IRAM-COMP-048

Revision: 0
07 JAN 2008

Contact Author

Institut de RadioAstronomie Millimétrique

PdB New Generation
Interferometer Control Software

Owner Alain Perrigouard (perrigou@iram.fr)

Keywords:

Approved by:	Date:	Signature:
A.Perrigouard	January 2008	

Change Record

REVISION	DATE	AUTHOR	AFFECTED SECTION/PAGE	REMARKS

Content

1 Introduction

The interferometer software should be compiled and then copied (installed) to its target directory from sessions opened on bure4b/5b that should be accessible from all the developers. In principle bure3b is more reserved for the observations and to the operators (account oper).
The target should be **`/control`** mounted as well on bure1b and bure3b/4b/5b.
The developers referenced with their UID/GID must have a home directory on **`/users`** to be allowed to login on bure3b e.g. **`/users/softs/gildas`** or **`/users/computer/perrigou`**. If someone needs to develop under a private account, ask for the corresponding directory creation. Furthermore, to add sub-directories to **`/control`** and to access all files in **`/control`**, the developers need to belong also to the group interf and to have switched to this GID. In other words one needs to execute the shell command "newgrp interf" before accessing /control. If a developer doesn't change its default GID to interf, I may create directories and files in /control but the files will be created with its original GID, letting them inaccessible from the other developers.

Here after an example for the computer group:
The software written to control and monitor the antennas and the receivers of the interferometer of Plateau de Bure can be down loaded, edited, updated and compiled in any working directory.
For instance on bure3b, to download the sources:

```
bure3b:~ $ mkdir devel
bure3b:~ $ cd devel
bure3b:~/devel $ cvs co -r FC6-branch LINUX
```

To build everything, i.e. to compile and link all executables and to generate intermediate files:

```
bure3b:~/devel/LINUX/bure1 $ make
```

By intermediate files one means the direct access binary files used in astrj, the astrometry package originally written by J.Delannoy, and the include files needed to describe the shared memory areas and which are used by the programs written in Fortran 95.
To install the executables, the binary files, the include files and all the script files, etc..., a "make install" has to be issued under root, with the environment variable INSTROOT set to /control if we wish to keep the structure similar to the one we had on the HP version of bure1.

```
bure3b :~/devel/LINUX/bure1 $ newgrp interf
bure3b :~/devel/LINUX/bure1 $ make install
```

In **`bure3b:/control`** the directories **`antenna/, astro/, command/, common/`** and **`receiver/`** are created and their contents are copied. The scripts clean and install are also copied into /control.
However, in the created directories a new level of sub-directories are created which may be bin/, obj/, data/ or include/ depending on the needs. For instance the files **`satd07.bin, tche0615.bin`** and **`vsp86.dat`** are found in **`control/astro/data/`**.
For extra development and in particular for **`obs`** and **`rdi`**, the files copied with the above "make install" to bure3b:/control should be necessary and sufficient for making (compilation and link) those packages (**`obs`** and **`rdi`**).

2 Addressing

```
iramr2b  192.168.10.1

bureACS  192.168.10.5
bureKVM  192.168.10.6

netapp1b 192.168.10.11
netapp2b 192.168.10.12
```

```
bure1  195.83.131.3     bure1b   192.168.10.51
bure2  195.83.131.4     bure2b   192.168.10.52

bure3  195.83.131.5     bure3b   192.168.10.53
bure4  195.83.131.6     bure4b   192.168.10.54
bure5  195.83.131.7     bure5b   192.168.10.55

bure-nfs-master 195.83.131.68 x86nfsrw 192.168.10.98
ifproc 195.83.131.69    ifprocb  192.168.10.99
clock  195.83.131.70    clockb   192.168.10.100

ant11  195.83.131.71    ant11b   192.168.10.101
ant12  195.83.131.72    ant12b   192.168.10.102
ant21  195.83.131.73    ant21b   192.168.10.103
ant22  195.83.131.74    ant22b   192.168.10.104
ant31  195.83.131.75    ant31b   192.168.10.105
ant32  195.83.131.76    ant32b   192.168.10.106
ant41  195.83.131.77    ant41b   192.168.10.107
ant42  195.83.131.78    ant42b   192.168.10.108
ant51  195.83.131.79    ant51b   192.168.10.109
ant52  195.83.131.80    ant52b   192.168.10.110
ant61  195.83.131.81    ant61b   192.168.10.111
ant62  195.83.131.82    ant62b   192.168.10.112

xcorl1 195.83.131.151      xcorl1b  192.168.10.151
xcorl2 195.83.131.152      xcorl2b  192.168.10.152
xcorl3 195.83.131.153      xcorl3b  192.168.10.153
xcorl4 195.83.131.154      xcorl4b  192.168.10.154
xcorl5 195.83.131.155      xcorl5b  192.168.10.155
xcorl6 195.83.131.156      xcorl6b  192.168.10.156
xcorl7 195.83.131.157      xcorl7b  192.168.10.157
xcorl8 195.83.131.158      xcorl8b  192.168.10.158

phaser 195.83.131.130      phaserb  192.168.10.171
```

