

```
==============================PRINTERS==============================
 nr       name              cols  rows     comment
====================================================================
  1   textprinter             77    64  Laserprinter in room ZG92A
  2   lineprinter            132    60  Printer in room ZG170
====================================================================
```

Table 5.2: PRINTER= menu

FIXHED

The program FIXHED adds, changes, and deletes (sub)set header items. Below, these options are described separately.
First, assign an input set (and subset(s)) to INSET=. Then, FIXHED prompts for the name of the header item to work on, with the keyword ITEM=. There are two reserved names, **HEAD** and **LIST**. To list the header items present in the specified set or (sub)set(s), give

ITEM=HEAD

FIXHED displays the items, in the format: item name, value, and, if available, FITS style comment. A typical section of FIXHED output is shown in Table 5.4.
The descriptor items in GIPSY programs are partly FITS keywords, partly keywords defined by GIPSY programmers and users themself. A list of the most used descriptor items can be obtained by typing

ITEM=LIST

FIXHED displays then the items, in the format: item name, meaning, and, optionally, possible values. For example,
`BUNIT    : data units (WU, MJY/SR, ...)`
where BUNIT is the item name, data units its meaning, and WU and MJY/SR are examples of possible values.
FIXHED allows for repeated input to the keyword ITEM=. Pressing <CR> stops the program. See now the option of your choice.

Add

To add an item to the specified (sub)set(s), assign the item name to the keyword ITEM=. For example,

ITEM=BUNIT

If necessary, consult the list of items used by GIPSY to find the name of the intended item (See the option ITEM= LIST explained above). It is recommended to use item names already available

```
*****************************************************************************
Set: VELRES  top level                                             26-FEB-1992
Observer: BROEILS          Obs. date:   / /         Object: N1560
Obs. type: POLN            Instrument: WSRT         Polarization: XX
Map center:                (Epoch: 1950.0)
RA       :           66.75 DEGREE =  4 h 27 m 0.00  s
DEC      :           71.80 DEGREE = 71 d 47 m 53.99 s
Axis length and range:
RA       :      131   [-70, 60]
DEC      :      211   [-120, 90]
Grid spacing:
RA       : -0.001390 DEGREE = -5.004000 arcsec
DEC      : +0.001390 DEGREE = +5.004000 arcsec
Data range = [-31.7353,37.884 ] (km/s)       Number of blanks: 0
*****************************************************************************
Fringe stopping center  RA  =    66.75 =  4 h 27 m 0.00  s
                        DEC =    71.80 = 71 d 47 m 54.00 s
Pointing center         RA  =    66.75 =  4 h 27 m 0.00  s
                        DEC =    71.80 = 71 d 47 m 54.00 s

Rest frequency     ****                    Interferometers  40
Total bandwidth          2.50000 MHZ       Polarizations    2
Channel separation ****                    Frequency points 128
Center velocity    ****                    Taper            HANNING

Minimum baseline   36.0 meter              Tape volume      P92758
Maximum baseline   2735.8 meter            Tape label             39

Beam (FWHM) = **** x ****
Position angle major axis (N->E): -0.74 degrees
Antenna pattern: ****

HISTORY/COMMENT
===============
....
*****************************************************************************
```

Table 5.3: HEADER output

```
HEADER at top level
==============================
APLAB   = '        0 '         / tape label of Antenna Pattern
APVSN   = '          '         / tape volume of Antenna Pattern
BANDW   =          2.500000E+00 / total bandwidth of observation
BLGRAD  = 'B        '           / baseline grading function
BMPA    =         -7.441014E-01 / pos. angle of major axis of beam (N->E)
BUNIT   = 'km/s     '           / Units of data
CDELT1  = -.13900000000000D-02
   .
   .
   .
UVGRID  = 'P        '           / convolving function code
VEL     =          6.658606E+01 / velocity (km/s) of V=0
ZGRID   =         -7.441014E-01 / grid unit of subset

HISTORY/COMMENT
===============
  .......
```

Table 5.4: FIXHED, ITEM=HEAD output

in GIPSY; a program needing the header information on the maximum value in a set searches for the header item DATAMAX; items with the same content that have other names are ignored. In case a new item name is entered, FIXHED requests that you select the type, with the keyword TYPE=. The types available are

- I(nteger), e.g., to store the number of interferometers used in an observation
- R(eal), e.g., to store the maximum of a set
- D(ouble), e.g., to store the total bandwidth of an observation
- L(ogical), seldom used
- C(haracter), that is, a character string, e.g., to store the date of observation
- T(ext), that is, one or more character strings, e.g., to store the set and subset used for an antenna pattern
- H(istory), text type, reserved for the items COMMENT and HISTORY

If the fictitious item name ODDNAME was introduced, input could be, for example,

TYPE=Double

or

TYPE=T

meaning text type.
Furthermore, a value is associated with the item, by specifying the keyword VALUE=.
The keyword COMMENT= can be used to enter a FITS style comment. For example, if the item to be added is NINTF, input can be
COMMENT=number of interferometers used
A carriage return results in no comment. FIXHED does not prompt with COMMENT for items of text or history type.
When FIXHED is working on a series of subsets, it prompts with the keywords VALUE= and COMMENT= for each individual subset. To provide remaining subsets with the same input to these keywords as the current subset, use the hidden keyword ALL=. For example,
VALUE=15 ALL=Y
To stop the program from adding the descriptor item to remaining subsets, give
VALUE=<CR>

Change

To change a descriptor item, enter the item name. For example,
ITEM=DATAMAX
FIXHED then requests that you change the value of this item, with the keyword VALUE=. A carriage return leaves the value unchanged. For the items HISTORY and COMMENT, the program first displays the old value, line by line. See the following example of an item of HISTORY type.

```
subset nr.1
1: DUMMY SET FOR TESTING 30/4/1991
2: MINOR CHANGES 10/5/1991
```

The keyword LINE= asks for the number of a line to be changed. Then, VALUE= has to be specified for the entered line number. FIXHED keeps prompting with LINE until a carriage return is given. For example,

LINE=1

VALUE=DUMMY SET FOR TESTING PROGRAM HEADER 30/4/1991

LINE=3

VALUE=DATAMIN ADDED ON FREQ SUBSETS 12/5/1991

LINE=<CR>

Subsequently, FIXHED allows for changing the FITS style comment. The program displays the old comment and prompts for a new one, with the keyword COMMENT=.
When FIXHED works on a series of subsets, it requests input to the keywords VALUE= and COMMENT= for each individual subset.

Delete

To delete a descriptor item, its name has to be entered. Also, the option D(elete) has to be assigned to the hidden keyword `MODE=`. For example,
`ITEM=`DATAMIN MODE=D
When the keyword `INSET=` was supplied with a set name only, then the program asks if the specified header item has to be deleted on all lower levels too, with `ALL=`.
Next, FIXHED asks for a confirmation of the intended deletion, with the keyword `OK=`. When a series of subsets was assigned to `INSET=`, the program prompts with `OK=` for each individual subset. To delete the header item for all remaining subsets, type
`OK=`Y ALL=Y
To leave the item attached to all remaining subsets, answer `OK=` with a carriage return.

DELETE

The program DELETE deletes sets in the current directory. DELETE first requests, with the keyword `INSET=`, to specify the name(s) of one or more sets to delete. For example,
`INSET=`GAUSS LOOK4
To check if it indeed has to delete these sets, DELETE prompts for each individual set with the keyword `OK=`. The default is NO. The name of the set prompted for is displayed in the status area. E.g.,
`Ok to delete GAUSS ? [Y/N]`
Then, DELETE repeats asking, with the keyword `INSET=`, for sets to delete. A carriage return stops the program.

5.4 Displaying data

GIDS (Groningen Image Display Server) is the application you need to display your 2-dimensional (sub)sets. It can be started with INIDISPLAY or with VIEW (which needs a (sub)set). Using GIDS is very simple. With your mouse (and one of its buttons) you select actions like zooming, scaling in the window with the menu bars. Move the mouse pointer in one of the menu boxes and click with one of the mouse buttons. An action is started or a sub menu appears. In the upper left corner of the GIDS window there are 7 boxes containing relevant information. Example:

1	`AURORA`	Name of the image.
2	`5168.24 KM/S`	The physical value of the current subset.
3	`REC= 7, 4, 3`	Seven images of current size can be stored, you have stored 4, and you are looking at number 3.
4	`MASK= 0 ON= 15`	Future use.
5	`VAL= -1.786E-2`	The value of the pixel you are pointing at.
6	`POS= -2, 5`	The position in grids of the pixel you are pointing at.
7	...	A text box in which you can enter numbers of recorded images for looping etc.

Some of the GIDS options are not yet implemented (for example Region) others need a little extra information. For interactive work the Cursor option is handy. If f.i. you are displaying an image and want to run STAT, the keyword `BOX=` can get its values manually or with your mouse buttons.

Button	Action
left	enter grid coordinates of mouse position in the UCA of tHermes.
middle	interactive definition of a box. The size of the shape is determined after this button has been released.
right	return the coordinates of a previously defined box in the UCA of tHermes.

Table 5.5: Mouse actions using the Cursor option in GIDS

After clicking Etcetera and Recording there are options for recording a limited number of (sub)sets, viewing those images, blinking two images, looping over a number of images (movie, see VIEW) and splitting an image.

VIEW

The program VIEW displays two-dimensional (sub)sets. First, VIEW requests that you specify the set (and subset(s)) it has to display, with `INSET=`.

Then, so-called "clip levels" can be set, a lower and an upper clip level, by specifying `CLIP=`. Only values falling within the range specified by these levels are displayed. For example,

`CLIP=`0 15

displays only pixel values between zero and fifteen.

If no display window is available VIEW will create one, asking the user specify a position for that window on the screen. Next VIEW displays the first image (subset) . If more than one subset was assigned to `INSET=`, then VIEW asks for each of the remaining subsets whether it has to be displayed or not, with the keyword `NEXT=`. The default is YES. Giving NO stops the program.

VIEW allows recording of images to create a "movie" of successive images (subsets). Then, start the program together with a specification of the hidden keyword `RECORD=`:

VIEW RECORD=Y

Now, VIEW prompts with the keyword `NEXT=` only once: after the display of the first image, for all the remaining images. All images are positioned the way the first one is positioned. In GIDS a movie can be started after you have specified image numbers in the text box and clicked the 'loop' button. The display rate is adjusted by positioning your mouse in GIDS and clicking the left mouse button. In GIDS, recordings can be done manually also.

5.5 Plotting data

An application is able to create a plot if it prompts with the keyword GRDEVICE=. The default is a list of devices. Examples are:

X	: Tektronix window on screen (b&w)
L1LASER	: QMS laser printer in landscape mode
P1LASER	: QMS laser printer in portrait mode
LCOLOUR	: Colour printer in landscape mode
PCOLOUR	: Colour printer in portrait mode
GIDS	: Gids window on screen (colour)
NULL	: null device (no plot output)

Programs like CPLOT and STAT and lots of other applications capable of plotting, all share this keyword. Usually you will enter X to plot in a Tektronix window (which is automatically generated then) on your screen. If you like your results, send it to a printer by changing the device (f.i. in STAT) or rerun the program with for example !STAT GRDEVICE=P1LASER . After the application is finished, a plot is sent to the QMS laser printer in portrait mode.

CPLOT

Introduction

Among standard ways of presenting astronomical data are contour and grayscale plots. If a map is represented by f(x,y), contours are lines of constant f. This requires the data you want to present with this program must be 2 dimensional. Therefore sets with more than 2 axes need a subset specification at INSET=. Information about the syntax can be found in the GIPSY users manual. The names of the axes can be found with the programs HEADER or DISK. If only a part of the data need to be displayed, use BOX= with four arguments to decrease the size. Input can be a lower and an upper coordinate pair, a coordinate pair and combination with physical coordinates. See the GIPSY user manual for the use of physical coordinates. If you want to display more than one subset of a set INSET=AURORA F 1:10) you get each subset on a blank screen or paper (advancing to a new page or clearing the screen is controlled by the NEXT=<CR> keyword) unless you want to display all the data in a mosaic. With MOSAIC= you can give the number of rows in the mosaic. The program calculates the number of columns. The first plot will be in the upper left corner of the screen or paper. Each subset in a mosaic will now be called 'subplot'.

Contours

The contours at CONTOUR= are a series of floating point numbers given in units of your map. This keyword obeys the syntax for reals as described in the GIPSY user manual. This includes the use of expressions also. An example could be CONTOUR=LOG([10:1000:50]) which is evaluated to

```
1, 1.77815, 2.04139, 2.20412, ...., 2.98227
```

Contours will be plotted with the same line width as the axes of your plot. The blanks in your data are ignored, making gaps in the contour map. The contour levels can be given in percentages also. To put input in percentage mode, select PERCENT=Y . The conversion from percentages to

absolute levels is done with the formula:

$$level[i] = min + perc[i]/100 * [max - min]$$

so that 0the level 'max'. The values min, max are the minimum and maximum values of the data as recorded in the header of the current sub)set. If you don't trust these values or if your box is smaller than the entire (sub)set, use CALCMNMX=Y to calculate these values. The min and

max are also calculated if PERCENT=Y and no header items descriptorDATAMIN and DATAMAX

are found in the header. If the number of given contours is greater than 1, you can select a contour style to distinguish negative from positive contours (STYLE=N) by default, or odd from

even contours (STYLE=O). If you want all contours to be solid, use STYLE=S. Contours are

associated with a color index. This index is an integer number that corresponds to a device dependent color. For a color plot you can use in COLIND= the color indices:

```
0      Background
1      Default (Black if background is white)
2      Red
3      Green
4      Blue
5      Cyan
6      Magenta
7      Yellow
8      Orange
9      Green + Yellow
10     Green + Cyan
11     Blue + Cyan
12     Blue + Magenta
13     Red + Magenta
14     Dark Gray
15     Light Gray
16-255 Undefined
```

The number of color indices must be equal to the number of contours. If COLIND=<CR> the indices are calculated by the program. If at least one value is given but the number of indices is less than the number of contours, the remaining indices are copied from the last one specified. For a black and white hardcopy you can use 0 for a white contour and 1 for a black contour. This combination is sometimes used to emphasize the contours in dark regions of a gray scale plot. If the device is the screen, the color indices will not be very useful. The width of the contours is controlled by `CWIDTH=`. Together with keywordLINEWIDTH= and keywordPLATT=, these keywords control the width of the lines to be plotted. The smallest number is 1 the largest possible number is 21. The `CWIDTH=` keyword is asked hidden just before plotting a contour, making it possible to draw different contours in different widths. The default values of all the keywords containing widths is 1 if the plotting device is the screen or 2 if it is a hardcopy device. **Example:** `CWIDTH=4;3;2;1` The lowest contour has width 4, the highest has width 1. The semi

colons are important here because `CWIDTH=` accepts only one number at a time.

