Strumenti software per ARNICA al TNG

E.Giani[1], C.Baffa[1]

[1]Osservatorio Astrofisico di Arcetri

Arcetri Technical Report N° 1/99
Firenze 1999

Abstract

Il Tirgo ed il gruppo di sviluppo strumentazione del Telescopio Nazionale Galileo (TNG) hanno concordato che la camera ARNICA sia portata al TNG fino all'installazione della camera IR del TNG, NICS. Il presente rapporto contiene una descrizione dettagliata degli strumenti software necessari a tale scopo. In particolare descrive il sorgente dell' ancillary process che consente la comunicazione tra il software di controllo XNIR della camera ARNICA ed il software installato sulle workstations appartenti al sistema del Telescopio Galileo.

1 Introduzione

Durante il mese di Dicembre 1998 la camera infrarossa ARNICA è stata trasportata al Telescopio Nazionale Galileo.
Lo strumento infrarosso ed il software che ne gestisce il funzionamento, come già descritto in un precedente rapporto interno ([1]) non sono conformi allo standard di sviluppo definito dai responsabili del progetto Galileo ([2] e [3]), per cui è stato necessario sviluppare uno specifico modulo software, nella nomenclatura TNG un ancillary process (AP) da integrare al software WSS.
Questo programma consente una forma elementare di comunicazione tra il sottosistema costituito da ARNICA ed XNIR da una parte ed una workstation appartenente al sistema TNG dall' altra sulla quale è eseguito il software WSS.
La comunicazione tra le due parti è ottenuta attraverso linea seriale RS-232 e permette lo scambio di alcune informazioni indispensabili per il corretto svolgimento dell' osservazione e la successiva analisi dei dati astronomici acquisiti.

2 Il programma ARN

Il programma relativo al processo ancillary per ARNICA è stato scritto in C ed è stata utilizzata la libreria TDBL ([4]).
Il codice dell' AP è costituito da un file sorgente *arn.c* ed un file header *arn.h*.

2.1 Installazione

La creazione del file eseguibile è effettuata utilizzando un Makefile la cui struttura è la stessa per tutti gli AP del sistema e che è fornito insieme al software WSS.
L'istruzione *make install* effettua la copia dell' eseguibile creato dalla procedura automatica, nella directory di destinazione in cui sono presenti tutti gli eseguibili del WSS.
La struttura delle directories del WSS viene riprodotta su tutte le workstations appartenenti al sistema ed i nomi di tali directories sono impostate come variabili di ambiente all' interno del file di configurazione *.bashrc* di ogni singola workstation.

2.2 arn.h

Il file header *arn.h* contiene la definzione di tutte quelle variabili suscettibili ad eventuali modifiche per l' addattamento finale del codice

dell' AP alla effettiva configurazione del WSS.
Tra queste variabili abbiamo incluso il numero dei parametri richiesti dal software di ARNICA, il nome del device file della porta seriale della worstation sulla quale viene effettuato il collegamento ed una lista di macro a cui corrispondono gli acronimi delle variabili del database del Telescopio Galileo (TDB) che vengono richieste dal software di ARNICA. In particolare l' insieme dei parametri richiesti dal software XNIR è il seguente: le coordinate correnti del telescopio (ascensione retta e declinazione), l' elevazione (per il calcolo della massa d'aria) ed il nome della sorgente osservata, mentre gli acronimi con cui tali grandezze sono identificate all' interno del TDB sono:
- ascensione retta → VTRK_TRK_ALFAUP
- declinazione → VTRK_TRK_DELTAUP
- elevazione → VTRK_TRK_ELASTR
- nome della sorgente → WTRK_ARN_OBJECT

2.3 arn.c

Tutto il codice necessario per il funzionamento dell' AP e contenuto all' interno di questo file sorgente.
In Appendice A riportiamo il listato del file *arn.c* mentre di seguito forniamo una descrizione più accurata di ciascuna funzione in esso contenuta.

2.3.1 main()

Questa funzione svolge i seguenti passi :
- apre e configura la porta seriale: la funzione preposta a questa operazione (*open_input_source()* vedi par. 2.3.2) restituisce un descrittore di file che viene utilizzato nelle operazioni successive
- registra l' acronimo dell' unità relativa all' AP: nel nostro caso è stata utilizzata la sigla ARN
- inizializza a 0 il valore del timeout per la chiamata di sistema *select* (polling)
- inizializza il set dei descrittori di file
- configura il set dei descrittori per la porta selezionata
- chiama la funzione della libreria TDBL *tngAPInit*
- inizializza la struttura che contiene gli acronimi dei parametri del TDB, e le stringhe con i nomi delle variabili utilizzate dal software XNIR. La funzione che compie tale operazione è *init_parameter()* (vedi par 2.3.3)
- infine chiama le altre due funzioni di libreria *tngAPRegisterHandler* e

tngAPMainLoop: la prima registra le funzioni di gestione degli eventi mentre la seconda avvia un loop durante il quale il processo rimane in attesa del verificarsi di uno degli venti registrati.