Deprecated:
The names ending with b will stay valid until we definitively switch to this new architecture. We cannot have a processor with the same name and 2 different IP addresses (e.g. ant11 195.83.131.71 and 192.168.10.101) and we cannot have 2 machines with the same name (e.g. bure1 HP Risk workstation-HPUX and PC 64 bits - LINUX FC6).
As soon as the switch to the new architecture will be decided, we will be able to move to the final names to finalize the scripts and the code which may be name dependent.
It will be useful to keep some machines on-line like for instance the current bure1 and bure2. They will be renamed bure1o and bure2o with the same address in the network 195.83.131.

At Bure there is also a private network dhcp 192.168.3.* (dhcp-pdb*). This network will later be moved to 192.168.11.0/24
We keep in mind to move the personal PCs to a private network 192.168.12.0/24.
The public networks 195.83.131.1/25 and 195.83.131.128/26 will be kept for special cases.
Vmware is installed on bure2b to host a virtual machine with R/W access to the FS distributed to the SBCs VMIC. The machine is called x86nfsrw.

3 Semaphores

As before, we use semaphores to synchronize interprocess communication or process execution.
common/src/sem_routines.c is a compilation of the functions to handle the semaphores.
The function sem_create() creates a set of 64 semaphores. The set is identified by the key "BURE".
Once created, the shell command ipcs shows information about the activated semaphore set. For instance:
bure1b:~ $ ipcs -s

```
------ Semaphore Arrays --------
key        semid      owner      perms      nsems
0x45525542 851968     root      666        64
```

key, 0x45525542, is the string "BURE" in hexadecimal, in reverse order.
perms are read and write for owner(root), group and others). write permission means permission to alter semaphore values).

And to remove the semaphore array:

bure1b:~ $ ipcrm -s 851968
to remove the semaphore array identified with semid=851968, or

bure1b:~ $ ipcrm -S 0x45525542 to remove semaphore array "BURE"

The list of the semaphore utility functions present in sem_routine.c:

- sem_create() : creation of a semphore id and the associated stucture
- sem_init() : set to 1 all semaphores
- sem_wait(semaphore_number) : wait until the semaphore = 0
- sem_clr(semaphore_number) : set to 0 the semaphore
- sem_set(semaphore_number) : set to 1 the semaphore
- sem_read(semaphore_number) : read the semaphore value
- shm_lock(semaphore_number) : lock the resource associated to the semaphore
- shm_unlock(semaphore_number) : unlock the resource associated to the semaphore
- setef(semaphore_number) : equivalent to sys$setef (=sem_clr)
- clref(semaphore_number) : equivalent to sys$clref (=sem_set)
- waitfr(semaphore_number) : equivalent to sys$waitfr (=sem_wait)
- readef(semaphore_number) : equivalent to sys$readef (=sem_read)

The function sem_create() has to be called first in all programs dealing with the semaphores. It returns a semaphore set identifier for a set of 64 semaphores defined for the key "BURE" and with the permissions R/W for the set owner, group or others. If the set does not yet exist, it is created with the process owner as the set owner.

shm_lock() and shm_unlock() are intended to be used for accessing shared memory areas in a safe way. They locks and unlocks the memory area associated to the semaphore number given as parameter. flg_s_ant is for instance the semaphore associated to the area identified with the key "ANTE". flg_s_ant is equal to 0 (see sem.h). (Sometimes and historically the name flag is used for semaphore)
The use of the functions sem_wait(), sem_clr() and sem_set() would not be full proved for accessing a shared memory area.

