Contribution Opticon f2f meeting March 2007
Authors: Martin Vogelaar, Hans Terlouw

GIPSY's Parameter Interface

1. Introduction

As a contribution to the face-to-face meeting of Opticon N3.6 in March 2007, we compiled this document which describes how task parameters are handled in GIPSY, the Groningen Image Processing SYstem. A significant part of the ideas behind the parameter system originate from the seventies of the previous century and continue to be useful until this date. Nowadays scripting languages like Python are available which can facilitate the parsing and processing of parameters, but an integrated facility remains valuable.

In this document we first describe the basics of the parameter interface, giving a global view of its architecture and operation, and the interaction between the user, the user interface and the tasks. In later chapters we give a more detailed description of how the user can specify parameter values.

2. Parameter Interface Basics

Central in GIPSY's parameter handling is the user interface Hermes, which is run as a separate process. Hermes allows the user to run one or more tasks concurrently, control these tasks and supply parameters to these tasks. Every task has its private set of parameters which is stored as a collection of keyword-value pairs. The value is always stored as a text string; when the task requests the parameter, it can specify what type of parameter it expects, e.g. an array of floating point numbers or a logical value. Hermes then checks the value and if it is valid, it decodes the text string and sends binary values to the task. If the value is invalid (for the requested type), it is *'rejected'* and the user is prompted to correct the value. This is all transparent to the task: only when a correct value is available, it is allowed to proceed.

If a task requests a parameter that has not been specified yet, the user will be prompted to enter a value, or, if the task allows this, a default value will be taken. Subsequent requests for the same parameter silently causes the current value to be sent to the task - the user is not prompted. Note that this current value may have changed since the last request.
This is different when the task has *'cancelled'* the keyword. Then the user *will* be prompted for that keyword. In this way a task can obtain a sequence of values for the same keyword. Such a sequence can also be pre-specified on the Hermes command line using a special syntax or can be obtained from a text file.

The user can modify a task's parameters at any time. The change becomes effective when the task requests the parameter from Hermes, which some tasks do very frequently. Also tasks can modify parameters, their own but also the parameters of other active tasks. In this way tasks can communicate with each other.

After a task has run, the task's parameters are retained by Hermes. The user can then re-run the task with the same parameters, possibly after having edited them or modified on the command line.

Finally, there is the possiblity to specify parameter defaults for a specific task or for all tasks in a special file.

Keywords and tasks

Transferring parameters to a task is done through keywords. When a task needs information, Hermes will prompt the user on the task's behalf with the keyword and an explanatory message. A keyword is a case-insensitive character string followed by an equal sign (e.g. `INSET=`). Following the equal sign, one or more values can be given. Depending on the function of the keyword these values can be floating point numbers, integers, character strings and also expressions.

Here is an example how to start a task (`INSPECTOR`) with two keyword values (keywords `INSET=` and `BOX=`)

```
-                                                                       17:01
-                                                                          26
-
-
INSPECTOR INSET=lmap_uni.vs2.cl FREQ 20:30   BOX=0 0 D 20 20__________________
______________________________________________________________________________
```

A prompt for a parameter appears on the command line of Hermes as the name of the task followed by the parameter's keyword, together with a short description of the input and a reasonable default e.g.:

```
- GAUSS  Enter name of profile axis:       [FREQ]                       17:07
-                                                                          31
-
-
GAUSS PROFILE=_________________________________________________________________
_______________________________________________________________________________
```

The programmer defines the keywords of a task, the order of appearance and the order of processing the keyword values. A program can guide the user through the keyword list with defaults that are sensitive to the context (e.g., the data set being processed, the values of previously entered parameters, etc.)

Parameter processing in tasks with a graphical user interface, which are event-driven, is exactly the same as in tasks without a GUI. In such tasks the elements of the GUI are usually associated with a parameter keyword. Such an associated element always reflects the parameter's value. On the other hand, when the user changes the element, the parameter value is updated. When the parameter value changes, the task will receive an event. It is these events the task's application code reacts to, not the events from the GUI elements themselves. Prompting by Hermes is not possible for tasks with a GUI because this would cause the GUI to 'freeze' until the input is given.