**** Trick:**

If the minimum value 1 is too thick on paper, there is a possibility to set 'setlinewidth' in your postscript file to a smaller value.

Devices

The destination of the plot is selected with `GRDEVICE=` If you give <CR> only, you get a list of devices that can contain for example:

```
X          : Tektronix window on screen (b&w)
L1LASER    : QMS laser printer 1 in landscape mode
P1LASER    : QMS laser printer 1 in portrait mode
LCOLOUR    : Colour printer in landscape mode
PCOLOUR    : Colour printer in portrait mode
GIDS       : GIDS window on screen (colour)
NULL       : null device (no plot output)

etc...
```

On some printers it is possible to plot on a transparency instead of paper. Instructions are obtained at local system management.

Gray scales

Instead or along with drawing contours, it is also possible to fill your plot with shades representing the image values. The shades are given in `GRAYSC=`. The maximum number of shades is 128. Again, input values are floating point numbers and expressions are allowed. The scales are in units of the map and must be in ascending order. **Example:** `GRAYSC=2 3.5 8` Image values smaller than 2 are white in the plot. Values equal to or greater than 2, but smaller than 3.5 get the first shade of gray, values $3.5 \leq f(x, y) < 8$ get the second shade of gray and values ≥ 8 are black in the plot. The available number of different shades depend on the selected device. If `PERCENT=Y` , the gray scale values are percentages.

** Hint

If you want to fill contours smoothly with gray scales, contours can intersect the 'gray scale pixels') simulate more pixels by regridding the map. Use {programREPROJ with changed grid spacings to achieve this effect.

Layout

A plot consists of a title along the lower x-axis and left y-axis, a space for labels containing positions in physical coordinates, a grid margin to reserve space for plotting positions in pixels, the plot itself and an area where important information about the plot is displayed.

Titles

The default titles along x and y axes are the names of these axes as found in the header. Titles can be altered with `XTITLE=` or `YTITLE=` The keywords accept text strings. A string can contain escape sequences. These are character- sequences that are not plotted, but are interpreted as instructions. All escape sequences start with a backslash character (\).

```
\u  Start a superscript or end a subscript;
\d  Start a subscript or end a superscript;
    (u and d must always be used in pairs)
\b  Backspace, i.e. do not advance text pointer after
    plotting the previous character;
\\  Backslash character;
\A  Angstrom symbol;
\gx Greek letter corresponding to roman letter x;
    (Lower case and upper case x give different greek
    characters )
\fn Switch to Normal font
\fr Switch to Roman font
\fi Switch to Italic font
\fs Switch to Script font
```

Like all 'plot' text, the text can contain so called PGPLOT symbols. Available symbols ($\approx$ 1000) are arranged according to Hersey's numerical sequence and are listed in Table B.1 of the PGPLOT manual. Any character can be inserted in a text string using an escape sequence of the form \(n), where n is the 4-digit Hersey number. The positions of the titles are calculated by the program but can be altered with `XTPOS=` and `YTPOS=`. Each keyword accepts a coordinate pair in mm. The numbers are absolute positions in the plot. If for some reason there is not enough space along the axes to place the title, increase the default values (10 mm in x and y direction) at `TITLEMARGIN=` If only one number is given, the y value is automatically copied from this value.

Position in pixels

Positions along the axes can be in pixels and physical values. If `GRID=Y` , each 10th pixel

position will be marked inside the plot box. There is some room reserved default 4 mm in x and y direction) for plotting the numbers. This space is controlled by the keyword `GRIDMARGIN=` which accepts two numbers for space in x and y direction. If only one number is given, the y value is copied from this value. If `GRID=N` (default) there are small grid margins.

Physical values

The labels along the axes indicate the physical positions i.e. if transformation from pixels to physical coordinates is possible. If transformation is not possible, the possible reason is mentioned in your log-file. There the program puts the result of transformation of pixel 0 to a physical position. Conversion always take place to the primary axis units as found in the header (`CUNIT#`). If there are also secondary units, the program will convert the pixels to these units. The default separations between two positions is calculated by the program. A position label is plotted together with a marker called a Tick. Space between two are controlled by `TICK=`. It accepts values for the x and y direction overruling the calculated default values. Negative numbers will be converted to positive numbers. The units are the axis units except if the header unit was 'DEGREE'. The values are in seconds of arc then. If an axis is the RA axis of an unrotated map, the `TICK=` units are in time seconds. Between two major tick marks there are minor tick marks. The default number is calculated by the program. The numbers of minor ticks in x and y direction can be changed with `NMINOR=` If you don't want minor tick marks, use `NMINOR=0 0`

It is possible to plot labels for all axes, but in most plots only the lower x axis and the left y axis do have labels, while the others have ticks only. For a single plot this is the default labeling. In a mosaic, lines connecting two 'subplots' have tick marks only. The decision to plot ticks or labels is made after `AXMASK=` The keywords accepts 8+1 entries. An entry is Y or N. The first four values indicate the plotting of ticks and the second four indicate the plotting of labels. Default is `AXMASK=Y Y Y Y Y N N Y y`

Counting of the axes is anti-clockwise and starts with the lower x-axis. The last value indicates plotting of ticks along axes shared by two plots in a mosaic. The default is Y. If you change this to N, there will be no ticks on the 'inside' axes of a mosaic. Space for the labels along both axes is calculated, but it is not known to the program beforehand how large the labels along the y axis are going to be. Therefore sometimes the space between these labels and the y title is to big. The spacing is controlled with `LABELMARGIN=` which accepts two values. The first is the space in mm in the x direction (for the y-labels) and the second is the space in the y direction.

Rotated images

Along spatial rotated axes, only offsets are plotted. An offset is wrt. a point in your map which is pixel 0 if this pixel is inside the box or the central pixel if pixel 0 is not in the box.

Scales

The plot is scaled in some way to fit on paper or screen. First the physical size in mm of the device plot area is determined. From this number the margins for grids, labels titles and info is subtracted. This number is divided by the number of subplots in one of the directions and is scaled onto the number of pixels in that direction . After a correction for the grid margin we end up with a scale in grids/mm. Changing these values can be done with `SCALE=` but then the scales are converted to units/mm. These units are the physical units as found in the header, only if the units are 'DEGREE', the scale input must be in ARCSEC/mm then. If both axes have the same units, then the scales are made equal in both directions.

Plot information

Information about plot characteristics is placed in the log file and some is also placed in the plot. The keyword that can be used here is `PLOTINFO=` which is N for the screen and Y for other devices. Note that this parameter also influences the scaling in your plot. Info in the plot contains items like the object name, the set name, used box, units, scales etc. It can be extended with a user supplied comment in `INFOCOM=` which is asked in a loop that is aborted with carriage return.The info is closed with a time stamp which is also placed in the log file so that you're always able to find the corresponding information in the log-file (GIPSY.LOG).

Marking of positions

Positions in a plot can be marked. The positions are taken from `MARKPOS=` The keyword is asked in a loop and canceled after each specification. In this way it made suitable for using so called recall files. **Example:** `MARKPOS=<mypos`

and mypos.rcl contains:

```
* 29 30 45 * 4 10 40
* 29 30 30 * 4 10  0
......
```

Each physical position in the recall file will be marked as long as it is contained in `BOX=` Maximum number of positions is 512. Each position is marked with a symbol as selected with `MARKER=` Such a symbol is called a Graph Marker and is associated with a symbol number. The symbol number can be: -1 to draw a dot of the smallest possible size; 0-31 to draw any one of the symbols in figure 4.1 of the PGPLOT manual; or 33-127 to draw the corresponding ASCII character. Examples of symbols are:

```
 1    dot
 2    plus
 3    star
 4    circle
 5    cross
 6    square
 ..   .....
17    small circle filled up
28    Arrow Left
29    Arrow Right
30    Arrow Up
31    Arrow Down
```

Markpos accepts at most the same number of symbol numbers as there are positions given within the box.

Beam

The process of drawing a beam in your plot is started with the keyword BEAMPOS= which accepts a coordinate pair either in pixels or in physical units. Default, the program reads the FWHM and position angle from the header and calculates the minor and major axis of the beam by dividing the header values (degrees) by 7200 to get a value in seconds of arc. The FWHM is read from header items BMMAJ and BMMIN and the angle (in degrees wrt the north) from BMPA. However if the items are not found in the descriptor, you are prompted to give the values yourself. If you want to overrule the defaults use BEAM= for major and minor sizes in seconds of arc, and BEAMPA= for the position angle in degrees. The beam is shaded with straight lines with default 1 mm separation and with an angle of 45 degrees wrt the major axis. Keyword BEAMATT= accepts three values: first value is the 'erase' variable. Default it is set to 0, and nothing happens with the beam, but when it is set to 1, the area in a box containing the beam will be erased with the background color. On screen however you will see the outline only. The second value is the line separation in mm and the third value is the angle in degrees. Default, the beam in a mosaic is plotted in the last subplot, but this can be altered with BEAMNUM= which accepts at most the maximum of subplots integer numbers. Subplot 1 is the upper left plot. If the map is not a spatial map, the beam is replaced by error bars. The default lengths will be the grid spacing in units as found in the header.

Plotting the physical value of the subset

If the set has more than two axes, and a coordinate transformation is possible, you can print the physical value of a subset in the plot. For this you use PLSUB=Y The number of decimals to

be printed is controlled by DECIMALS= which has default 2. The default position is somewhere in the upper left corner, but if you want a different position, use SUBPOS= with a position in pixels or physical coordinates. The position of the text applies to all the subsets in a mosaic. The units are the units as found in the header. If there is a secondary axis defined in the header, the units that correspond to that axis are used. The units are returned to your log-file.

Comments in plot

With keyword `COMMENT=` you can put strings in your plot. The keyword is prompted only if previously `COMPOS=` was defined. As with other positions, `COMPOS=` expect you to give a coordinate pair either in pixels or in physical coordinates. A loop is started now. For each specification of `COMPOS=` there is a `COMMENT=` prompt. The keyword accepts the so called Hersey symbols also. The loop is aborted with `COMMENT=<CR>` . Each comment can be in different height, width, font

etc. These so called text attributes are connected with the text attributes keywords. The text string can also contain escape sequences. These are character-sequences that are not plotted, but are interpreted as instructions.

Text attributes keywords

Input are 7 floating point numbers that control the appearance of the text to be plotted. The input syntax is:

```
XXXXXATT=Hght,Wdth,Fnt,Jstf,Era,Angl,Col   [x1,..,x7]

More specific:

Hght:      Character height
           It's a normalized value corresponding to about
           1/40 of the height of the plot device.
Wdth:      Thickness of chars and lines (1..21):
Fnt:       Font type 1,2,3, 4:
           1  single stroke "normal" font
           2  roman font
           3  italic font
           4  script font
Jstf:      Justification of text: 0=left, 0.5=middle, 1=right
           of a given coordinate.
Era:       Erase text box first before plotting text.
           0 = Do not erase background.
           1 = Erase background for the number of characters
               found in the text.
           n = Erase background for the n characters.
Angl:      Angle of text, 0 is horizontal.
Col:       Character color, for indices see color indices
           above.
```

Not all text obey these attributes. For example labeling has its own justification and has no erase option. Defaults are all determined by the program.

```
Defaults:
LABELATT=1.0, currentlinewidth, 2, 1, 0, 0.0, currentcolor
COMMATT*=1.0, currentlinewidth, 2, 1, 0, 0.0, currentcolor
TITLEATT=1.0, currentlinewidth, 2, 1, 0, 0.0, currentcolor
INFOATT= 0.5, 1,                2, 1, 0, 0.0, currentcolor
SUBATT=  0.6, currentlinewidth, 2, 1, 0, 0.0, currentcolor
GRIDSATT=0.5, currentlinewidth, 2, 1, 0, 0.0, currentcolor

* is a number from 1 to the number of comments in one (sub)plot
```

Extended ticks

The tick marks can be extended so that they are connected by curves following the current projection (`EXTEND=Y`). The number of points calculated in a curve is determined with `LINESTEPS=`

Default is 40 points, for a quick look you can do with less points, but for a quality plot 40 will probably not be sufficient. The extensions of ticks in other sky systems is not yet implemented.

Drawing lines in plot

To add some free style plotting use `PLLINE=` The keyword is hidden but after specification it will be asked unhidden in a loop that is aborted by `PLLINE=<CR>` The keyword accepts at

most 4 floating point numbers i.e. 2 coordinate pairs (in pixels or physical coordinates). The action that follows is the plotting of a line between the two points. If the keyword appears again and you specify only one coordinate the line is plotted from the previous last coordinate to the specified coordinate. An application for these lines is the indication of zero levels for instance. The line width, style and color are controlled by keywordPLATT= The range in width is [1..21], the range in colors is 0..15] and line style is one from:

```
1   Full line
2   Long dashes
3   Dash-dot-dash-dot
4   Dotted
5   Dash-dot-dot-dot
```

Plotting ellipses

keywordELLPOS= is a position in grids or physical coordinates. If the keyword is not empty, you start a loop in which a beam and angle is asked in `ELLAXESPA=` First value of this keyword is the major axis, second the minor axis and the third is the angle in degrees wrt the positive y axis. The units of the major and minor axes are in seconds of arc if both axes in the plot are spatial axes, else, the units are grids. The appearance is controlled by `ELLATT=` and (see remarks at `PLATT=`).

Using overlays

Overlays are made possible by asking in the current plot for a new (sub)set with `INSET2=` The keyword is hidden so it must be specified beforehand. If no more overlays are needed use `INSET2=`<CR> to abort the overlay loop. For each subset it is possible to create overlays. The

program calculates a box for the new (sub)set so that it is contained in the original box in physical coordinates. With the prompted keyword `BOX2=` you can rearrange these values. Pixel 0,0 in the original set is transformed to physical coordinates which are converted to a pixel position in the new set. From this point on all new pixels are scaled with new scales. A new scale is calculated with:

```
                       OldGridSpacing
NewScale = OldScale * --------------
                       NewGridSpacing
```

There is no check on projection. You have to decide yourself whether the overlay makes sense or you have to use the program REPROJ first. The keywords `CONTOUR2=`, `COLIND2=`, and `GRAYSC2=` are described under `CONTOUR=` etc. If you select no original contours and gray scales in a mosaic, but use `INSET2=` for a second (sub)set, it is possible to use different data origins in your mosaic or use different contours per subset etc.
WARNING: If you want to use a grayscale map in your overlays, make sure that this is your first (sub)set. Otherwise a grayscale map will obscure all previous plotted contours in a hardcopy.

examples using default files

1) Create contour plot of subset containing 'short observation beam' (data from A. Broeils). Note the special use of the comment keyword.

```
linewidth=2
BEAMPOS=
BOX=-80 -80 80 80
CONTOUR= -0.1 -0.05 -0.025 0.025 0.05 0.1 0.2 0.4 0.8
GRAYSC=
GRDEVICE=x
GRID= n
INSET= app 1
SCALE=10 10
tick=360 360
nminor=1 1
markpos=
compos=10 51.8;-31 45;-10 29;-43 49;-70 65;
comment=\(2522);\(2522);fan beam;res. axis;(b)
commatt1=1 2 2 0.5 0 126;
```

```
commatt2=1 2 2 0.5 0 -144;
commatt3=1.2 2 2 0 0 36;
commatt4=1.2 2 2 0 0 -54;
commatt5=1.2 2 2 0 1;
xtitle=R.A. - offset (arcmin)
ytitle=Dec. - offset (arcmin)
titleatt=1.5 2
plline=-20 30 10 51.8; -31 45 -20 30;
platt=3;3
labelatt=1.3 2
plotinfo=n
```

The use of a Hersey symbol is demonstrated in: comment=\ (2522); ....