The functions setef(), clref(), waitfr() and readef() which recall the VAX/VMS functions may be used for old code written originally for VMS.
The include files sem.h and sem.f define the assigned semaphore numbers. They are copied to /control/common/include/

/control/common/include/sem.h:

```
#define flg_s_ant 0
#define flg_astro 1
#define flg_tcpip 2
...
```

/control/common/include/sem.f:

```
       integer*4 flg_s_ant         ! locks antenna data
       integer*4 flg_astro         ! trigs astro
       integer*4 flg_tcpip         ! synchronizes interp
 ...
       parameter (flg_s_ant = 0)
       parameter (flg_astro = 1)
       parameter (flg_tcpip = 2)
```

```
  ...
```

Example of program in C (exa.c):

```
#include <stdlib.h>
#include "sem.h"
int main(int argc,char* argv[])
{
  int sem_tcpip = flg_tcpip, sem_s_ant = flg_s_ant;

  sem_create();
  shm_lock(&sem_s_ant);
  //  sub();
  shm_unlock(&sem_s_ant);
  sem_clr(&sem_tcpip);
  exit(0);
}
```

Example of program in Fortran (exb.f):

```
      include 'sem.f'

      call sem_create
 10   call sem_wait(flg_astro)
      call sem_set(flg_astro)
!      call sub
      go to 10

      end
```

Example of makefile (~/devel/LINUX/bure1/common/Makefile):

```
BINDIR=bin
SRCDIR=src
OBJDIR=obj

COM = .
COMOBJ = $(COM)/obj
SEMOBJ = $(COMOBJ)/sem_routines.o

$(OBJDIR)/%.o: $(SRCDIR)/%.c
   $(CC) -c $(CFLAGS) -MMD $< -o $@

$(OBJDIR)/%.o: $(SRCDIR)/%.f
   gfortran -c $(FFLAGS) $< -o $@

$(BINDIR)/exa: $(OBJDIR)/exa.o $(SEMOBJ)
   $(CC) $(CFLAGS) $^ -o $@

$(BINDIR)/exb: $(OBJDIR)/exb.o $(SEMOBJ)
   gfortran $^ -o $@

INCLUDES= -I/usr/include -Iinclude
CFLAGS = -Wall -g $(INCLUDES)
FFLAGS = $(INCLUDES) -fno-underscoring
```

3.1 utility program

There is the program **sem** copied to **/control/common/bin** which can be useful for setting, clearing and reading the semaphores. Its execution without any parameter shows its usage.

```
bure1b :~ $ /control/common/bin/sem
Usage:
sem init
sem clear <semaphore_number>
sem set <semaphore_number>
sem read <semaphore_number>

bure1b :~ $ /control/common/bin/sem read 3
Read
semaphore 3 : 1
```

4 The functions to include in obs

4.1 Functions to control the antenna mounts:

write_coo(int* iant_p, int* itel_p, Coo_t* coo_p,
    General_t* general_p, Antenna_t* antenna_p)
write_dri(int* iant_p, int* itel_p, Off_t* off_p,
    General_t* general_p, Antenna_t* antenna_p)
write_off(int* iant_p, int* itel_p, int* del_p, Off_t off_p,
    General_t* general_p, Antenna_t* antenna_p)
write_pla(int* iant_p, int* itel_p, Coo_t* coo_p,
    General_t* general_p, Antenna_t* antenna_p)
write_point(int* iant_p, Point_t* point_p, Antenna_t* antenna_p)
write_sec(int* iant_p, Subref_t* subref_p, Antenna_t* antenna_p)

The new types are defined in /control/common/include/general.h, general.f, antenna.h and antenna.f
If *itel_p is not null, the parameters are applied to all the antennas of the telescope *itel, a number smaller or equal to 6. That means for all antennas iant=anttel[*itel-1][i] for i from 0 to IMAX-1(=5) with anttel being a table set in the shared memory area "GENE" itself defined with the struct general_s declared in general.h.
If *itel_p is null, the parameters are applied to the antenna *iant_p, a number from 1 to IMAX (=6).