Parameter help

Hermes can provide context-sensitive information about a task's parameter request at a single keystroke (the `TAB` key). This help is extracted from the documentation of the task. With the keystrokes `ESC TAB` the user gets a description of the keyword category with examples and some detailed request information. At the prompt '`INSET=`', typing `ESC TAB` causes the following text to be displayed:

```
        INPUT OF SET (AND SUBSETS)
        ==========================
Example: 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 RA 0 DEC 0 FREQ 1
gives a pixel at (RA, DEC, FREQ) = (0, 0, 1)

See also: INPUT SYNTAX FOR SETS [1]
-----------------------------------------------------------
Detailed request information:
Up to 512 characters. See also: text input  [2].
No default is available.
```

The bracketed numbers in this help display allow the user to navigate to other documentation about parameter input. E.g., typing `ESC 2` will show information about text input.

Parameter Types

The basic input parameter types are:

- *Numbers*: can be specified as integers, using fixed point notation or using E or D format.
- *Logicals*: can for instance be entered as `Y` or `N`.
- *Characters*: case sensitivity depends on the task's request.

Defaults

Three default levels are supported:

- *Forced* parameters are parameters for which no task default is allowed. The user must respond to the prompt with a correct value.
- *Defaulted* parameters have a default value defined by the task. If the parameter has not already been specified, then the user will be prompted. If on this occasion the user enters an empty value, this signifies that the task default is acceptable and should be used. Any other input will override the default.
- *Hidden* parameters are parameters for which the user is not prompted and which have a default value defined by the task. They can be specified by unprompted input by the user.

3. Parameter values

Numbers

Numerical parameters can be specified as numeric or symbolic constants and also as expressions. Expressions can be built from operators, numbers, constants, variables and functions.

A list of numbers can be specified as a sequence separated by spaces or using the *start*:*end*:*increment* notation, where the :*increment* part is optional and defaults to one. A list of *n* identical values can be specified as *value*::*n*.

A list can also be used as an operand in an expression or as an argument to a function. In the former case the list must be enclosed by square brackets `[ ]` or parentheses `( )`. The expression is evaluated for each element of the list.

The operator `?` can be used to select one or more items from a list.

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
[1:3]+[90:70:-10]     yields      91 82 73
[20:30]?[3 4 5]       yields      22 23 24
ranu(0,1)::10         yields      10 copies of the same random number
ranu(0,1::10)         yields      10 different random numbers
```

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

```
INCL= deg(acos(0.3 0.5))
```

The argument for `acos` is a list consisting of the elements 0.3 and 0.5. The expression evaluates these two values for `acos` and converts the resulting values to degrees. In this way the values 0.3 and 0.5 are converted to the angles 72.5424° and 60.0°.

The following list gives an overview of the elements that can be used:

```
numbers:   Numbers can be specified as integers, using fixed point
           notation or using E or D format. Internally all numbers are
           represented as double precision floating point numbers.
           e.g. 123  123.45  1234.5E-2  .12345D3

operators: The following operators are known:
+          addition              -          subtraction
*          multiplication        /          division
**         power

constants: The following constants are implemented:
pi         3.14159....           c          speed of light (SI)
h          Planck (SI)           k          Boltzmann (SI)
g          gravitation (SI)      s          Stefan-Boltzman (SI)
m          mass of sun (SI)      p          parsec (SI)
BLANK      Universal undefined value

functions: The following mathematical functions are implemented:
abs(x)     absolute value of x
sqrt(x)    square root of x
sin(x)     sine of x
asin(x)    inverse sine of x
cos(x)     cosine of x
acos(x)    inverse cosine of x
tan(x)     tangent of x
atan(x)    inverse tan of x
exp(x)     exponential of x
sinh(x)    hyperbolic sine of x
cosh(x)    hyperbolic cosine of x
tanh(x)    hyperbolic tangent of x
ln(x)      natural log of x
```