2) Example of mosaic and use of recall files. There are five rows in the mosaic.

```
INSET=clean 1:29:2
BOX=-50 -65 60 35
GRDEVICE=x
PLOTINFO=
MOSAIC=5
LINEWIDTH=1
PLSUB=y
SUBATT=0.8 1 2 0 1
DECIMALS=0
SUBPOS=
PERCENT=n
CONTOUR=-0.48 -0.24 0.24 0.48 0.96 1.44
STYLE=
GRAYSC=
GRIDMARGIN=
labelmargin=25 9.1
SCALE=15 15
GRID=
MARKPOS=<opt2460;
MARKER=
TICK=60 360
NMINOR=1 1
XTITLE=R.A. (1950)
YTITLE=Dec. (1950)
AXMASK=
BEAMPOS=-48 -63
```

```
BEAMATT=
BEAMNUM=2
COMMENT=
COMPOS=;
EXTENDTICKS=
LINESTEPS=
LABELATT=0.8 2
TITLEATT=1.0 2
```

The file 'opt2460.rcl' contains:

```
* 7 52 35.8 * 60 29 01
* 7 52 36   * 60 24 00
* 7 51 57.4 * 60 26 12
* 7 51 56.8 * 60 26 17
```

3) Example use of overlays. De primary set is called '30' and the overlay is the subset 'dtotal.new 1' (data: A. Szomoru).

```
axmask=
markpos=;
BEAMPOS=45 -65
beampa=-98.50
beam=74.79 58.58
BOX= 30 -75 100 -25
BOX2= 30 -35 50 -22
CONTOUR2=0.005 0.01:0.06:0.01
contour=
GRAYSC=2 5:20:5
graysc2=
GRDEVICE=x
GRID=n
INSET=30
inset2=dtotal.new 1 ;
LINEWIDTH=3
MARKER=
PERCENT=
SCALE=3 3
STYLE=
titleatt=1.5,3
labelatt=1,3
```

```
xtitle=R.A. (1950)
ytitle=Dec. (1950)
xtpos=109.208405 10
titlemargin=20 20
markpos=;
compos=;
```

Plotting more overlays is achieved with more (sub)sets in `INSET2=`. For instance use '1005' as primary data and 'warpleft 1' and 'warpright 1' as overlay subsets.

```
........
BOX= -280 -190 230 210
BOX2= 30 -35 50 -22;30 -35 50 -22;
CONTOUR=
contour2=0.002;0.002;
GRAYSC=6.69 17.49 43.94 110.4
graysc2=; ; ;
INSET=/dt1/arpad/bootes/1005
inset2=warpleft 1 ;warpright 1;
........
```

Note the use of semi colons in the file.

4) Example of using the color indices. The first 5 contours (in the dark regions of the gray scale plot) are white and the remaining contours are black.

```
INSET=velo 1
BOX=-60 -50 50 30
GRDEVICE=x
MOSAIC=
PLSUB=
DECIMALS=
SUBPOS=
PERCENT=n
LINEWIDTH=2
CONTOUR=3650:4100:50
COLIND=1::5 0
STYLE=
GRAYSC=3620:4100:10
graysc=
PLOTINFO=
```

```
GRIDMARGIN=
SCALE=5 5
GRID=
MARKPOS=<opt5533;
MARKER=
TICK=20 360
NMINOR=1 2
XTITLE=R.A. (1950)
YTITLE=Dec. (1950)
AXMASK=
xBEAMPOS=-55 -45
BEAMPOS=-170 -140
BEAMSHADE=
COMMENT=NGC 5533
COMPOS=-170 80;
TEXTATT=;1.3;1.5 2 2 0;1.5
INSET2=;
```

5) Example offset axis and the use of PLLINE= to draw offset axes in the plot. The contents of the used recall files is displayed also.

```
AXMASK=
BEAMPOS=
BOX= -30 14 30 40
CONTOUR= -1 -0.5 0.5:2:0.5
GRAYSC=
GRDEVICE= x
GRID=n
INSET= 19slice dec 0
LINEWIDTH=3
gridmargin=0 0
PERCENT=
SCALE=
STYLE=
box=-30 14 30 40
markpos=<xvrot
marker=17;
titleatt=1.5,3
labelatt=1,3
markpos=;
compos=<compos
commatt1=2,3,1,0.5,0,0,1
```

```
commatt2=2,3,1,0.5,0,0,1
comment=<comment
xtitle=Offset along major axis (arcsec)
ytitle=Heliocentric velocity (km/s)
titlemargin=20 20
plline=<plline
platt=<platt
```

Contents of the used recall files:

```
xvrot.rcl: (positions of markers)
-11.3636 33.74362
-10.2272 33.68168
-9.09090 33.58433
-7.95454 33.47814
-6.81818    33.31
.....
11.36363 22.98256
;

compos.rcl: (positions of comments)
-4.59 14
4.59 14
;

comment.rcl: (Hersey symbols as comments)
\(2262)
\(2262)

plline.rcl: (physical- and grid  coordinates are mixed)

0 14 0 40
u 229.286 u 14234.2 u 229.248 u 14234.2
14 17 18.5455 17
16.273 16 16.273 18

platt.rcl: (attributes for the lines, width, style, color,
two lines in long dashes, two lines in full)
1,2,1
1,2,1
3,1,1
```

```
3,1,1
```

5.6 Printing data

PRINT

The program PRINT writes pixel values on the screen or prints them.
PRINT first selects an input set (and subset(s)) and a frame, with the keywords `INSET=` and `BOX=`. Then, the program requests that you assign an output format to the keyword `FORMAT=`. The formats available are floating, exponential, and integer format. To specify a floating format, give a format string beginning with any character except an E, e.g.,
`FORMAT=`FFFF.FF
resulting, for the sample value 3453.32456, in 3453.32. To assign an integer format, use the same format but without a dot, e.g.,
`FORMAT=`TTTT
resulting, for the same value, in 3453. To specify an exponential format, type a format string with an E as first character, for example,
`FORMAT=`EEEE.E
resulting, for the same value, in 3.5E+03. If a carriage return is given as input to `FORMAT=`, then PRINT selects the format best fitting the absolute values of the minimum and maximum found in the header; in absence of this header information, the program selects the exponential format.
The default number of columns the data are written in is 80. To specify another number, use the hidden keyword `TWIDTH=`. For example,
PRINT TWIDTH=120
The default output device is the screen. To select another output device, use the hidden keyword `PRINT=`, described under DISK.

5.7 Statistics

STAT

The program STAT calculates and plots elementary statistics in frames.
First, supply an input set (and subset(s)) to `INSET=`. Then, one or more frames can be defined for the specified subset(s), with the keywords `BOX01=` up to `BOX10=`. To specify the entire subset as the only frame, answer `BOX01=` with a carriage return. Entering a carriage return to a next prompting with `BOX*=` (* ranging from 02 to 10) stops the input of frames.
Next, STAT prints in the Common Output Area per specified frame the results of the calculations. These results are given for each subset in particular and for all subsets together. For a sample set part of the output can be:

```
subnr  pixels       sum       mean         rms        min        max    blanks
==============================================================================
   1   57447        3.01  5.245E-05       .263       -.628       1.67     153
==============================================================================
```

Furthermore, select a graphics device to plot on, with `GRDEVICE=`. Finally, assign plot options to the keyword `PLOT=`. The program repeats prompting with this keyword until an X is entered. For a list of available options, give a carriage return. The plot shows the calculations for all frames and subsets. For example,

`PLOT=SUM PLOT=MEAN PLOT=X`

HISTOG

The program HISTOG calculates some statistics and creates a histogram of values for (sub)sets. First, specify the data, with `INSET=` and `BOX=`. Then, the keyword `RANGE=` can be used to in- or exclude a range of pixel values from the calculations. Next, HISTOG requests a graphics device to plot on, with `GRDEVICE=`.
If the specified data span more than one subset, a plot mode has to be specified, with `AUTOSCALE=`. In the default mode (`AUTOSCALE=` <CR>), the program itself scales the plot for each subset. In the other mode (`AUTOSCALE=NO`), all plots are scaled as the first one.

To determine the range of values along the X-axis of the histogram, HISTOG needs the minimum and maximum data values. If they are not present in the set header, the program prompts for them, with the keyword `MINMAX=`. For example,
`MINMAX=`−4.8701 14.857
Next, the keyword `FIXBIN=` allows one to choose how the bin width of histograms has to be specified, by giving the bin width itself (`FIXBIN=YES`) or by giving the number of bins (

`FIXBIN=`<CR>). HISTOG prompts for the bin width with `BINWIDTH=`. E.g.,

`BINWIDTH=`10
HISTOG prompts for the number of bins with `BINS=`. The default is 100.
Subsequently, some statistics are displayed for the specified subset(s), in the Common Output Area of the Hermes Window. Furthermore, the Y-axis interval, that is, the range of counts, can be entered with `COUNTS=`. As a default, the entire range is selected. For example,
`COUNTS=`5 20
To include a Gaussian curve in the plot(s), see the document HISTOG.DC1.

5.8 Manipulating sets

COPY

The program COPY copies (sub)sets, with the option to change subset sizes.
The keyword `INSET=` prompts for specifying the input (sub)set(s). Next, a frame can be selected for the input subset(s) to define the output subset(s), with `BOX=`. `BOX=` allows for entering a

frame greater in size than the input subset(s). Then, in the output set, the values of the pixels that cannot be copied from the input set are set to blank.
Furthermore, an output set has to be assigned to OUTSET=.

SCALE

The program SCALE scales the data in subsets by a factor A and an offset B. The output subsets have the format $A \times$ *input subset* $+ B$.
The first three keywords SCALE prompts with are INSET=, BOX=, and OUTSET=. OUTSET= has as a default the input set name.
To specify the scaling factor A and the offset B, SCALE prompts with AB=. For example, to assign the value 3.5 to the scaling factor and 2 to the offset, the input
AB=3.5 2
has to be given. The default values of A and B are, respectively, 1 and 0.

CLIP

The program CLIP has two options:

1. CLIP transfers values of an input set to an output set if they lie within a preset range, setting to blank values outside that range.
2. CLIP blanks values of an input set in an output set if they lie within a preset range, transferring to the output set without change the values outside that range.

First, the program requests that you give values to INSET=, BOX=, and SETOUT=, to specify, respectively, the input set (and subsets), a frame for the subsets, and an output set.
The keyword RANGE= is used to specify the range of values. To specify the preferred program option, choose one of the following input formats.

1. A range of values $x <$ *value* $< y$ to be transferred has the input format:

   RANGE=x y.

2. A range of values $x <$ *value* $< y$ to be blanked has the input format:

   RANGE=y x.

For example, RANGE=2 5 results in a transfer of all values falling within the range $2 <$ *value* $<$

5 (and a blanking of all other values), and RANGE= 8 3 results in setting blank the values falling within the range $3 <$ *value* < 8 (and transferring all other values).
To specify a range, the values INF and −INF, representing the minimum and maximum values of the system, also can be used. For example,
RANGE=−INF, 4
results in a transfer of all values smaller than 4.

CONDIT

The program CONDIT provides, like CLIP, a conditional transfer of values from an input set to an output set. Here, the condition (set by RANGE=) applies to the values of another input set, the so-called "test set".
Consider the following example. A radio astronomer has a series of images with varying frequency of a particular sky area. Only one of them shows clearly a number of sources, having values above 50 mJy. CONDIT can be used to identify these sources in the other images as well: pixel values of the input set are transferred to the output set if their corresponding values in the test set are higher than 50 mJy.
The program requests that you specify an input set (INSET=), a frame for the subsets (BOX=), a test set (SETX=), and an output set (OUTSET=). Then, the transfer condition has to be specified, by assigning a range of values to RANGE=. For the above example, the appropriate range is
RANGE=50 INF
passing values corresponding with test values higher than 50, while blanking all the other values.

EDITSET

The program EDITSET edits data in a set on pixel-by-pixel basis.
The program requests a set (and subsets), with INSET=. Furthermore, a frame can be specified, with BOX=.
To change the specified data, either an expression can be assigned to the keyword EXPRESSION= or, skipping this keyword by giving a carriage return, one or more values can be entered with NEWVAL=.
If you choose to specify an expression, use as parameters axis names (RA, DEC, etc.) and DATA, meaning the old data value. Furthermore, see the document EDITSET.DC1 for a list of all mathematical functions, constants, and operations allowed. For example,
EXPRESSION=DATA*ABS(RA)
meaning that the selected pixels are to be multiplied by the absolute value of their RA grid position.
Choosing the "manual mode", give input to NEWVAL= for the pixel position displayed on the screen in a message of the format:

```
New value at (x1,...,xn):                    [old value]
```

For example,

```
New value at (1,5,2):                [0]
```

If two or more pixels were selected, then the program repeats prompting with NEWVAL=. Giving a series of input values prevents repeated prompting. For example,
NEWVAL=1::4
assigns the value 1 to the displayed pixel as well as to the next three pixels that were selected.

DECIM

The program DECIM decimates a set along one or more axes by (an) integer factor(s). DECIM first selects an input set (and subset(s)) and a frame, with INSET= and BOX=.

Subsequently, a decimation factor can be entered for each axis, to the keyword DECIM=. As input, only integers are accepted. If subsets with the axes RA, DEC, and FREQ were specified, then DECIM= can be answered with, e.g.,
DECIM=2 1 3
decimating the RA axis by the factor 2 and the FREQ axis by the factor 3. On a decimated axis, the value zero always is retained. If, for example, the RA axis ranged from −8 to 7, then the decimated RA axis has the positions −8, −6, −4, −2, 0, 2, 4, and 6 of the original axis.
Furthermore, DECIM asks for an output set, with OUTSET=.