2.3.2 open_input_source()

La funzione apre la porta seriale, salva la configurazione corrente e configura i nuovi attributi della comunicazione seriale.
I valori relativi a baudrate,parità etc. sono:

9600 8n1

uguali a quelli configurati per la comunicazione seriale all'interno del software di ARNICA.

2.3.3 init_parameter()

La funzione si limita a copiare nella struttura *arn* (definita all' inizio del file sorgente *arn.c*) gli acronimi dei parametri del TDB (definiti come già accennato in precedenza nel file *arn.h*) e le relative stringhe (di due caratteri) utilizzate da XNIR per identificare i parametri richiesti.
La corrispondenza tra i parametri elencati nel file header e le stringhe di due caratteri, identificative degli stessi parametri, utilizzate all' interno del codice di XNIR è la seguente : AR-> ascensione retta, DE-> declinazione, AL->elevazione, NO->nome sorgente.

2.3.4 alm()

Tale funzione viene attivata quando l' AP riceve il comando EXIT inviato dal processo di inizializzazione a tutti i processi "figli" al termine della sessione di lavoro. Nel caso dell' AP da noi sviluppato,la funzione *alm* ristabilisce la configurazione originale della porta seriale e ne effettua la chiusura.

2.3.5 descr()

Questa routine costituisce la parte fondamentale dell'AP.
La funzione *descr()* viene chiamata quando il device file della porta seriale è pronto per essere letto.
Le operazioni svolte dalla routine *descr()* sono perciò le seguenti:

- esegue una lettura del device file. Nel caso che la chiamata di sistema *read* abbia avuto successo, la stringa letta viene analizzata confrontandone il contenuto con il formato prestabilito dal protocollo di comunicazione.

Secondo tale protocollo, la stringa letta deve essere del tipo

"INFO n m\n"

dove n è il numero della prima informazione da leggere ed m il numero di informazioni richieste .

- esegue un ciclo *for* (sul numero dei parametri richiesti, cioè m) durante il quale viene chiamata la funzione di libreria *tngAPRead-Parameter()*. Questa funzione ammette come argomento l' acronimo del parametro del TDB ed una stringa nella quale viene restituito il valore di tale parametro .

- la stringa contenente il parametro letto, viene trasformata dalla funzione *convert()* (vedi par.2.3.6)

- viene costruita la stringa di risposta la quale è costituita dalla stringa di due lettere utilizzata dal software XNIR, il segno di uguale ed il valore relativo della variabile, cioè ad esempio:

  NO=NGC459

  La stringa inoltre inizia e termina con i caratteri CR e LF

- la stringa di risposta viene scritta sul device file della porta seriale.

Nella funzione *descr()* le operazioni di lettura e scrittura sul device file vengono effettuate con le chiamate di sistema *read()* e *write()*. Nelle pagine del manuale in linea relativo a tali chiamate, si legge che se la chiamata è interrota da un segnale prima della lettura (scrittura) dei dati, essa restituisce il valore -1 e pone la variabile *errno* uguale a **EINTR**. Se invece la chiamata è interrotta dal segnale dopo che ha letto (o scritto) con successo alcuni dati, allora restituisce il numero dei bytes letti.
Dato che il WSS fa un uso intensivo di segnali per spedire messaggi ai vari processi è possibile che qualcuno di essi si manifesti durante la fase di lettura o scrittura dei dati sul device file, con il risultato che l' operazione può non essere completata con successo.
A questo scopo sono state scritte due routine di interfaccia per le operazioni di lettura e scrittura sul device file, che tengono conto di questa possibilità. Le due routine sono *read_input()* e *write_output*.

read_input()
la funzione esegue un ciclo i*while* durante il quale esegue i seguenti passi:

- (1) esegue la chiamata di sistema *read* sul device file

- (2) analizza il valore restituito dalla chiamata di sistema: se tale valore è uguale a -1, analizza la flag *errno*. Se questa è uguale a EINTR, la chiamata è stata interrota da un segnale ed allora la funzione ritorna al punto (1),altrimenti la funzione ritorna con valore -1 alla routine chiamante (cioè *descr()* la quale segnalerà all'utente l'errore nella lettura dal device file). Se invece il valore ritornato da *read* è maggiore di 0, la routine prosegue