```
log(x)     log (base 10) of x
rad(x)     convert x to radians
deg(x)     convert x to degrees
erf(x)     error function of x
erfc(x)    1-error function
max(x,y)   maximum of x and y
min(x,y)   minimum of x and y
sinc(x)    sin(x)/x
atan2(x,y) inverse tan (mod 2pi) x = sin, y = cos
sign(x)    sign of x (-1,0,1)
mod(x,y)   gives remainder of x/y
int(x)     truncates to integer
nint(x)    nearest integer
ranu(x,y)  generates uniform noise between x and y
rang(x,y)  generates Gaussian noise with mean x and dispersion y
ranp(x)    generates Poisson noise with mean x

ifeq(x,y,a,b)  returns a if x equal y, else b
ifne(x,y,a,b)  returns a if x not equal y, else b
ifgt(x,y,a,b)  returns a if x greater y, else b
ifge(x,y,a,b)  returns a if x greater or equal y, else b
iflt(x,y,a,b)  returns a if x less y, else b
ifle(x,y,a,b)  returns a if x less or equal y, else b
```

Special functions have been implemented that can retrieve data from data sets (e.g. give the value of the central pixel in an image) or can get information from GIPSY descriptors (i.e. headers containing FITS keywords), ASCII files or GIPSY tables.

descr(set, name)
: descriptor item 'name' from (sub)set(s) 'set'.

table(set, tab, col, rows)
: cell(s) from column 'col' of table 'tab' in (sub)set 'set'.

image(set, box)
: pixel(s) from (sub)set 'set'.

file(name, col, rows)
: number(s) from a column in a text file 'name' in range 'rows'

The argument 'rows' has the following syntax:

```
:    use the whole file;
n:m  use line numbers n to m;
:n   use line numbers 1 to n;
n:   use line numbers n to the end of the file;
n    use only line number n.
```

Examples:

descr(aurora freq 10, noise)
: yields the value of (FITS-) descriptor item NOISE.

image(aurora freq 10, -5 -5 5 5)
: yields a list with the central 121 pixel values of set 'aurora' at FREQ=10 which is a data slice in RA-DEC at a certain position along the FREQ axis.

table(aurora p 2, ROTCUR03, VROT, 1:20)
: yields 20 values from column 'VROT' in table 'ROTCUR03' from set 'aurora' at PARAM=2.

table(aurora p 2, ROTCUR03, VROT, :)
: like previous example, but all values from the column are returned.

file(rotcur03.dat, 4, 1:20)
    yields 20 values from column 4 in ASCII file rotcur03.dat.

Expressions are evaluated whenever the task requests the parameter from Hermes. E.g., a parameter which is defined as the expression `'ranu(0,1)'` causes a different value to be sent to the task each time it is requested. Likewise a parameter which refers to an image function will have a different value when the underlying image has changed since the last request.

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 be rejected as an error.

Examples:

```
PLOTGRIDS=N
OK=YES
```

Character strings

Normally character strings are case-sensitive, but individual tasks can deviate from this or convert to either uppercase or lowercase. Some programs require an array of strings. The strings should then separated by blanks or commas. Sometimes spaces however can be part of the string. Then the input is called 'text'.

Examples:

```
INSTRUME=WSRT                      (character string)
COMMENT=Set smoothed on 2-2-92     (text)
```

4. Data Sets

Image data in GIPSY is stored in so called *sets*. A set consists of two components: an *image* component and a *descriptor* component. These components are stored in separate files, but in GIPSY they are always used together.

The image data is stored in a 'crystalline' structure with a virtually unlimited (up to 20) number of dimensions. Image data is only referenced using coordinates on symbolic axes. File coordinates are hidden from the user.

Within a set subsets (or *slices*) can be defined. A subset is a part of a set with a dimensionality not greater than the dimensionality of the set. So in a three-dimensional set ('*cube*') the following subset types can be identified:

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

Associated with the image data there exists a hierarchical system of descriptor (or 'header') items. Headers can be associated with any of all possible subsets, ranging from the whole data set down to the pixel level.

Many tasks operate only on a part of a subset. Such a part is called a *frame*.

A typical GIPSY set with radio data might be a 3-dimensional 'cube' named AURORA, with axes `RA`, `DEC` and `FREQ`. Pixel, or grid, coordinates in this set can be seen as tuples: (RA,DEC,FREQ). This set AURORA is used as an example in some of the next sections.

Specifying sets and subsets

When a task requires a set or subset specification, it depends on the operations defined in that program, what subset types are allowed. A subset can be specified by giving the axis name(s) and grids for each subset. For instance,