CBLANK

The program CBLANK changes a certain value into BLANK and/or changes a BLANK into a certain value. First, CBLANK requests to specify the input (sub)set(s), with INSET=. Then, the keyword BLANK= asks for a value to be blanked. As a default, BLANK itself is selected.
Subsequently, the keyword VALUE= selects a value to replace BLANK. The default value is again BLANK.

5.9 Combining sets

ADD, SUB, MUL, and DIV

The programs ADD, SUB, MUL, and DIV perform the respective arithmetic operations of addition, subtraction, multiplication, and division. They can be used either to combine a series of subsets (or frames) with a single subset (or frame) or to pairwise combine two series of subsets (or frames).
The *first* option results in a series of calculations in which the first operand has changing values and the second operand remains the same in value. The chosen program requests the first operand, a series of subsets, with INSET= and BOX=, and the second operand, one subset, with SETX=. The last keyword is OUTSET=.
The *second* operation results in a series of calculations in which the first operand is the *n*th subset of the first input set and the second operand is the *n*th subset of the second input set. For example, in the program SUB, specifying

INSET=SAMPSET1 FREQ 1:15 BOX=<CR> SETX=AURORA FREQ 1:15 OUTSET=SAMPSET2

results in subtracting the first subset of the second input set, AURORA, from the first subset of the first input set, SAMPSET1, that is
(SAMPSET1 FREQ 1) – (AURORA FREQ 1)
followed by
(SAMPSET1 FREQ 2) – (AURORA FREQ 2)
(SAMPSET1 FREQ 3) – (AURORA FREQ 3)
etc.

MEAN

MEAN calculates the mean of a number of subsets in a set.
First, similar as to SUM, the input set and the axis along which to average pixels have to be assigned with the keyword `INSET=`. Subsequently, the keyword `BOX=` can be used to define the frame over which the input subsets have to be averaged.
To specify weights for individual subsets, `WEIGHT=` can be used. By default, equal weights are assigned.
Finally, MEAN requests that you give a value to `OUTSET=`.

SUM

The program SUM sums a number of subsets of a set.
First, SUM asks, with the keyword `INSET=`, for the input set and the axis along which pixels have to be integrated. E.g., if the set AURORA has three axes:

RA-NCP	from	-7	to	8
DEC-NCP	from	-7	to	8
FREQ-OHEL	from	1	to	59

and DEC is the axis along which pixels must be summed, then the input has to be
`INSET=`AURORA DEC
With the following keyword, `BOX=`, SUM requests the frame over which the input subsets have to be added.
Subsequently, with the keyword `CUT=`, cutoff levels can be set, a lower and an upper cutoff level. Data values not between these levels are ignored in the sum.
The last keyword is `OUTSET=`, asking for the set where the results must be stored.

COMBIN

The program COMBIN can be used to perform several mathematical operations for more general purposes upon sets.
First, a mathematical expression is requested, with the keyword `RESULT01=`. In this expression, input sets have to be represented by a symbol with the format $\$n$, where $1 \leq n \leq 32$. COMBIN later asks for the full specifications of these parameters. For example, to specify an expression that adds two input sets, type
`RESULT01=`$1 + $2
or, to specify an expression that adds gaussian noise to an input set, with a mean of zero and an rms of two, give
`RESULT01=`$1 + RANG(0, 2)
The last example contained the function RANG. For a list of all mathematical functions, constants, and operators allowed, see the document COMBIN.DC1.
With the keywords `RESULT02=`, `RESULT03=`, etc., COMBIN allows for specifying more mathematical expressions, up to 32. A carriage return stops COMBIN asking for more expressions. Subsequently, COMBIN requests that you fully specify the respective input sets, subsets, and

frames, with the keywords `SET01=, BOX01=, SET02=, BOX02=`, etc. All input sets must have the same number of subsets. The output set(s) get(s) the same size as the first input set.

5.10 Saving sets

WFITS

The program WFITS transfers data on disk to magnetic tape or disk, in FITS format.
First, WFITS selects the data to transfer, with the keyword `INSET=`.
Pixel values are transferred as integers of 8, 16, or 32 bits. The program prompts for the number of bits with the keyword `BITPIX=`. The given number determines the range of integer values, preferably matching the range of input data values, as well as the size of the output file, preferably taking up no more space than necessary. Input data outside the specified range are represented by the lowest value in that range. For antenna patterns, 32 bits is advised. The default number is 16.
WFITS searches for the minimum and maximum values of the input data in their header. If it cannot find these values, it prompts with the keywords `DATAMIN=` and `DATAMAX=`. For example,

```
DATAMIN=−3.6 DATAMAX=17.2
```

Finally, an output device has to be assigned to `TAPE=`. For a list of available devices, give a carriage return. To write the data to a subdirectory on disk, enter the name of the subdirectory.

Chapter 6

Reducing radio data

6.1 Introduction

6.2 General radio applications

- CLEANing
- SMOOTHing
- (HI) profiles
- PBCORR
- Converting mJy to Kelvin

6.3 WSRT specific applications

still to come

6.4 VLA specific applications

still to come

Chapter 7

IRAS reduction

7.1 Introduction

This chapter describes IR-GIPSY, the GIPSY software for the reduction of data obtained from the Infrared Astronomical Satellite (IRAS). It is intended as an introduction for the novice IR-GIPSY user. For more general information on GIPSY, see chapter 5. Section 7.2 introduces the IRAS instrument[1]. Section 7.3 describes the data base for IRAS data. Section 7.4 presents the software for these data.

7.2 IRAS

7.2.1 Instrument and mission

The Infrared Astronomical Satellite (IRAS) surveyed about 95 % of the sky, in four broad spectral bands centred on 12, 25, 60 and 100 μm, during a ten month period in 1983. A low-resolution spectrum of the bright sources was obtained as well, with the Low-Resolution Spectrometer (LRS). Precession at a rate of about 1° per day, kept the orbit of the spacecraft remaining perpendicular to the earth-sun vector (Figure 7.1). The spacecraft attitude control system permitted the instrument to scan in ecliptic longitude along small circles at constant elongation[2] between 60° and 120°. Most survey scans, however, were taken within 6° of the quadrature. For each spectral band there were two detector arrays. The arrays were arranged such that the second one scanned the same area of sky some 5 to 10 seconds later than the first one.
A semi-overlapping scan strategy was used for the "all-sky" survey. Redundant coverage on the time scale of hours was provided by advancing the instrument in elongation by half of the width of the focal plane on a subsequent scan, usually the next orbit. A complete survey conducted in this fashion is designated as an HCON (Hours Confirmation), and locally consists of (almost) parallel scans which overlap.
Roughly 70 % of the sky was surveyed three times in such a manner, 25 % was covered twice, and about 5 % was missed. Thus most regions near the ecliptic plane were observed 12 times,

[1] For more information, see *Infrared Astronomical satellite (IRAS) catalogs and atlases, volume 1 of Explanatory Supplement, NASA reference publication 1190, Washington, DC, 1988.*

[2] Here, elongation designates the angle between the telescope axis and the IRAS-Sun vector.

Figure 7.1: A schematic drawing of the orbital geometry. The orbital altitude, 900 km, and inclination 99°, combined with the earth equatorial bulge lead to a precession of the plane of the orbit about 1° per day. As a result, the orbit normal always pointed towards the Sun as the satellite orbited above the Earth's terminator. By pointing the satellite radially away from the Earth, the cold telescope was shielded from the heat loads from the Sun and Earth while providing natural scanning motion across the entire sky in about six months. A sequence of hours-confirming scans on the celestial sphere is also shown. A celestial object does not change its position on the sky, during the passage over the IRAS focal plane and is seconds (see figure 3.2) and hours confirmed.

Figure 7.2: IRAS focal plane. Of the 62 infrared detectors the three filled-in were inoperative; the cross-hatched detectors showed a higher noise level. The normal scan direction of objects farther away than a few 100,000 km is shown. Detectors at the four wavelength bands are ordered in two arrays per band, allowing confirmation of the detection of a source within 5 to 10 seconds.

on time scales of seconds, hours, and months. The overlap of the scans increases toward the ecliptic poles. Scans from different HCONs over the same area usually intersect at an angle due to precession of the satellite orbit. Therefore, even small areas of the sky can be very unevenly covered. The layout of the focal plane (see Figure 7.2) and the data processing for the initial IRAS product, the IRAS Point Source Catalog (PSC), was optimized for detecting discrete unresolved celestial sources.

Each of the 59 active IRAS detectors has a different response function. Most detectors have rectangular apertures of 0.75 to 3 (in the in-scan direction) and 4.5 to 5 arc minutes (in the cross-scan direction). Consequently, the spatial resolution is different in the two directions, significantly worse in the cross-scan direction.

The scanning speed of IRAS was 3.85 arc minutes/sec. Thus the in-scan length of 0.75 arc minutes of a 12 or 25 μm detector was traversed in 0.2 seconds. That is, in just 3 data samples a point source is visible. All astronomical objects, including solar system objects, exhibit the same timing at an instantaneous crossing. Solar system objects, however, will in general not be hours-confirmed.

During all survey observations, LRS observed as well. The LRS was basically an objective prism spectograph, oriented in such a way that the dispersion was aligned with the scan direction. Therefore, it measured useful spectra of relatively isolated point sources only. Two overlapping wavelength bands were scanned simultaneously, one ranging from 7.7 to 13.4 μm and the other from 11.0 to 22.6 μm. In both bands, the resolution increased from 20 at the short-wavelength to 60 at the long-wavelength side. The integration time per resolution element was 60 ms, which resulted in noise levels equivalent to 1.5 Jy at the shorter and 3 Jy at the longer wavelengths.

The LRS aperture mask in the focal plane measured 6 arcmin in-scan by 15 arcmin cross-scan (see figure 7.2). This wide field of view ensured a sky coverage identical to that of the IRAS survey: 96 percent of the sky was observed at least twice and 72 percent three times. To reduce spatial confusion, the aperture width was covered by three detectors at the short wavelengths and by two at the long wavelenghts. The detectors were sampled continuously, and the data were received on the ground together with the data from the survey array.

7.2.2 IRAS data products

IR Data products available in GIPSY:

- Survey band 1, 2, 3, 4, LRS
- AO band 1, 2, 3, 4, LRS
- Spline band 1, 2, 3, 4

7.2.3 Pointing reconstruction

7.2.4 Timing

IRAS timing is done in so-called satcal ticks. A satcal tick is not quite equal to a second (actually 1 tick ~1.00005 seconds). The satcal tick time units are used everywhere in IR-GIPSY, unless explicitly stated otherwise. Satcals and satcal ticks are interchangeable.

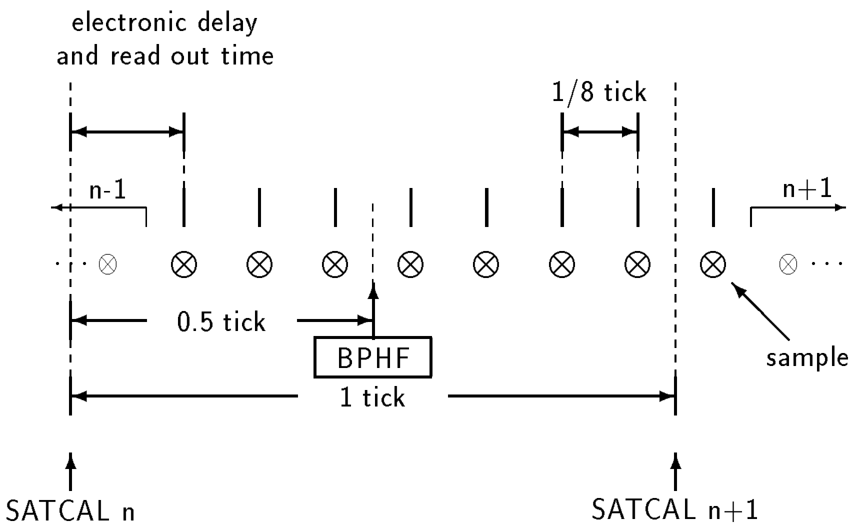

Figure 7.3: What happens in a satcal tick

Within one satcal tick a number of samples (from 1 for splines up to 32 for LRS data) are taken. The timing parameters of the samples and Boresight Pointing History File (BPHF) with respect to satcals are indicated in figure 7.3 (for 60μ data with 8 samples per satcal). As shown in the picture, the BPHF is valid for the time *mid between* two satcal ticks. Furthermore, the samples are offset with respect to the satcal due to the read out time (different for each detector) and the electronic delay (different for every band). Note that this offset can be negative. IR-GIPSY programs account, if timing is at importance, for these different effects.

7.3 IRAS data in GIPSY: IRDS

7.3.1 Introduction

This section describes the database for raw IRAS data, the InfraRed Data Structure (IRDS). Raw IRAS data represent the data as they are sent down from IRAS to a ground station: a series of digitized voltages corresponding to sky brightness as a function of time. By defining a particular on-sky area and extracting from the database the raw data of a certain band associated with that area, a so-called "custom plate" is created. The part of a scan falling within a custom plate is called a snip. So, a custom plate corresponds to a collection of snips in a single band. See figure 7.4 for a sample custom plate. Subsection 7.3.2 discusses how raw data are ordered in IRAS data sets. Subsection 7.3.3 deals with the header information in such data sets.

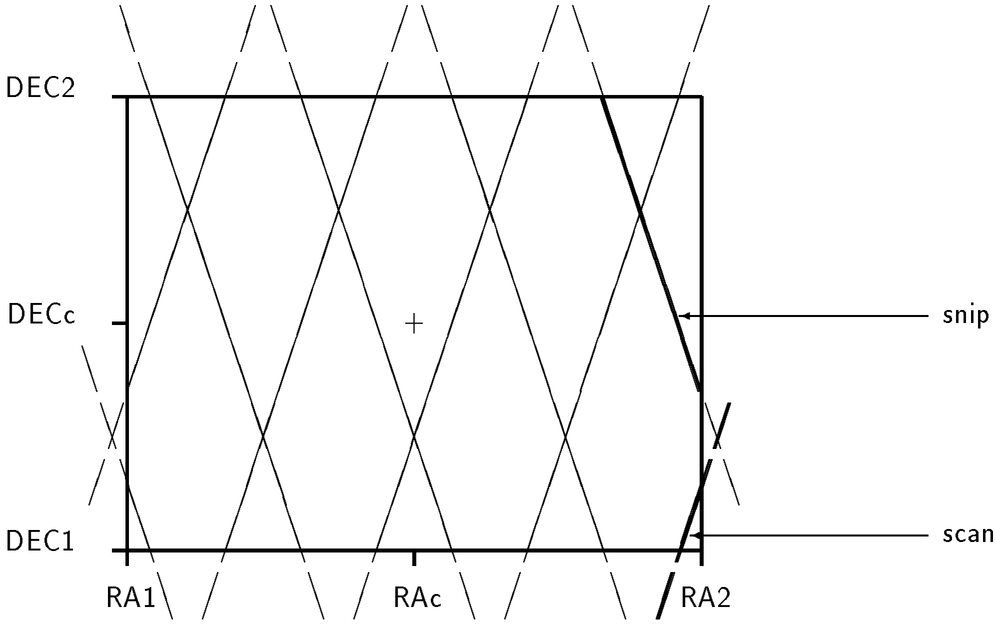

Figure 7.4: Sample Custom Plate with CENTER = (RAc,DECc) and SIZE = (RA1−RA2) (DEC1−DEC2).