4.2 Functions to control and monitor the SIS and WVR receiver

// Any function returns 0 when completed successfully.
// The function returns a value != 0 when an error occurs during its execution or when a parameter is out of range

```
typedef enum {
   calModeInvalid,  /* use by external task status to describe an
invalid position */
   calModeObserving,
   calModeHotLoad,
   calModeColdLoad,
   calModeVlbi,
   calModeMaxIndex
}
```

```
ReceiverCalibrationMode_t;
```

// bandNum from 1 to 4. Any function with a wrong bandNum return 1.

// This function returns when the requested calibration mode is set
// The mechanical operation may last a moment

```
int set_calibration_mode(
          int* iant,
          int* itel,
          int* bandNum,
          ReceiverCalibrationMode_t* calibrationMode,
          General_t* general_p,
          CabinReceivers_t* sisStatus_p)

typedef enum {
   polarV,
   polarH,
   polarMaxIndex,
} ReceiverPolar_t;
```

// attenuation from 0. to 20dB. Any other values are truncated

```
int set_attenuation(
          int* iant,
          int* itel,
          int* bandNum,
          ReceiverPolar_t* polar,
          float* attenuation,
          General_t* general_p,
          CabinReceivers_t* sisStatus_p);


int set_attenuationHV(int* iant,
            int* bandNum,
            float* attenuationH,
            float* attenuationV,
            CabinReceivers_t* sisStatus_p);
```

// This function returns when the requested calibration mode is set
// The mechanical operation may last a moment

```
int set_calibration_attenuation(
            int* iant,
            int* itel,
            int* bandNum,
            ReceiverPolar_t* polar,
            float* attenuation,
            ReceiverCalibrationMode_t* calibrationMode,
            General_t* general_p,
            CabinReceivers_t* sisStatus_p);

typedef enum {
   sidebandLower,
   sidebandUpper,
   sidebandMaxIndex
}
ReceiverSideband_t;

typedef enum {
   deltaFMinus    ,
   deltaFPlus     ,
   deltaFMaxIndex
}
```

```
ReceiverDeltaF_t;

int set_frequency(
        int* iant,
        int* itel,
        int* bandNum,
        float* frequency,
        ReceiverSideband_t* sideband,
        ReceiverDeltaF_t* deltaF,
        General_t* general_p,
        CabinReceivers_t* sisStatus_p);

typedef enum {
   rlo2A        ,
   rlo2B        ,
   rlo2Both    ,
   rlo2MaxIndex
}
ReceiverRlo2Mode_t;
```

// This function returns when the LO2(s) is(are) reset. It may last a moment

```
int reset_lo1_ref(
        int* iant,
        int* itel,
        ReceiverRlo2Mode_t* rLo1RefMode,
        General_t* general_p,
        CabinReceivers_t* sisStatus_p);

int switch_to_direct_lo1_ref(
              int* iant,
              int* itel,
              General_t* general_p,
              CabinReceivers_t* sisStatus_p);

int switch_to_crossed_lo1_ref(
               int* iant,
               int* itel,
               General_t* general_p,
               CabinReceivers_t* sisStatus_p);

int beginObserving(
         int* iant,
         int* itel,
         General_t* general_p,
         CabinReceivers_t* sisStatus_p);

void write_wvr(
   int* p_iant,
   int* p_itel,
   WvrRequest_t* request_p,
   struct s_general* p_general,
   CabinReceivers_t* cabinReceivers_p);
```

4.3 Practical considerations

All the object files corresponding to these functions are copied to /control/command/obj

They are commands which call these functions and which can be directly executed from a shell are copied into /control/command/bin. They are:
coo, dri, off, pla, put (corresponding to write_point()), **sec, calibrationMode, attenuation, attenuationHV, calibrationAttenuation, frequency, rLo1Ref, directLo1Ref, crossedLo1Ref, observing** and **wvr**.

5 Shared Memory Areas

The header files general.h, antenna.h and receiver.h describe the structure of the shared memory areas defined with the keys "GENE", "ANTE" and "RECE".
These files are copied into /control/common/include/ and they must be included in the C programs dealing with the shared memory areas.

The description files for the program written in Fortran 95 are general.f, antenna.f and receiver.f. The .f include files are generated from the .h header files with the utility c2f.