```
AURORA
```

specifies the whole 3-dimensional 'cube';

```
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: AURORA FREQ 1:4 8)
```

specifies 5 subsets: 5 RA-DEC planes in AURORA;

```
AURORA DEC 5:10 FREQ 10
```

specifies 1-dimensional subsets ('lines') for values of DEC from 5 through 10 at FREQ 10. Specifying only the axis name, and no grids, selects *all* subsets along that axis:

```
AURORA FREQ
```

selects all RA-DEC planes in AURORA.

And finally the extreme example '`AURORA RA DEC FREQ`' specifies as many subsets as there are pixels in the set AURORA.

Specifying positions in a (sub)set

Many tasks require one or more positions in a (sub)set. Positions can for instance be used:

- to set projection centers for tasks which perform reprojections
- to identify the position of a profile in a data set
- to mark positions in a plot
- to identify features in the data

Usually a task prompts the user to enter positions with the keyword `POS=`. The task should also indicate which type of coordinates must be specified and which ranges (in pixels) are allowed. In practice this is not always the case, because creating such messages may not be trivial.

The most common position is a *grid position* in a map, e.g.:

```
POS= 3 5
```

It is also possible to enter a number in the physical units corresponding to a subset axis. Then the number should be followed by valid units, e.g.

```
POS= 1418.6 MHz
```

which is equivalent to:

```
POS= 1.4186 GHz
```

Physical units that GIPSY recognizes are given in the table below.

DEGREE	ARCSEC	ARCMIN	RADIAN
CIRCLE	DMSSEC	DMSMIN	DMSDEG
HMSSEC	HMSMIN	HMSHOUR	
METER	ANGSTROM	NM	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			

For spatial positions (images with a sky system) more formats are possible. For example in the `AURORA FREQ 10` subset, RA and DEC can be specified as follows:

```
POS=* 14 26 9 * -12 3 5.4
```

For n-dimensional subsets an n-tuple coordinate must be specified. For instance `POS=3 5 10` specifies the same grid point as above but now for the 'unsliced' 3-dimensional set AURORA.

There are two symbols that denote a special spatial position:

`PC`
: projection centre, effectively grid position (0,0,...,0).

`AC`
: axis centre. If the length of axis *i* is `NAXIS`*i* and the reference pixel is `CRPIX`*i*, then the *i*-th coordinate is given by the expression `NAXIS`*i* / 2 - `CRPIX`*i*.

Coordinates can also be specified in units that correspond to the FITS keywords CUNIT*n*. We have to prefix them with character 'U' as in the next example:

```
COORDS POS=U 2.8125 U 35.70
```

If the corresponding CUNITs are in degrees, then this position is processed as degrees. When specifying spatial coordinates, the following prefixes can be used:

```
*        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
```

Examples:

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

RA = 10 hours, 12 minutes, 8 seconds, DEC = -67 degrees, 8 minutes, 9.6 seconds, in a 2-dimensional area and in the epoch as found in the desciptor of the set.

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

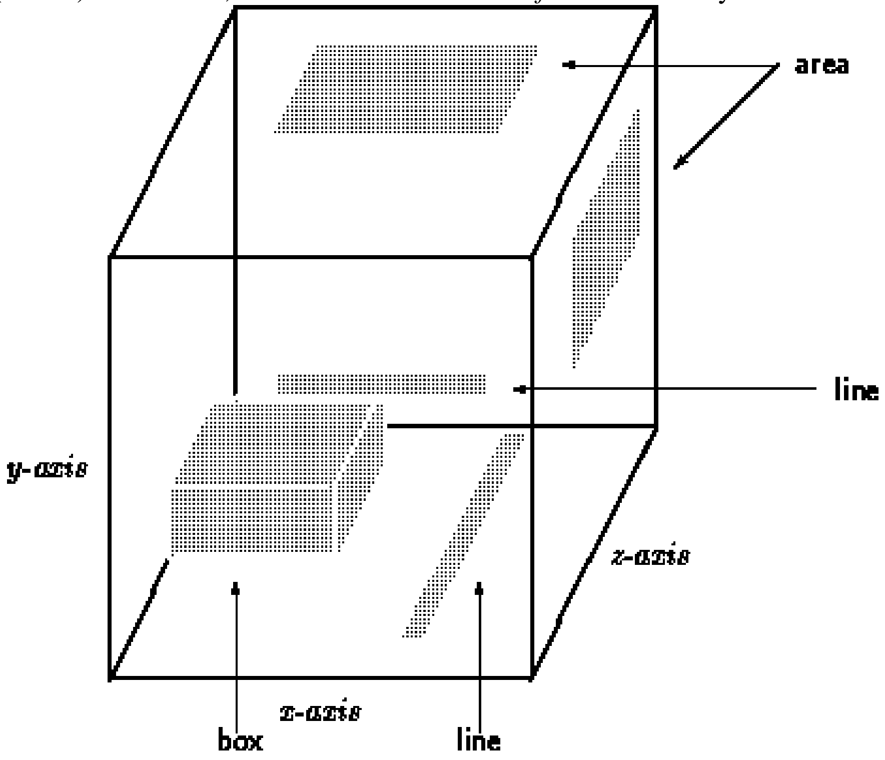
Input of RA 3 h 14 m 38.02 s, DEC 41 d 13 m 54.84 s in epoch 2000.0

Specifying frames

Many tasks operate on specified *parts* of (sub)sets. Examples of subsets for which this can be useful are:

- The user wants to fit Gaussian parameters to the data of all velocity profiles in a WSRT radio set with channel maps. With some knowledge of the data, channels with only noise can be excluded.
- The user wants only a part of the data in a set to speed up calculations, e.g. in sets with blotted data.

More generally, tasks apply operations to user-defined n-dimensional section(s) of an m-dimensional data set ($n \leq m$). In GIPSY, such sections are called *frames* and they are associated with the `BOX=`



keyword.

Similar to the `POS=` keyword, the `BOX=` keyword is accompanied by a message for the user indicating along which axes he or she needs to specify the limits:

```
Give BOX in RA,DEC:         [entire subset]
BOX=
```

A box is specified with *two* positions. For example in a 2-dimensional image we need one for the lower left, and one for the upper right corner of the frame. These positions can be specified in the same formats as used for the `POS=` keyword (see above). For example,

```
BOX= -3 -3 4 4
```

specifies a 2-dimensional box containing 64 pixels.

Boxes can also be specified using a position-and-size notation:

```
center D size
```

e.g.

```
BOX= PC D 8 8
```

(The `D` stands for *Delta*.) If only the size is given, the user will be prompted to supply the center position.
Position and/or sizes can also be specified in physical coordinates. For example:

```
BOX= * 3 0 0 * 45 0 0 D 6 arcmin 6 arcmin
```

to get a box centered at 3h0m0s and 45d0m0s with sizes in RA and Dec of 6 minutes of arc. The units of the sizes must be compatible with the axis units in the header of the set. or:

```
BOX= G 56.7 G 9.89 D 12 ARCSEC 10 ARCSEC
```

This is the input for a box centered at 56.7° galactic longitude and 9.89° latitude with size 12×10 seconds of arc.

February 21, 2007

Original URL: `http://www.astro.rug.nl/~gipsy/general/taskparameters.html`