7.3.2 Data structure description

Raw IRAS data in GIPSY are stored as four-dimensional data sets. Going from fastest to slowest axis, in an IRSD the raw IRAS data are ordered as follows:

1. samples (per satcal tick) (SAMPLE axis)
2. satcal ticks (TICK axis)
3. sequential detector number (SDET axis)
4. snip number (SNIP axis)

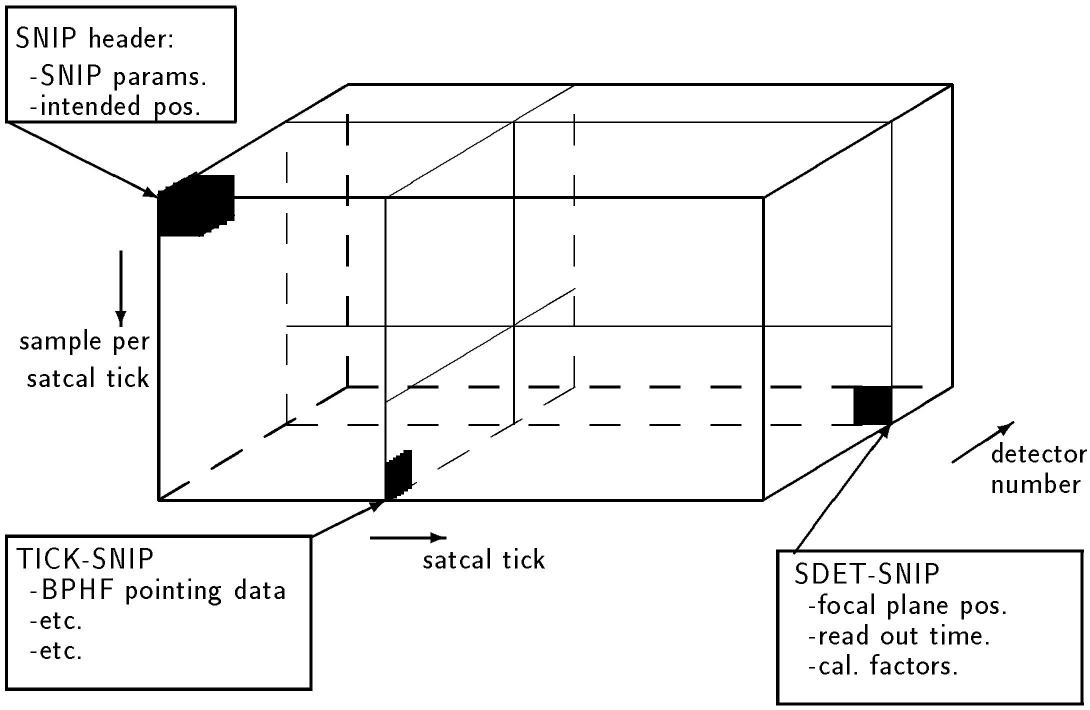

Figure 7.5: GIPSY implementation of a single snip

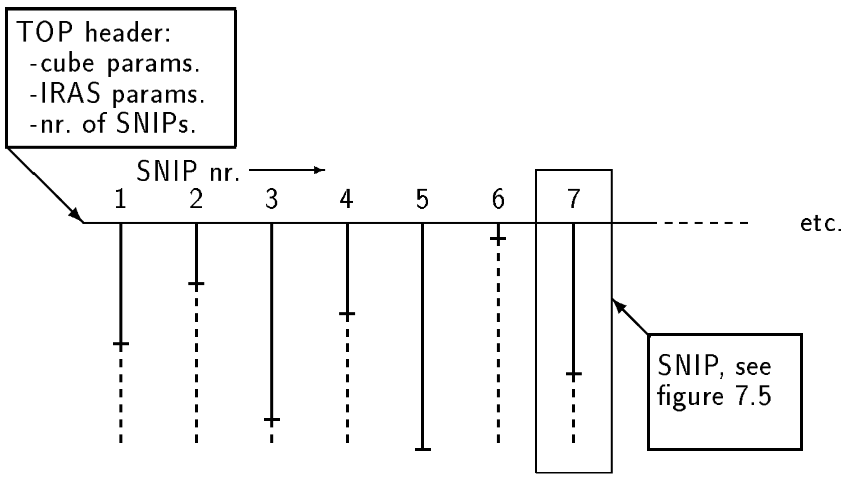

Figure 7.6: GIPSY implementation of an IRAS Data Structure (IRDS)

Figure 7.5 shows the way an individual snip (the part of a scan which falls within a custom plate) is put into a three-dimensional data structure. Figure 7.6 shows a complete IRAS Data Structure (IRDS) consisting of a series of snips from different scans. Since snips are collected by, e.g., SNIPping parts of scans that run through a given custom plate (see section 7.4.2, program PLATE), in general different snips have different lengths. As a result, the shorter snips in an IRDS have to be padded with undefined values(see figure 7.6). Using this padding scheme, in the worst case 50% of a given IRDS will be empty.
Different "instruments" (for example, LRS, Survey band 1, Survey band 2, etc., Spline band 1, etc.) are *all* treated as different bands. Different bands are put into different data sets.

7.3.3 IRDS headers

Part of the header information must be present to allow programs to work on the data. The program PLATE automatically creates these headers. Examples of header items at top level are `NAXIS`, giving the number of axes in the data set, `INSTRUME`, specifying the instrument/band, and `SKYSYS`, specifying the coordinate system of the snip area. At the SDET level, at least `DETECTOR`, giving the detector number, must be present. Some of the obligatory headers at the snip level are `SNIPCAL`, giving the satcal ticks since the beginning of the snip, and `SNIPDUR`, giving the length of the snip in satcal ticks.
For more detailed information about IRDS, see the IR-GIPSY Programmer's Manual.

7.4 IRAS reduction programs

7.4.1 Introduction

This section describes how to use the GIPSY software for reduction of IRAS data. Subsection 7.4.2 deals with programs that create IRAS data structures, PLATE and RDBPHF. Subsection 7.4.3 presents some programs that can be used to display raw IRAS data, TRACKS, PLSCAN, and VIEW. Subsection 7.4.4 concerns a program for intensity calibration of the raw survey data in these structures, ZODYCAL. Subsection 7.4.5 discusses a program that creates survey maps, ***. Subsection 7.4.6 presents a program that calibrates LRS data, LRSCAL. More details about the discussed programs can be found in the relevant .DC1 documents.

7.4.2 Getting IRAS data into GIPSY

This subsection describes two programs to get IRAS data into GIPSY, PLATE and RDBPHF

PLATE

The program PLATE builds an IRAS Data Structure for a particular area on the sky. All IRAS data pertaining to such an area is called a "custom plate" (see figure 7.4). The layout of a custom plate is defined by specifying values for the keywords `CENTER=`, `SIZE=`, and `COOR=`.
To start the program, give
PLATE

The program asks then to identify the instrument, with `INSTRUME=`. The identification consists of observation mode and band name. Observation modes are SURVEY, AO, SPLINE, etc. Band names are 12, 25, 60, 100, LRS, etc. The band names 12, 25, etc., indicate a 12 μm band, a 25 μm band, etc. For example,

`INSTRUME=SURVEY 25`

to read 25μ survey data.
PLATE prompts then with the keyword `CENTER=`. The center of the custom plate must be entered here. Some examples of values are

`CENTER=3h25m28s -18d16m3s`

and

`CENTER=46.6 -18.2`

The next keyword is `SIZE=`. The default custom plate sizes are 1.0 by 1.0 degrees. For example,

`SIZE=1.5 2.8`

results in a custom plate measuring 1.5 degrees in longitude and 2.8 degrees in latitude.
With the keyword `COOR=`, PLATE requests the standard coordinate system. The default coordinate system is EQUATORIAL 2000.0. A sample input value is

`COOR=GALACTIC`

With the following keyword, `IRSET=`, PLATE asks to give the name of the set to which the data have to be written. For example,

`IRSET=B-THREE`

where B-THREE is an arbitrary set name.
The last two keywords, `OBJECT=` (for example, `OBJECT=CAS A`) and `OBSERVER=` (for example, `OBSERVER= YTSMA`), add the names of object and observer to the custom plate. A carriage return leaves these names in the IRDS blank[3].

RDBPHF

The program PLATE adds the intended positions as found on the input optical disk into the header of the IRDS. This results in a positional accuracy of ~10 arcminutes.
Better position accuracy is obtained when the Boresight Pointing History File[4] (BPHF) data are used. The IRAS program RDBPHF adds BPHF information to the header of the IRDS.
After starting up this program (to start a program, just type the program name), RDBPHF requests the name of the input IRDS, with the keyword `IRSET=`. For example,

[3] More sensible default values will be inserted later.
[4] In GIPSY the latest version of the BPHF is being used which is sometimes called the Super BPHF

IRSET=B-THREE

where B-THREE is the name of a set created with PLATE.

7.4.3 Inspecting raw IRAS data

This subsection explains three programs to inspect raw IRAS data: TRACKS, PLSCAN, and VIEW.

TRACKS

The program TRACKS plots IRAS tracks.
First, TRACKS asks for an input set, with the keyword IRSET=. Then, the keyword SIZE= prompts for the size of the plot, in longitude and latitude. The plot has the same centre as the specified set. The default size equals to the size of the input set. For example,

SIZE=2.5

Next, enter the snips to plot, to SNIP=. As a default, all snips are plotted.
Subsequently, give the sequential detector numbers to plot, to SDETS=. By default, all sequential detector numbers are plotted.
Furthermore, the program requests an output device, with GRDEVICE=. To obtain a list of the available devices, give a carriage return. The list looks like

```
NULL        TEKTRONIX        L1LASER        P1LASER
```

where NULL means "no plot", TEKTRONIX indicates the screen, and both P1LASER and L1LASER are devices for hardcopies.
Finally, the keyword POS= allows for specifying positions to be plotted. In the plot, they appear as plusses. As a default, no positions are plotted.

PLSCAN

The program PLSCAN plots raw IRAS scans. This program can plot the data as function of the in-scan distance to a reference position, to assign to the next keyword, POS=. Such a position has to lie within the coordinates defined in the set header. As a default, PLSCAN plots the data as function of the satcal ticks.
PLSCAN first asks to specify an input set, with IRSET=. Next, the program requests that you specify the sequential detector numbers to be plotted, with SDET=. Give the first and the last of the series of numbers. As a default, all detectors are plotted.
The keyword SORT= asks if the specified detectors have to be plotted according to the layout of the focal plane. The default value is YES.
PLSCAN allows for plotting the detectors aligned in the time domain. Then, enter a carriage return to ALIGN=.
The keyword TICKS= prompts for the part of the snip to be plotted. Here, the first and last satcal tick of that part can be typed. A carriage return selects the entire snip.
Finally, PLSCAN requests the graphics device to plot on, with GRDEVICE=.

VIEW

Still to come: use of VIEW for IRAS

7.4.4 Calibration of survey data

The calibration program ZODYCAL calibrates the raw data of an IRAS data structure created by PLATE and possibly –although not necessarily– enriched with BPHF information by RDBPHF.
First, ZODYCAL requests the name of the set it has to calibrate, with the keyword `IRSETIN=`.
Then, the program asks to specify the name of the output set, with the keyword `IRSETOUT=`.
The default is the name of the input set.
With the next keyword, `ZODY=`, the program asks if the zodiacal emission model has to be subtracted. The default is YES.
The last keyword, `UNITS=`, prompts to specify the units of the output set. The default units are MJy/sr. Other input can be, for example,

`UNITS=Jy`

7.4.5 Map making

still to come

7.4.6 Calibration of LRS data

The program LRSCAL generates calibrated LRS spectra from an IRDS.
The input data set is requested with the keyword `IRSET=`. This data set must contain LRS data.
Otherwise, LRSCAL generates an error message and returns with `IRSET=`. Then, the position at which one or more spectra must be generated has to be specified, with the keyword `POS=`. The default position is the center of the custom plate (see subsection 7.4.2, program PLATE, keyword `CENTER=`). Subsequently, LRSCAL displays the snip numbers of the spectra in the data set that correspond to the positions specified with `POS=` on the screen.
With the keyword `IRSETOUT=`, the program prompts for the name of the output IRDS. As a default, no output set is generated.
The keyword `SNIP=` allows you to specify the snip number(s) of the spectrum or spectra to be generated. The program allows for repeated input to this keyword.
The *first* appearance of `SNIP=` has by default value all snip numbers in the data set. Other input can be, for example,

`SNIP=1`

or
`SNIP=`1:4
where 1:4 implies snips 1, 2, 3, and 4.
LRSCAL then prompts *again* with `SNIP=`. Giving a carriage return results now in calibration of all *remaining* spectra. If *all* spectra in the set have to be calibrated, then give
`SNIP=`ALL

LRSCAL generates, in addition to a specified series of spectra, an average spectrum of that series. So, entering
`SNIP=1:4 SNIP=5:7`
differs, with respect to the generated average spectra, from entering
`SNIP=1:3 SNIP=4:7`
To stop LRSCAL from prompting with `SNIP=`, give
`SNIP=QUIT`
The program prompts then again with the keyword `POS=`. A carriage return causes LRSCAL to use the same positions as entered previously. The program stops by giving
`POS=QUIT`
After the *first* time `SNIP=` is asked, the program prompts with the keywords `UNITS=` and `GRDEVICE=`. With `UNITS=`, the units of the data after calibration are requested. The default value is Jy (Jansky). With `GRDEVICE=`, the program requests a graphics device for display of the generated spectra.