For instance to generate antenna.f in ~/devel/PdB/LINUX/bure1/common:

```
bure1b:~/devel/PdB/LINUX/bure1/common $ make include/antenna.f
bin/c2f include/antenna.h > include/antenna.f
```

and for receiver.h:

```
bure1b perrigou:~/devel/PdB/LINUX/bure1/common $ make
include/receiver.f
cpp include/receiver.h include/receiver.hh -I
.../.../cabin/receiver/include/
bin/c2f include/receiver.hh > include/receiver.f
```

(The C preprocessor cpp is required to merge the internal header files).

Once the memory spaces are created, the shell command ipc shows information about the activated shared memory segments:

```
bure1b :~ $ ipcs -m
------ Shared Memory Segments --------
key        shmid      owner     perms     bytes     nattch    status
0x454e4547 983040     root      666       520       2
0x45544e41 1015809    root      666       5040      2
0x45434552 1048578    root      666       6864      1
```

"GENE" => 0x454e4547
"ANTE" => 0x45544e41
"RECE" => 0x45434552

And to remove the segments, for instance:

bure1b :~$ ipcrm -m 983040 to remove identified shared memory "GENE")
bure1b :~$ ipcrm -M 0x45544e41 to remove shared memory "ANTE"

In C the function shm_connect() returns a pointer to the shared memory area identified by its key. For instance

```
#include "general.h"
general_p = (General_t *)shm_connect("GENE", sizeof(General_t));
```

If the memory area doesn't yet exist when shm_connect() is executed, the function allocates enough memory space according to the size indicated as the 2nd input parameter.

In Fortran the subroutine shm_connect_() is called. For instance

```
include 'test2.f'
type(b_s) :: b
common / test2 / b
call shm_connect_('TEST', b, stat)
```

In this example 'TEST' is the key assigned to the shared memory area, the 2nd argument is the single variable of the common block and the 3rd variable is a status which is equal to 0 when the connection is alright.
The variable of the common block is of the derived type b_s i.e. a structure.
A derived-type object has no storage association. In order to access to all the elements of a derived type, the derived type definition must contains a sequence statement making it a sequence type (see below the definition of the derived type b_s in test2.f).
The common block test2 is mapped to an arbitrary address at the link operation time.
As the size of the shared memory area is considered only in the C version of shm_connect (2^{nd} argument of the function shm_connect()) the first reference of a shared memory area, at execution time, should be in a program written in C to allocate enough memory space (before any other program referencing the same memory area).

Example of program in C (test2c.c):

```
#include <stdio.h>
#include <string.h>
#include <stdlib.h>

#include "test2.h"
#include "shm.h"

int main(int argc,char* argv[])
{
  B_t* b_p;

  int val;
  float fval;

  b_p = (B_t*)shm_connect("TEST", sizeof(B_t));

  printf("b_p->ba %d\n", b_p->ba);
  printf("b_p->bb %f %d\n", b_p->bb.aa, b_p->bb.ab);

  if (argc > 1)
    {
      sscanf(argv[1], "%d", &val);
      printf("val %d\n", val);
      b_p->ba = val;
    }

  if (argc > 2)
    {
      sscanf(argv[2], "%f", &fval);
      printf("val %f\n", fval);
      b_p->bb.aa = fval;
    }

  if (argc > 3)
    {
      sscanf(argv[3], "%d", &val);
      printf("val %d\n", val);
```

```
      b_p->bb.ab = val;
    }

  exit(0);
}
```

Example of program in Fortran (test2f.f):

```
      program test2f

      include 'test2.f'

      type(b_s) :: b

      integer*4 addr, stat

      common / test2 / b

      call shm_connect_('TEST', b, stat)
      if (stat.eq.0) then
         print*, 'shared memory ',  b%ba, b%bb%aa, b%bb%ab
      end if

      end
```

test2.h:

```
struct a_s {
  float aa;
  int ab;
};
typedef struct a_s A_t;

struct b_s {
  int ba;
  struct a_s bb;
  float bc[2];
  double bd[3][2];
};
typedef struct b_s B_t;
```

test2.f ($ bin/c2f test2.h >test2.f):

```
      type a_s
      sequence
      real*4 aa
      integer*4 ab
      end type a_s
      type b_s
      sequence
      integer*4 ba
      type( a_s ) bb
      real*4 bc(2)
      real*8 bd(3,2)
      end type b_s
```