- (3) per essere certi che la chiamata di sistema non sia stata interrotta da un segnale durante la lettura, viene confrontata il penultimo carattere della stringa letta [1] con il carattere di LF. Infatti quando è selezionato l'input canonico [2] (come nel nostro caso) la lettura viene eseguita fino a che non viene trovato il carattere LF, per cui se tale carattere non è presente nella stringa letta, significa che l' operazione non è stata completata con successo. Se si verifica tale situazione la funzione *read_input* ritorna al punto (1) ripercorrendo gli stessi passi fino alla lettura completa della stringa di input.

write_output()
la funzione esegue un ciclo *while* durante il quale esegue i seguenti passi:

- (1) esegue la chiamata di sistema *write* sul device file

- (2) analizza il valore ritornato dalla chiamata di sistema: se tale valore è uguale a -1, analizza la flag *errno*. Se questa è uguale a EINTR, la chiamata è stata interrota da un segnale ed allora la funzione ritorna al punto (1), altrimenti la funzione ritorna con valore -1 alla routine chiamante (cioè descr() la quale segnalerà all' utente l' errore nella lettura dal device file). Se invece il valore ritornato da *write* è maggiore di 0, la routine prosegue.

- viene calcolata la differenza tra la lunghezza della stringa da scrivere e di bytes scritti. Se tale valore è nullo, la funzione ha completato la scrittura della stringa sul device file e ritorna con successo alla funzione chiamante, altrimenti la funzione ritorna al punto (1) ripercorrendo gli stessi passi.

[1] l' ultimo carattere della stringa è il carattere nullo (in C indica la fine stringa)
[2] nell' input canonico i caratteri ricevuti da linea seriale sono processati per linee la cui fine è determinata dalla presenza del carattere LF.

2.3.6 convert()

Questa funzione viene utilizzata esclusivamente per convertire i valori delle coordinate ascensione retta, declinazione ed altezza da radianti (unità che viene utilizzata per rappresentare gli angoli all' interno del TDB) nelle rispettive unità. In particolare il software XNIR si aspetta di ricevere i valori sotto forma di stringhe del tipo HH:MM:SS,DD per l' ascensione retta e GG:MM:SS,DD per altezza e declinazione. La conversione dei parametri da gradi (float) in stringa di caratteri con la precedente formattazione, viene eseguita dalla routine *gradi2ascii()*.
Per distinguere il tipo di conversione che deve essere effettuata sul parametro, viene utilizzata una flag intera (definita per ciascun dato all'interno della routine *init_parameter()*). I valori che tale flag può assumere sono:
. -1: il parametro non necessita di alcuna conversione
. 0: il parametro deve essere converito da radianti in gradi
. 1: il parametro deve essere convertito da radianti in ore

3 Modifiche

Durante il periodo trascorso tra lo sviluppo del presente modulo software e il trasferimento di ARNICA al Telescopio Nazionale Galileo, il software WSS ha subito alcune modifiche (riguardanti principalmente l' interfacciamento ai processi ancillary) rispetto alla versione installata sulla workstation HP sulla quale abbiamo eseguito le nostre prove di debug.
Il codice da noi sviluppato è stato perciò modificato in loco in modo da uniformarsi alle nuove specifiche.
In Appendice B riportiamo tali modifiche.

4 References

[1] E.Giani,C.Baffa "Ancillary Process per ARNICA al TNG",Arcetri Technical Report n.8/98
[2] A.Balestra,P.Marcucci,F.Paisian,M.Pucillo,R.Smareglia,C.Vuerli "GALILEO Project Workstation Software System", TNG Technical Report n.9,1991
[3] A.Baruffolo, C.Bonoli,A.Ciani "GATE-Command Architecture", TNG Technical Report n.6,1991
[4] C.Vuerli "The TDBL Library User's Guide"