Chapter 8

Advanced use of GIPSY

8.1 Recall files

Keyword values can be extracted from default or recall files. The "*recall file*" serves as input to the keyword. It must be a text file with a name like "*name*`.rcl`". Every line in this file is a separate input and either the whole file or a part of it can be specified (See chapter 4).
Example: In CPLOT it is possible to mark positions. The coordinates of these positions can be given manually at the `MARKPOS=` prompt or with a recall file. A recall file 'positions.def' contains for example:

```
* 3 13 33.51 * 41 16 34.2
* 3 13 55.67 * 41 15 9.5
* 3 13 59.41 * 41 12 9.3
* 3 13 59.74 * 41 14 19.1
* 3 14 3 * 41 8 50
```

If you want to plot positions 2, 3 and 4, use: `MARKPOS=<positions` 2:4

8.2 Default files

Another way to pre-specify keywords for a task, is the use of a "*default file*" containing keywords specifications. Default files have the format "*name*`.def`". At present only the current working directory is searched for default files. Only one keyword specification per line is allowed. Example: The default file 'myplot.def' is used to control CPLOT and contains:

```
inset=velo2 2
BOX=-48 -30 35 10
CALCMNMX=Y
PERCENT=
CONTOUR=-10 -5 5 10 140:250:10
MARKPOS=<positions 2:4
MARKER=
GRAYSC=150 160 170 180
```

```
MOSAIC=3
SCALE=20 40
GRIDS=y
STYLE=N
BEAMPOS=20 -20
line=1
tick=19 10*6
nminor=3 3
xlabel= R.A. (1950.0)
ylabel= Dec. (1950.0)
xtitle= R.A. (1950.0)
ytitle= Dec. (1950.0)
deltashade=0.5
grdevice=x
```

In tHermes CPLOT using this default file is started with: myplot(CPLOT). Keyword values specified in the UCA are taken into account first. Keywords that are not prespecified, are prompted by CPLOT.

8.3 GIPSY in batch mode

GIPSY can be used in a non interactive way. For this there is a special Hermes version called nHermes. The syntax for nHermes is explained in chapter 4.

- Example:

  nhermes –lMylog "FLUX, INSET=cg1517 v "

  Let 'nhermes' start task FLUX on all subsets of set cg1517 and put results in file Mylog.LOG.

8.4 COLA FILES

8.4.1 Introduction

COLA is a small (but powerful) high-level control language for use with Hermes (in tHermes type run `COLA`, with nHermes, a COLA file is one of the parameters). Using COLA you can write a program to execute a sequence of commands. A COLA script file must have the extention .col In such a COLA file you can use a number of facilities, like:

- variables (and substitution of their values in commands)
- a conditional statement (IF)
- repetitive statements (FOR, WHILE, REPEAT)
- input/output with Hermes console (READ, WRITE)

COLA compiles a source program to some intermediate code, and, if no errors are detected, starts to execute that code. If an error is found, the line containing the error is printed, together with an explanation and compilation is stopped. To introduce the general form and appearance, an example COLA program is presented, which shows a couple of maps on your monitor, prints them, and copies them to another set.

```
!  example COLA program
!  date: 15 Jul 1985

integer i                                       ! loop counter
integer nsubs                                   ! number of subsets
nsubs = 32                                      ! default number
READ "Subset numbers [%nsubs] ?" nsubs          ! prompt user
FOR i = 1 , nsubs                               ! for all subsets...
   WRITE "Processing subset %i"                 ! give message
   "VIEW,INSET=NGC4214 %i,CLIP="                ! display
   "PRINT,INSET=NGC4214 %i,BOX=,FORMAT="        ! print
   IF i <= 10 THEN
      "COPY,INSET=NGC4214 %i,OUTSET=OUT1 %i" ! first 10 to OUT1
   ELSE
      "COPY,INSET=NGC4214 %i,OUTSET=OUT2 %i" ! others to OUT2
   CIF
CFOR
```

This example shows some aspects of COLA, all of which will be discussed in detail further on. But a few points can be made immediately:

- COLA makes no distinction between upper and lower case letters
- layout doesn't affect the meaning of a program
- comments can be added (the rest of a line, following ! is skipped, except when the ! occurs in a string)

8.4.2 Cola syntax

Constants

COLA knows 4 datatypes:

integer	1, 0, -12, etc.
real	3.14, -.01, etc.
logical	True, False
string	"This is a sample string"

Variables

Variables which are introduced into a program are given "identifiers" by the programmer. An identifier is a name, made up from letters and digits, starting with a letter. The length of a name is physically determined by the length of a source line, although only the first 8 characters are significant. Some words are "reserved", and have a special meaning in the language. Therefore they may not be used as names for variables. The reserved words are:

REPEAT	WHILE	READ	ELSE	FOR	MOD	GE	LE
OR	UNTIL	HALT	THEN	AND	IF	GT	LT
CWHILE	WRITE	CFOR	CIF	NOT	NE	EQ	

Before use, variables have to be declared first. Declarations for the different types look like this:

integer	i,j,nsets
real	f, pi
logical	quit
string	message
string*10	keyw

The value of a variable can be substituted in a string by prefixing the name with a percent sign (%): "The number is: %num !!" At runtime the sequence '%num' will be replaced by the actual value of the variable num.

Remarks:

- to put a percent sign in a string, it should be prefixed by another percent sign
- to get a variable substitution followed by a letter or digit, separate them with an underscore (_)

Expressions

COLA has three kinds of expressions: arithmetic, logical and string. Arithmetic expressions are made up from integer or real constants, variables and the arithmetic operators:

+	plus
−	minus
*	times
/	divide (DIV for integer arguments)
MOD	modulo function

For logical expressions the relational operators:

=	or	EQ	equal
<>	or	NE	not equal
<	or	LT	less than
<=	or	LE	less or equal
>	or	GT	greater than
>=	or	GE	greater or equal

and the logical operators: AND, OR and NOT can be used. Example expressions:

```
5
num
2*(num+3)
num < 10
```

```
(a LE b) OR (c > 0)
```

Assignment

An assignment gives the value of an expression to a variable. Example:

```
nr = 3
subset = subset + 1
message = "Processing set: \%iset"
quit = ( num > last )
```

8.4.3 Cola statements

Task statement

The task statement is the most important one of COLA. It is used to start a servant task. The statement itself is just a string. Since these strings have to be passed through to Hermes, they receive a special treatment:

- lowercase letters are replaced by their uppercase equivalents
- multiple spaces are replaced by single spaces

It is also possible to continue one string over more than one line. Examples:

```
"VIEW,INSET=\%set \%subset"
"print,inset=aurora freq 1,
 box=,format=xxx.xx"
```

The last one just becomes:

```
"PRINT,INSET=AURORA FREQ 1,
 BOX=,FORMAT=XXX.XX"
```

READ statement

This statement is used to read a new value for a variable from the console. Variable with the same name as keywords might cause problems. The user can enter a new value, or keep the current (by default). You also have the option to specify a prompt string for the Task Status Area.

Example: The statement sequence

```
MYSET = "AURORA"
READ "Name of set [\%MYSET] ? " MYSET
```

leads to the following question at the console:

```
- COLA     * Name of set [AURORA] ?

COLA  ,MYSET=
```

If you omit the prompt string, COLA will display:

```
"Value for <variable> "
```

Of course it is also possible to supply a value when starting up the program with:

```
COLA, NAME=somename , SET=AURORA
```

WRITE statement

This statement writes a string to the screen (and log file). It is made up of the keyword WRITE and a string. Example:

```
WRITE "Now processing set \%set \%subset"
```

N.B. The first character of the string is treated as a fortran carriage control character (1 gives newpage, etc)

HALT statement

A HALT statement immediately stops the execution of a COLA program. This can be used to stop a program if a task stops abnormally.

IF statement

The IF statement allows a choice between two possible statement paths:

```
IF logical_expression THEN
...
ELSIF logical_expression THEN
...
ELSIF ...
.
.
ELSE
...
CIF
```

The ELSIF's and/or ELSE parts of this statement may be omitted.

FOR statement

This statement can be used to execute a body of statements a (fixed) number of times. It takes the following form:

```
FOR cv = lwb , upb , step
...
CFOR
```

The variable 'cv' is the control variable, 'lwb' and 'upb' (resp. the initial and final value) are general (arithmetic) expressions and 'step' is an integer constant which denotes the step size. The last may be omitted.

> Remarks:
> - although the control variable shouldn't be changed inside the loop, there's no checking against it (but I'm working...)

REPEAT and WHILE statement

These two statements can be used to repeat a body of statements an (in advance unknown) number of times. The statements take the following form:

```
WHILE logical_expression
...
CWHILE
and
REPEAT
...
UNTIL logical_expression
```

The program COLA is used as an interpreter of your COLA file. It can also list source and generated code, make a storyboard and has some control over canceling keywords (See cola.dc1).

8.5 Using different machines

Still to come.

8.6 Programming in GIPSY

For people who want to make minor changes in existing GIPSY programs, or want to write major new applications, there is a Programmers Guide available. 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.

Part II

The GIPSY architecture

Chapter 9

System architecture

9.1 Introduction

This chapter gives an overview of the system architecture of GIPSY. Section 9.2 describes the development of the GIPSY system. Section 9.3 gives a general introduction to X Window System, the protocol for the network GIPSY is part of. Section 9.4 presents an overview of the components of the GIPSY system. The other sections describe these different components of GIPSY in more detail.

9.2 History

GIPSY started in 1971 as a large single, interactive program. It ran on a PDP-9 computer with a single-user operating system and modest hardware resources. It could only process WSRT continuum data. In 1976, GIPSY also ran on a CDC 6600, with a multi-user operating system. To GIPSY's processing capabilities, WSRT line data reduction facilities were added. In the late seventies, GIPSY became a set of independent programs. This independency prevented that faults in one application influenced the whole system. A PDP 11/70 computer replaced the PDP-9. An M70E image processing computer, connected to the PDP 11/70, provided advanced image processing and display facilities. At the same time, processing of TAURUS data was implemented in GIPSY.

In 1979, the master control program HERMES was added to the system to make it more user-friendly. In 1984, the PDP 11/70 computer was replaced by a VAX 8600. As a result, GIPSY had to be adapted to the operating system VMS: many software modules had to be rewritten. Some years later, GIPSY could also process VLA image data, IRAS post-map data, and optical data from other instruments than TAURUS. Starting in 1990, an effort began to make GIPSY portable. It now runs under UNIX and has been ported to the following platforms: Alliant(Fx4-Fx80), SUN (Sparc), DEC 3100, and the HP400 and 700 series. A number of programs to process IRAS pre-map data were added to the system. For display, GIPSY makes use of the X Window System. Most of the functionality of the VMS GIPSY is present under UNIX, though further development will continue over the coming years.

9.3 X Window

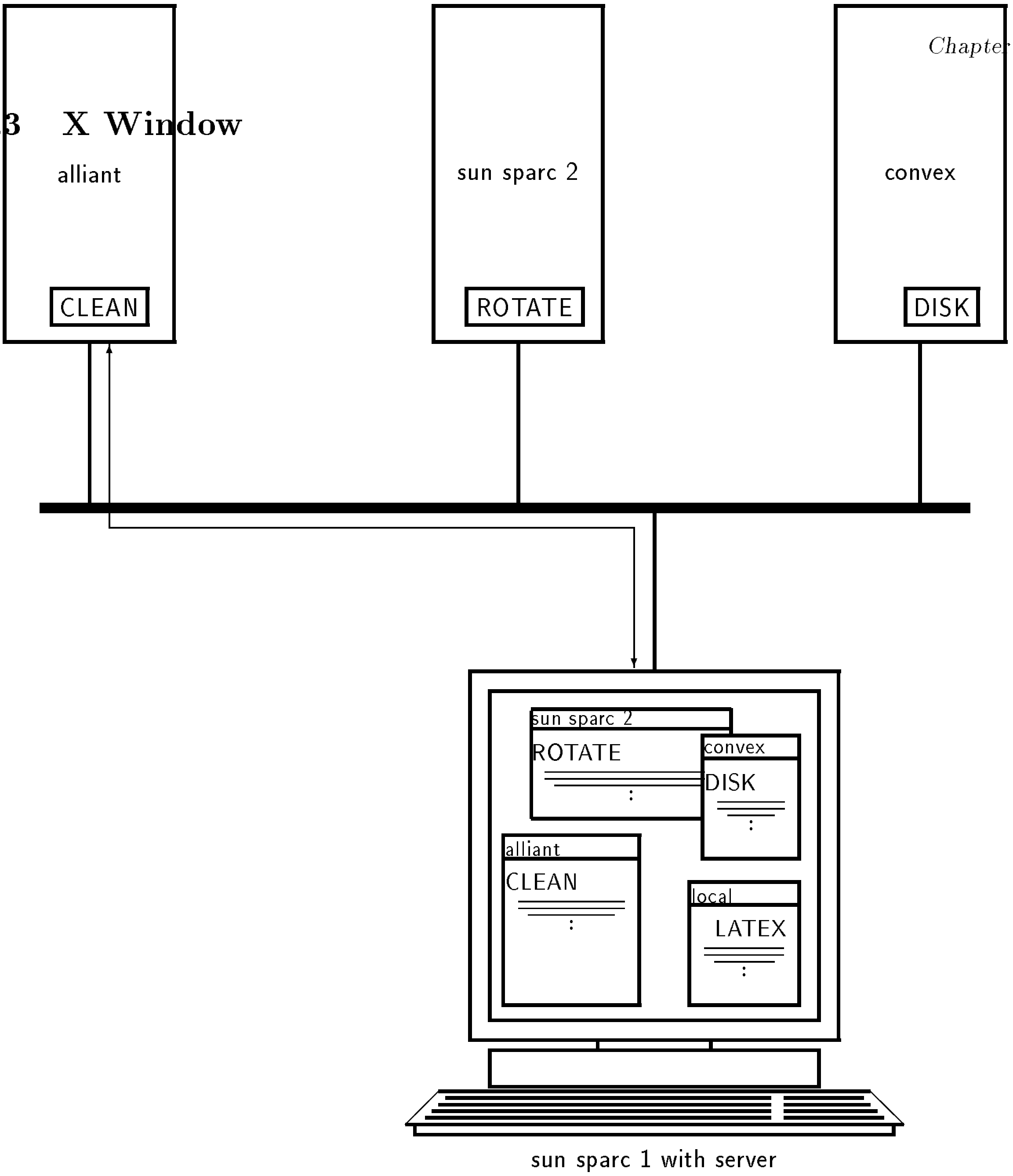

Figure 9.1: Distributed processing using X Window.

This section should contain some remarks on:

- X11 philosophy
- windows
- display customization (tiled, overlapping, cursor, colors etc.)
- client-server model

- distributed processing (refer to figure 9.1
- from here on we say X11 rather than X Window

9.4 System architecture

Figure 9.2 shows the main components of the GIPSY system and its environment and how these components are connected with each other. The user runs a GIPSY session from a workstation that is part of a network that uses the X11 protocol. To start up an application program, the user gives a command to the master control program HERMES. HERMES interprets the command and starts up the application task. An application consists of a main program and subroutines: utilities and interface subroutines. For the output of images and plots to the screen and the plotter, the main program uses the display interface and the plotting interface PGPLOT. To communicate with the database, it uses the interface GIPSY Database Subsystem (GDS). It communicates with the user through the user interface. The user and control interfaces are closely connected to HERMES.