and the makefile (~/devel/LINUX/bure1/common/Makefile):

```
BINDIR=bin
SRCDIR=src
OBJDIR=obj
```

```
COM = .
COMOBJ = $(COM)/obj
SEMOBJ = $(COMOBJ)/sem_routines.o
SHMOBJ = $(COMOBJ)/shm_connect.o

files = size c2f sem test2c exa

binfiles = $(files:%=$(BINDIR)/%)

sources = $(files:%=%.c) scanner.c analyse.c sem_routine.c

incfiles = general.f antenna.f receiver.f test2.f

# Pattern rules

$(OBJDIR)/%.o: $(SRCDIR)/%.c
   $(CC) -c $(CFLAGS) -MMD $< -o $@

$(OBJDIR)/%.o: $(SRCDIR)/%.f
   gfortran -c $(FFLAGS) $< -o $@

$(BINDIR)/test2c: $(OBJDIR)/test2c.o $(SHMOBJ)
   $(CC) $(CFLAGS) $(LIBS) $^ -o $@

$(BINDIR)/test2f: $(OBJDIR)/test2f.o $(SHMOBJ)
   gfortran $^ -o $@ -Wl,--defsym -Wl,test2=0x40000000

INCLUDES= -I/usr/include -Iinclude
CFLAGS = -Wall -g $(INCLUDES)
FFLAGS = $(INCLUDES) -fno-underscoring
```

5.1 c2f

This utility program translates a structure declared with a struct instruction in a C header file to a derived type in a Fortran file which later may be correctly interpreted in Fortran 95.
c2f consists of a scanner, a syntax analyzer and finally a translator.
The grammar of the analyzer is the following:

```
  {declaration_list,
     declaration, scolon, declaration_list, or,
     declaration, scolon, or, end},
  {declaration,
     class_specifier, or,
     struct_specifier, or, end},
  {class_specifier,
     TYPEDEF, type_specifier, identifier, or, end},
  {struct_specifier,
     STRUCT, identifier, lbrace, struct_declaration_list, rbrace, or,
     STRUCT, lbrace, struct_declaration_list, rbrace, or,
     STRUCT, identifier, or, end},
  {struct_declaration_list,
     struct_declaration, struct_declaration_list, or,
     struct_declaration, or, end},
  {struct_declaration,
     type_specifier, declarator_list, scolon, or, end},
  {declarator_list,
     declarator, comma, declarator_list, or,
```

```
    declarator, or, end},
  {declarator,
    identifier, brackets, or,
    identifier, or, end},
  {brackets,
    lsbracket, constant, rsbracket, brackets, or,
    lsbracket, constant, rsbracket, or, end},
  {type_specifier,
    INT, or,
    LONG, or,
    FLOAT, or,
    DOUBLE, or,
    identifier, or,
    struct_specifier, or, end}
```

It is based on the Kernighan and Ritchie C language grammar given in appendix in their book. The definition of declarator is slightly modified to allow the implementation of a top-down parser (analyzer) which accepts only right recursivity.
(left recursivity: number = number digit | digit
right recursivity:number = digit number | digit)
The current translation corresponds to the features of the Fortran 95.
Example:

```
c2f test.h >test.f

/* file test.h
*/
typedef int logical;
struct aa {                                  type aa
                                             sequence
  int b, c;                                  integer*4 b, c
  double e[4];                               real*8 e(4)
  float f[3][5];            ==>              real*4 f(3,5)
  long g;                                    integer*8 g
  logical d;                                 logical*4 d
};                                           end type aa
struct bb {                                  type bb
                                             sequence
  int h;                                     integer*4 h
  struct aa i,j;                             type( aa ) i, j
};                                           end type bb
```

6 Init and background tasks

The script **`bure1b:/control/install`** creates and initializes the shared memory areas, the semaphores and then starts the setup and periodic/background tasks.

6.1 Antenna (`/control/antenna/bin`)

`init` Creates and initializes the shared memory segment “GENE” and “ANTE”. The command “ipcs –m” shows the shared memory segment keys: 0x454e4547 for “GENE” and ox45544e41 for “ANTE”.

`interp` Creates the semaphore set “BURE” and then exchange data with the micros ant*1b. The transfers are triggered by the semaphore sem_tcpip (2) every second.

flag1s Receives each second a message from clockb. At the message reception the semaphores sem_tcpip(2), sem_ut(6) and sem_ut22GHz (19) are cleared. The message also contains the UT time in seconds from midnight of the time bus pulse at the origin of the transmission.

6.2 Astro (/control/astro/bin/)

astrj the task prepares the coordinate transformations and provides the sun equatorial position each time a new source or planet is requested.

6.3 Receiver (/control/receiver/bin/)

statusReceivers collects periodically (1s) essential receiver data and copies them in the shared memory area "RECE". The command /control/receiver/bin/dumpReceivers dumps these data.

interp22GHz controls, monitors and downloads data periodically (1s) from the water vapor receivers. Status and data are also available with the command dumpReceivers.

7 Common (/control/common/bin/)

sem is a simple utility program to display, clear and set semaphores.

8 commands

The main commands are stored in **bure1b:/control/command/bin**.
The commands attenuation, attenuationHV, calibrationAttenuation, calibrationMode, coo, crossedLo1Ref , directLo1Ref, dmp, dri, frequency, io, observing, off, offInc, pla, put, ref, rLo1Ref, sec, sto, tel and wvr display their usage when they are entered without or wrong number of arguments.

- **attenuation** set the receiver attenuations
- **calibrationHV** set the receiver horizontal or vertical attenuations
- **calibrationAttenuation** set the receiver calibration modes and attenuations
- **calibrationMode** set the receiver calibration modes
- **coo** request to move to a source
- **crossedLo1Ref** switch to crossed Lo1ref
- **directLo1Ref** switch to direct Lo1ref
- **dmp** dump shared memory area(s)
- **dri** request a drift
- **frequency** set the 1st LO frequency
- **io** set or read antenna(s) input/output bits
- **observing** to switch from stand-by to observation mode or vice versa
- **off** request antenna offsets
- **offInc** reads IAZ and IEL from ~oper/pdbi-data/base/general.an<antennaID> and executes on the specified pedestal micro the local command offInc for changing both axis, absolute and incremental encoder offsets
- **pla** request to move to a planet
- **put** set the pointing parameters
- **ref** set refraction parameters
- **rLo1Ref** reset the Lo1Ref's
- **sec** to change antenna subreflector offsets
- **set_sun** sends the sun coordinates to all antennas
- **sto** stop one or several antennas
- **tel** set the antennas members of an interferometer pseudo telescope

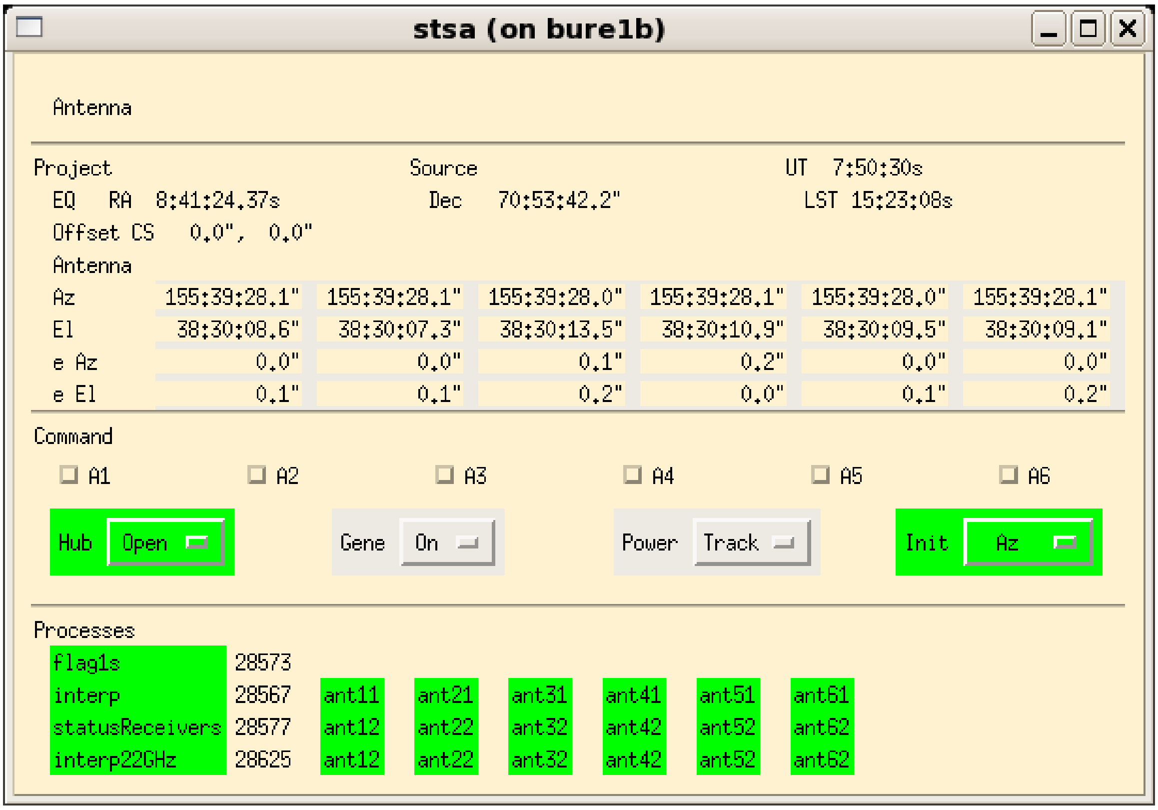
wvr set the water vapor receiver parameters