9.5 HERMES

HERMES is GIPSY's master control program. It controls the transfer of messages between parts of the system. It mediates

- between the operating system of the host computer and the user
- between the operating system and applications
- between applications and the user
- between applications and applications

It not just passes the messages, but it also simplifies and improves the communication in many ways. Some services it offers to the user are

- The user need not address the operating system in its own command language.
- HERMES provides the user with information about the status of applications in the HERMES window. For example, that an application is running, pausing, requiring input, or has finished its job, either normally or caused by an error.
- HERMES keeps a log file of all activities and presents pages of this file on the screen.
- The user can store command character strings for frequent use in macros. HERMES replaces every occurrence of a macro name in the user input by the appropriate character string.
- HERMES stores user input to an application in an application macro. This facilitates repeating an operation with the same input values.

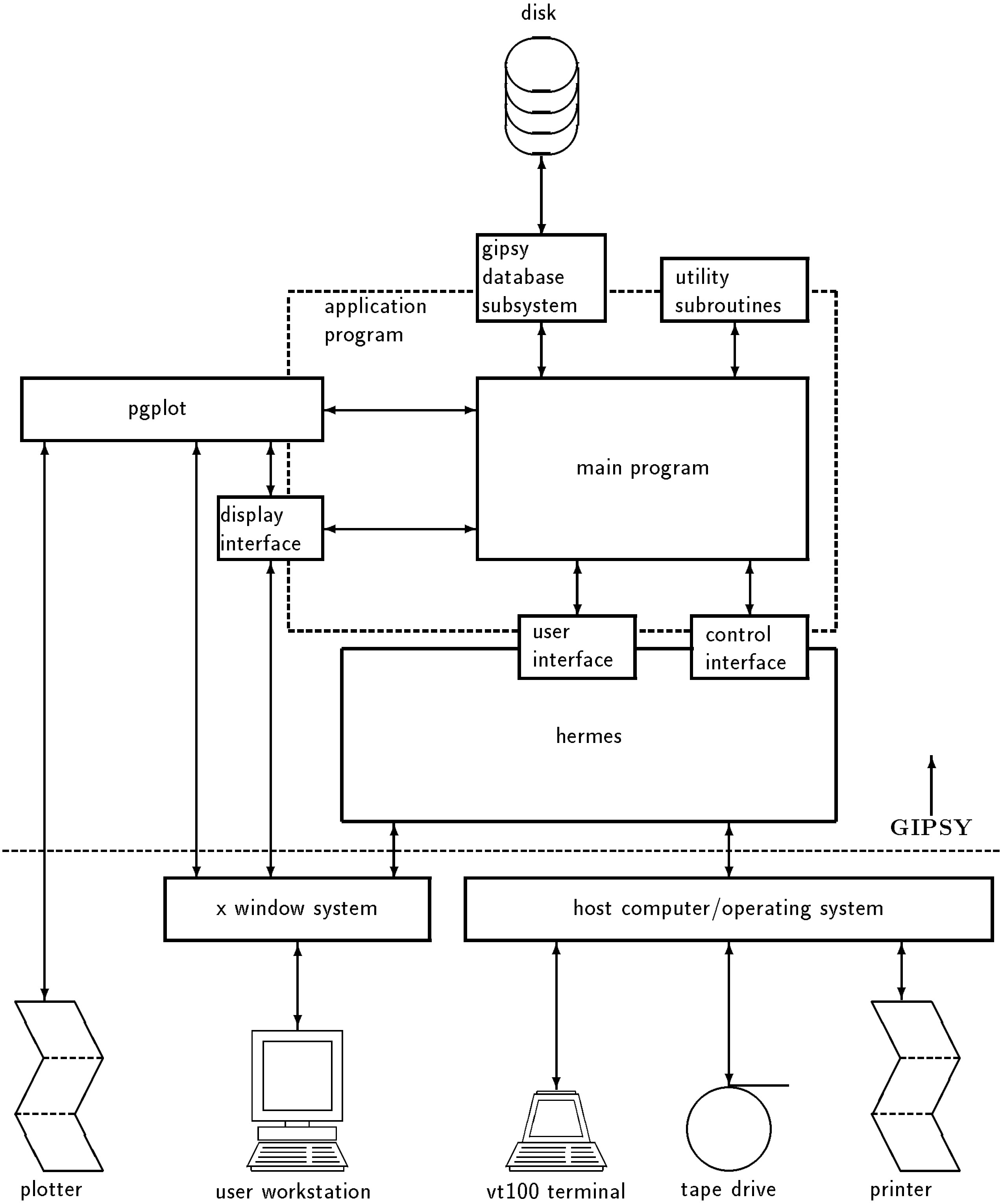

Figure 9.2: GIPSY system architecture.

- HERMES allows control characters for some frequently used commands.
- The user can give input to an application in free format. HERMES converts it into the format requested by the application and then passes it. Examples of HERMES' user-friendly input can be found in chapter 4

As figure 9.2 indicates, HERMES is closely connected to the user interface and the control interface. The user interface is a group of subroutines for the communication between the user and an application program. The control interface is a group of subroutines. Sections 9.9 and 9.10 discuss these interfaces in more detail.

9.6 GIPSY and X11

This section discusses some customizations of X11 that have been made for GIPSY. It has to be noted here that advanced users of X11 still have the possibility to change the choices made for GIPSY according to their own preferences.
GIPSY defines two types of windows: the HERMES window and the display window. The HERMES window is used to start up GIPSY applications, to provide them with input and to receive their output. In the display window, plots and images appear.
In GIPSY, windows have a rectangular form. The focus window is the window that contains the pointer. When a window gets focus, it appears in the foreground.
While GIPSY applications are running, non-GIPSY programs, with their own windows, may be started as well. They may run on the same, but also on other computers.
To run GIPSY, it is not necessary to use X11. It can also be run from a VT100 compatible terminal, with a character display. Then, the screen functions only as a HERMES window and there is no display of plots and images possible. Furthermore, X Window System itself can create a so-called xterm window that functions as a VT100 compatible terminal.

9.7 Database

Data enter GIPSY in FITS (Flexible Image Transport System) format. Data transfer programs convert FITS files into files in the internal GDS (GIPSY Data Structure) format.
The database consists of logical units called sets. A set contains data and their header information. The data structure can be seen as an n-dimensional array. The header information consists of descriptor items with associated values. These items are defined according to the FITS standard or are of any computer supported type, for example, a real or a character. They describe the data in a set. Some apply to all data in the set, others only to the data of a subset.

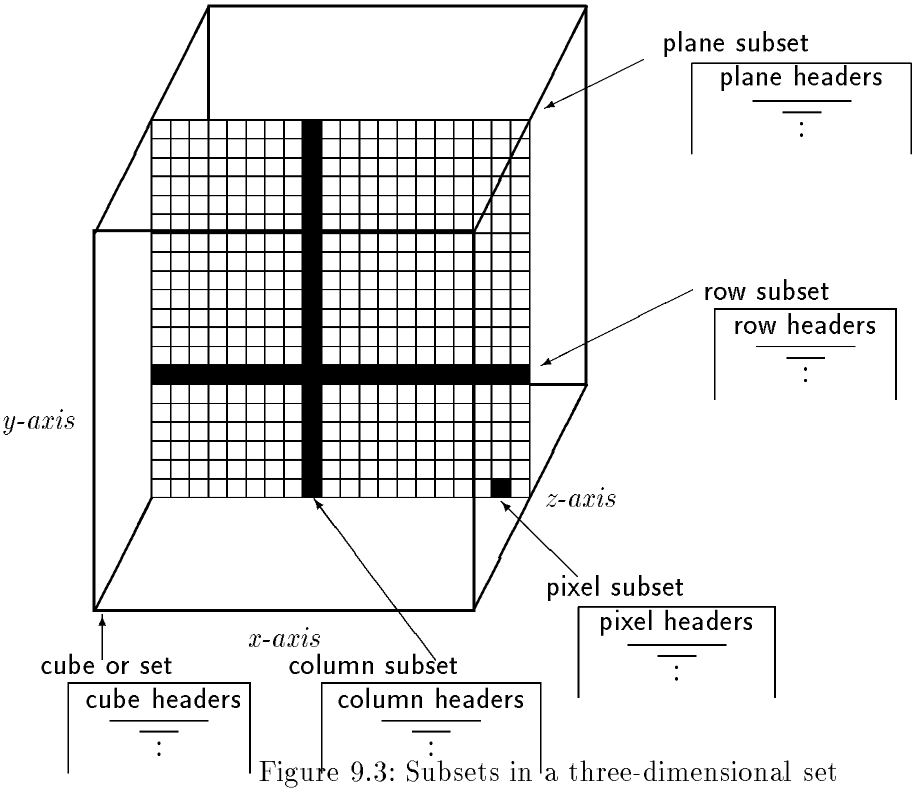

Figure 9.3: Subsets in a three-dimensional set

A subset of an n-dimensional set is an m-dimensional part of the set, m being smaller than or equal to n; the subset contains all data and related header information in the m-dimensional sub space of the set.

Example Figure 9.3 shows the possible subset types in a three-dimensional set, a cube:

- the *cube* itself, a three-dimensional subset
- *planes* parallel to the lateral planes of the cube, two-dimensional subsets
- *rows* and *columns* parallel to the main axes of the cube, one-dimensional subsets
- *pixels*, zero-dimensional subsets

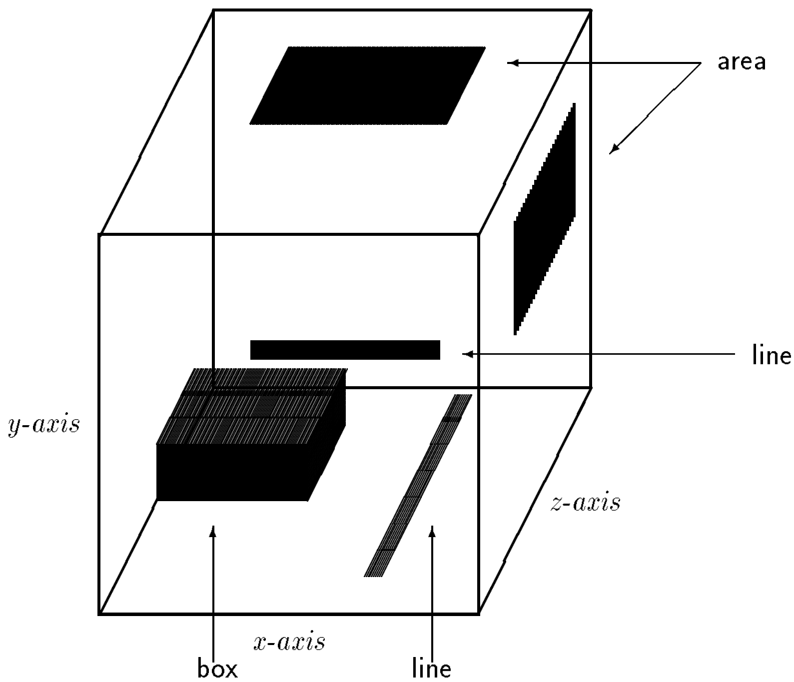

Figure 9.4: Frames in a three-dimensional set

Each subset can be characterized by one or more descriptor items. Header information on a subset can be attached to the subset itself or to subsets encompassing it.
In addition to subsets, GIPSY defines frames. A frame is a part of the data of an m-dimensional subset. Such a part is also m-dimensional, but has for one or more of its dimensions not the full range of values at that dimension.

Example Figure 9.4 shows the possible types of frames in a three-dimensional set:

- a *box*, as part of the cube (sub)set
- *areas*, as parts of plane subsets
- *lines*, as parts of row and column subsets

Header information on a frame is found in the descriptor items of the subsets encompassing it.

9.8 Database interface: GIPSY database subsystem

Through GIPSY Database Subsystem (GDS), an application program accesses the database. GDS contains the following types of subroutines:

- *basic* routines, that create, delete, and close files, test their existence, and increase the number of their dimensions
- *image read/write* routines, that retrieve and store image data
- *descriptor read/write* routines, that retrieve and store descriptor items
- *coordinate* routines, that transform coordinates of one system into coordinates of another system

Image read/write routines select a whole file or a part of it. Parts can be subsets or frames.
Descriptor read routines retrieve specified header information of a specified subset. If they find no relevant header information on that subset itself, then they search for it on subsets encompassing it.
Coordinate routines can transform physical coordinates into grid coordinates and grid coordinates into file coordinates. To access data in the database, programs need their file coordinates. File coordinates indicate where data are stored on disk. To process data, programs need their grid coordinates. The user can specify data by grid coordinates as well as by physical coordinates.

9.9 User interface

An application program communicates with the user through the user interface. This communication is not direct but via HERMES. For example, when an application program needs user input, it sends a message to HERMES. HERMES can pass this message to the user, but it also can return the desired input from an application macro.

The user interface consists of input and output subroutines.
Input subroutines allow an application to select the parameter values that it needs for its execution. It sends to Hermes a request to give a value to a certain parameter name. In GIPSY terminology, a parameter name is called a keyword. The keyword value can be obtained in a number of ways (see chapter 4).
Examples of output subroutines are

- *ANYOUT*, that sends text to the screen and/or HERMES' logfile
- *ERROR*, that sends error messages to the screen
- *STATUS*, that sends information about the program execution
  to the screen

9.10 Control interface

still to come

9.11 Display interface

Display application programs use the display interface (GIDS) to send images and related information to the display window on the screen. The display window contains a menu of options that enables the user to adjust the color, contrast, and other features of the displayed image.

9.12 Plot interface: PGPLOT

Plot applications use the plot interface to send plots and related information to the display window on the screen or to a plotter. GIPSY uses the plot interface PGPLOT, a standard package. For some functions, PGPLOT uses subroutines of GIPSY's display interface, for example, to display coordinate frames.