8.1 Meteo

meteo this task sends the meteo data found in the shared memory area “GENE” every 10s to the system logger. The log facility LOG_LOCAL5 and the priority LOG_INFO are the parameters. In the syslog configuration file /etc/syslog.conf it is mentioned to remote log this logging stream to websrv3 and iralx5. The task also copies the meteo data every 5 minutes to a file, a new file each day ~oper/Meteo/Data/<dd-mmm-yy>.met.

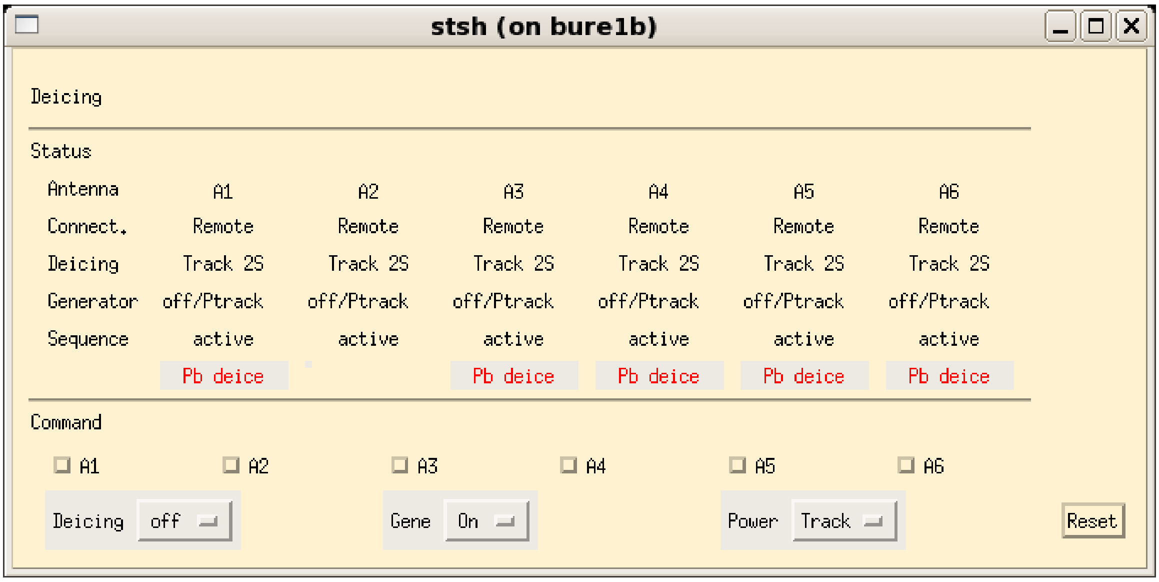
9 Graphic User interfaces

9.1 Stsa



The bottom part of the gui display the status of the main tasks.
When a task is running, its process ID is displayed.
These tasks should loop forever at 1Hz. A red background color of the task name indicates a pending or waiting task, not looping anymore. Orange indicates a running task but with one or more antenna connection interrupted. A green background indicates a nominal behaviour: Running task connected to all antennas members of the pseudo telescope.

9.2 Stsh



9.3 stsm