9.13 Application programs

For GIPSY programmers, the main programming languages are SHELTRAN and C. A SHELTRAN precompiler translates SHELTRAN source text into FORTRAN source text, which subsequently is processed by a standard FORTRAN compiler.
Figure 9.5 shows the structure of a GIPSY application program. (For a more detailed picture, see the dashed box in figure 9.2.) The levels indicate the distance to the operating system, level 3 being most close to it. The core part of an application is the main program. To fulfill some of its tasks, it makes calls to subroutines: standard interface subroutines and utility subroutines.
The standard interface subroutines perform frequently used tasks that can be subdivided according to function:

- *database* interface subroutines

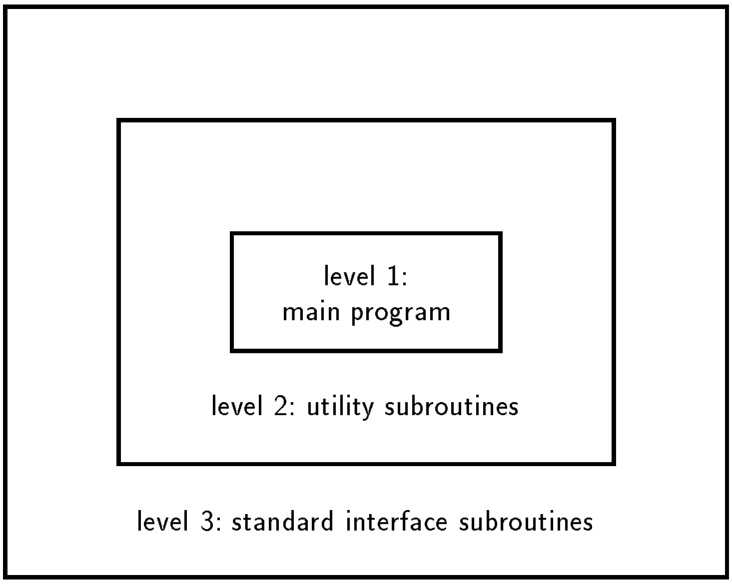

Figure 9.5: Application program structure

- *display* interface subroutines
- *plot* interface subroutines
- *user* interface subroutines
- *control* interface subroutines

The utility subroutines perform a variety of tasks that most programs use, for example, sorting algorithms.
The documentation of applications shows three levels:

- main program documents, with the title format <*PROGRAM NAME*>.DC1, for example, RDFITS.DC1.
- interface subroutine documents, with the title format <*SUBROUTINE NAME* >.DC2, for example, ANYOUT.DC2
- utility subroutine documents, with the title format <*SUBROUTINE NAME*>.DC3, for example, BACKUP.DC3

In addition to these documents, there are level zero documents, providing general information about the GIPSY system, for example, SOFTWARE.DC0, logging current software developments.

Appendix A

GIPSY commands

A.1 Keyword syntax

A.2 List of GIPSY tasks

Tables A.1 lists all GIPSY tasks that are avalable on October 4, 1995.

Program	Purpose
ADD	Add a series of input subsets and one or another
AID	Selects one of the tasks SUB, MUL, ADD, DIV or CONDIT.
ANTPAT	Calculate beamwidth and position angle of synthesized beam.
AOSNIP	Snip AO-scan into seperate legs
BLOT	Conditional transfer of maps using blotch region.
BUG	reports (a fix of) a bug.
CBLANK	Converts datavalues to BLANK and vice versa.
CLEAN	This program cleans maps
CLIP	Blank values of input if they are outside range.
CODER	Create C-code for simple programs.
COLA	Control language for use with Hermes.
COLOUR	Loads a COLOUR Look Up Table in GIDS. This LUT can be activated with the USER button in the COLOR menu.
COMBIN	Combine sets to create new sets using mathematical expressions
CONDIT	Transfer values if test value within certain range.
CONREM	Removes continuum from channel maps by fitting a polynomial to the continuum channels.
COORDS	Converts user supplied coordinates into physical coordinates.
COPY	Program to copy set and subsets. It can also increase or decrease the frame size of the subsets. BLANKs will be written in the output set outside the original subset frame.
CORMAP	Check whether there is a correlation between two maps.
CPLOT	contour and gray scale plot
CREATE	Creates a SET (and subsets)
DECIM	Decimate set in any direction with an integer factor
DELETE	Delete sets
DISK	DISK lists the contents of the current directory and lists some properties of sets.
DIV	Divides a series of input subsets by one or another series of subsets.
EDITSET	Program to edit data in a set.
ELLINT	Programme fits an ellipse to an user defined bunch of points.
ELLFIT	Integration of map in ellipses.
ENHANCE	Enhance image (Local enhancement method)
FIXHED	Add, change, delete or list item(s) in set header.
FIXSNIP	Adds the keywords OBS and SCANTYPE to the header if these are not present in an IRDS.
FLAT	Apply a flat field correction to a map by fitting a 2-D polynomial to the background.
FLUX	Program to calculate flux in user defined box.

Table A.1: List of available GIPSY tasks as of October 4, 1995

Program	Purpose
GAUFIT	GAUFIT calculates initial estimates and/or fits multi-component gaussians to profiles.
GAUPROF	GAUPROF examines influence of parameters in GAUFIT by plotting Gaussian estimate and fit of a selected profile.
GIDS	Description of the Groningen Image Display Server.
GRADIENT	Calculates the gradients or the laplacian of a map, using three by three sized templates.
HEADER	Display header information
HERMES	Master control program for GIPSY.
HISTOG	Program creates histogram of intensities in a set
IMAGE	Imager and/or destriper for IRAS
INIDISPLAY	Initializes the display for GIPSY applications.
INSERT	Program to insert (parts of) a set into another existing set.
IOTEST	Tests the speed of disk io.
IRDSPOS	Extract projected sky positions from positions
IRSET	Generate a test IRDS
LFITS	List FITS files on a tape
LRSCAL	Generate a calibrated LRS spectrum from data in an IRDS
MEAN	Program to calculate the mean of a number of subsets.
MERGE	Merges a number if IRDS's.
MIRROR	Mirrors a two-dimensional image in x, y or both.
MNMX	This program finds the minimum and maximum value in a subset or part of a subset. It also counts the number of blanks.
MOMENTS	Program to calculate moments and other properties
MTCOPY	Copies selected files from one tape device to another
MUL	Multiplies a series of input subsets with one or another series of subsets.
NEWS	Displays the GIPSY news which you have not yet seen.
NHERMES	GIPSY master control program for a non interactive environment
NOHERMES	Provide user communication and process control functions
OUTAGE	Remove possibly corrupt data around outage from irds
PATCH	Patch up a map, by interpolating through holes.
PBCORR	Corrects maps for primary beam correction
PGDEMO	Demonstrates PGPLOT.
PLATE	Snips raw data from tape(s) to custom plate
PLSCAN	Plots raw IRAS scans
PPLOT	Plot the intensity along a line through a set.
PRINT	Print part of a (sub)set on terminal/printer.
QFIT	Fit 2-dim gaussian to data in box
QPLOT	QPLOT plots an one dimensional slice through a set.
RDBPHF	Add BPHF information to an existing IRDS
REPROJ	Transfers sky map from one coordinate system to another
RESTORE	This program restores the residual map(s) created by CLEAN.

Table A.2: List of available GIPSY tasks as of October 4, 1995 (cont.)

Program	Purpose
REWIND	Rewinds a tape.
RFITS	Load FITS files from tape into a GIPSY set.
RHEAD	List header elements (descriptors) and their level in grids
SCALE	Program to scale the data in subsets by a factor and an offset
SKIP	Skips files forward or backward on tape
SLICE	Make a slice through a set
SMOOTH	Smooth sets with a gaussian beam
SNIPIN	Generate a test IRDS
STAT	Program to calculate elementary statistics in user defined box(es).
SUB	Subtract one or a series of subsets from another series of subsets.
SUM	Program to add a number of subsets.
TABLE	GIPSY Table maintenance program
THERMES	GIPSY master control program for ASCII terminals
TRACE	TRACE plots image values at user given positions
TRACKS	Plots IRAS tracks.
VELSMO	Program does a one dimensional smoothing of subsets.
VIEW	Displays two dimensional images.
WEED	Copies snips to an irds (or part of it), or removes snips.
WFITS	Writes disk set and subsets onto magnetic tape in in FITS format
XHERMES	GIPSY master control program for X Window
ZODYCAL	Zodiacal emission calibration program

Table A.3: List of available GIPSY tasks as of October 4, 1995 (cont.)

Appendix B

Sky Projections used in GIPSY

GIPSY knows about 4 Sky coordinate types. The kind of coordinate is conveyed into the first characters of the axis type as 'RA...', 'DEC...' etc. The Sky coordinates known are:

'RA'	Equatorial, Right Ascension
'DEC'	Equatorial, Declination
'GLON'	Galactic Longitude
'GLAT'	Galactic Latitude
'ELON'	Ecliptic Longitude
'ELAT'	Ecliptic Latitude
'SLON'	Supergalactic Longitude
'SLAT'	Supergalactic Latitude

GIPSY knows of 10 different projections from rectangular grids to sky coordinates and vice versa. This chapter discusses all these projections. The type of geometry is conveyed in the last characters of the axis type, as '...TAN', '...NCP', etc. The known projections are:

'AIT'	Aitoff equal area projection
'CYL'	Equivalent Cylindrical projection
'FLT'	Flat projection
'TAN'	Gnomonic projection
'SIN'	Orthographic projection
'ARC'	Rectangular projection
'GLS'	Transversal projection
'NCP'	Nort Celestial Pole projection
'STG'	Stereographic projection
'MER'	Mercator projection

To separate the coordinate type from the projection type at least one minus-sign (−) must be present ('RA−NCP', 'DEC−NCP').
In a projected plane, the position of a point (x, y) with respect to the coordinate reference point in an arbitrary linear system may be represented as

$$x = Lcos(\rho) + Msin(\rho)$$

$$y = Mcos(\rho) - Lsin(\rho)$$

where ρ is a rotation, L is the direction cosine parallel to latitude at the reference pixel, and M is the direction cosine parallel to longitude at the reference pixel. Both the (x, y) and (L, M) systems are simple linear, perpendicular systems. If we represent longitude and latitude with the symbols α and δ and the projection centres with α_0 and δ_0, we get the conversion routines shown in the following sections.

B.1 Aitoff equal area projection

$$L = \frac{2f_\alpha cos(\delta)sin(\frac{\alpha-\alpha_0}{2})}{Z}$$

$$M = \frac{f_\delta sin(\delta)}{Z} - M_0$$

where

$$Z = \sqrt{\frac{1 + cos(\delta)cos(\frac{\alpha-\alpha_0}{2})}{2}}$$

$$M_0 = \frac{f_\delta sin(\delta_0)}{\sqrt{\frac{1+cos(\delta_0)}{2}}}$$

B.2 Equivalent Cylindrical projection

$$L = \alpha - \alpha_0$$

$$M = sin(\delta - \delta_0)$$

B.3 Flat projection

$$L = (\alpha - \alpha_0)$$

$$M = (\delta - \delta_0)$$

B.4 Gnomonic projection

The Gnomonic projection is common in optical astronomy.

$$L = \frac{cos(\delta)sin(\alpha - \alpha_0)}{sin(\delta)sin\delta_0 + cos(\delta)cos(\delta_0)cos(\alpha - \alpha_0)}$$

$$M = \frac{sin(\delta)cos(\delta_0) - cos(\delta)sin(\delta_0)cos(\alpha - \alpha_0)}{sin(\delta)sin\delta_0 + cos(\delta)cos(\delta_0)cos(\alpha - \alpha_0)}$$

B.5 Orthographic projection

The Orthographic projection is used by the VLA.

$$L = cos(\delta)sin(\alpha - \alpha_0)$$

$$M = sin(\delta)cos(\delta_0) - cos(\delta)sin(\delta_0)cos(\alpha - \alpha_0)$$

B.6 Rectangular projection

The Rectangular projection is used in Schmidt telecopes.

$$L = \frac{\theta}{sin(\theta)}cos(\delta)sin(\alpha - \alpha_0)$$

$$M = \frac{\theta}{sin(\theta)}\Big(sin(\delta)cos(\delta_0) - cos(\delta)sin(\delta_0)cos(\alpha - \alpha_0)\Big)$$

where

$$cos(\theta) = sin(\delta)sin(\delta_0) + cos(\delta)cos(\delta_0)cos(\alpha - \alpha_0)$$

B.7 Global Sinusoidal projection

$$L = (\alpha - \alpha_0)cos(\delta)$$

$$M = (\delta - \delta_0)$$

B.8 North Celestial Pole projection

The North Celestial Pole projection is used by the WSRT.

$$L = cos(\delta)sin(\alpha - \alpha_0)$$

$$M = \frac{cos(\delta_0) - cos(\delta)cos(\alpha - \alpha_0)}{sin(\delta_0)}$$

B.9 Stereographic projection

$$L = 2\frac{cos(\delta)sin(\alpha - \alpha_0)}{1 + sin(\delta)sin(\delta_0) + cos(\delta)cos(\delta_0)cos(\alpha - \alpha_0)}$$

$$M = 2\frac{sin(\delta)cos(\delta_0) - cos(\delta)sin(\delta_0)cos(\alpha - \alpha_0)}{1 + sin(\delta)sin(\delta_0) + cos(\delta)cos(\delta_0)cos(\alpha - \alpha_0)}$$

B.10 Mercator projection

$$L = f_\alpha(\alpha - \alpha_0)$$
$$M = f_\delta ln\Big(tan(\frac{\delta}{2} + \frac{\pi}{2})\Big) - M_0$$

where

$$f_\alpha = cos(\delta_0)$$
$$f_\delta = \frac{\Delta_\delta}{ln\Big(tan(\frac{\delta_0+\Delta_\delta}{2} + \frac{\pi}{2})\Big) - ln\Big(tan(\frac{\delta_0}{2} + \frac{\pi}{2})\Big)}$$
$$M_0 = f_\delta ln\Big(tan(\frac{\delta_0}{2} + \frac{\pi}{4})\Big)$$

Appendix C

GIPSY files

C.1 Important files

- gipsy_local.dc0
- manual.tex

C.2 Where are they?

GIPSY email address:
gipsy@[129.125.6.204]
gipsy@[129.125.6.205]

Appendix D

Host computer and operating system

At present, GIPSY runs on the following computers:

- Alliant, with the operating system Concentrix
- DEC 3100, with the operating system Ultrix
- HP9000, with the operating system HP-UX
- SUN, with the operating system SunOs

Concentrix, Ultrix, HP-UX, and SunOs all are UNIX dialects. The user need not have any knowledge about UNIX or VMS to work with GIPSY. Hermes intermediates between the user and the operating system.

Appendix E

Hardware requirements

To work with GIPSY, the following hardware devices are minimally needed:

- a host computer
- a VT100 compatible terminal, for example, a PC, or a workstation

On a VT100, the screen functions only as a Hermes window and no images or plots can be displayed.
To make hard copies of plots, a Postsript printer is required. For hard copies of, e.g., the logfile, tape listings, etcetera, an ASCII printer is required.

List of Figures

List of Tables