5 Appendice A

```
#define MAIN

#include <tng.h>
#include <termios.h>


/* parameter acronym*/
#include "arn.h"

#define BAUDRATE B9600/* define baud rate*/

/* define the structure to store parameter acronym,relative  values and*/
/* a two-character string needed in xnir program to save the values in */
/* the  right variable                                                 */
typedef struct {
char acronym[NAMELEN];          /* TNG parameter acronym        */
char value[80];                 /* parameter value             */
char twochar[3];                /* XNIR parameter notation      */
int type;                       /* type angle conversion        */
 } param;

int comfd;               /* file descriptor of the serial device        */
struct termios oldtio;   /* structure to save old I/O configuration     */
param arn[MAXPARAM];     /* vector to store parameter information       */
void convert() ;
void gradi2ascii();
void init_parameter();
int read_input();
int write_output();


int alm(from,acronym,txt,flags)
  char from[],acronym[],txt[];
  long flags;
  {
  if(!strncmp(acronym,"EXIT",4)){
```

```
        /* restore old port setting*/
        tcsetattr(comfd,TCSANOW,&oldtio);
        /* close the serial port*/
        close(comfd);
        exit(0);
  }
}

int msg(code)
  int code;
  {
  }

int cmd(from,acronym,txt,flags)
  char from[],acronym[],txt[];
  long flags;
  {
  }

int tout()
  {
  }

/* int descr(fd_set)*****************************************************
 *                                                                      *
 *    This function execute the ancillary work. It reads from the       *
 *    serial port,analyze the input and then reads the required         *
 *    parameters from the TDB. It builds the output string and then     *
 *    writes it to the serial port.                                     *
 *  OSS: the expected format of the input string is of the type:        *
 *     "INFO n m" where the first element is the first row of the       *
 *     parameter table from where to start the trasmission, while the   *
 *     second element is the number of rows to trasmit.                 *
 *     The table is defined as follow:                                  *
 * Row          String to trasmit              Description             *
 *  1             AR= HH.MM.SS,D            current telescope coord.    *
 *  2             DE= GG.MM.SS,D            current telescope coord.    *
 *  3             AL= GG.MM.SS,D            elevation                   *
 *  4             NO= "source name"         source name                 *
 *                                                                      *
 ************************************************************************/

int descr(fds)
```

```
  fd_set fds;           /* file descriptor set                          */
  {
  int rbyte;            /* number of bytes read from the serial port    */
  int wbyte;            /* number of bytes written on the serial port   */
  int i;                /* temporary counter                            */
  int ret;              /* number of bytes returned by scanf operation  */
  int len;              /* length  of the string to write on serial port*/
  int firstelem;        /* index of the first element required          */
  int numanswer;        /* number of parameter asked from xnir          */
  char inpstring[255];  /* string received on serial port               */
  char outstring[255];  /* string to send on serial port                */
  char header[6];
  char errmsg[256];

   rbyte=read_input(comfd,inpstring);
   if(rbyte ==-1){
        sprintf(errmsg,"ARN: read failed");
        tngAPShowWarn(errmsg);
        return(-1);
   }
/*   inpstring[rbyte]='\0';*/
/* analyze the read string and compare the format with the expected one*/
   if(strncmp(inpstring,"INFO",4)){
        sprintf(errmsg,"ARN:error in reading header");
        tngAPShowWarn(errmsg);
        return(-1);
   } else {
        ret=sscanf(inpstring,"%s %d %d",header,&firstelem,&numanswer);
/* if the string header is not equal INFO -> error                       */
/* if the number of data read is less than 3 -> error                    */
        if(ret !=3){
               sprintf(errmsg,"ARN:error in reading number of parameters");
                tngAPShowWarn(errmsg);
                return(-1);
        }
/* if the number of asked parameter is more then MAXPARAM -> error       */
        if(numanswer+firstelem-1 >MAXPARAM){
                sprintf(errmsg,"ARN:too much parameter asked");
                tngAPShowWarn(errmsg);
                return(-1);
 }
/* all ok -> we ask the parameter values                                 */
        for(i=firstelem-1;i<firstelem+numanswer-1;i++){
```

```
                tngAPReadParameter(arn[i].acronym,arn[i].value);
                convert(arn[i].value,arn[i].type);
/* build the string with information for each parameter                 */
                sprintf(outstring,"\n\r%s= %s\n\r",arn[i].twochar,arn[i].value
                len=strlen(outstring);
/* write the string on the serial port                                  */
                wbyte=write_output(comfd,outstring,len);
                if(wbyte ==-1){
                        sprintf(errmsg,"ARN:write failed");
                        tngAPShowWarn(errmsg);
                        return(-1);
}
                outstring[0]='\0';   /* blank the output string        */
   }
        return(0);
  }
}
/* void convert(char*,int)*********************************************
 *                                                                     *
 *This routine execute the conversion from radianti to sexagesimal*
 *      angle or hour,minut,second.                                    *
 * !!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!   *
 * OSS: it seems the angle are store in the TDB in radianti            *
 *                                                                     *
 *********************************************************************/
void convert(value,type)
char value[80];
int type;
{
double angle;

  switch(type){
        case 0: /* input: angolo radiante output -> gradi sessagesimali*/
        case 1: /* input: angolo radiante output -> ore,minuti secondi*/
                angle=atof(value)*(180./M_PI);
                printf("angle %f \n",angle);
                if(type==1){
                        angle=angle/15.;
                }
                gradi2ascii(angle,value);
         default:
                return;
 }
```

```
}

/* void open_input_source(int)****************************************
 *                                                                   *
 *    This function opens the serial port and sets its attributes    *
 *                                                                   *
 ********************************************************************/

void open_input_source(com)
  char com[];
  {
  struct termios newtio;        /* termios structure with new I/O       */
                                /* configuration                        */

/* O_RDWR:   read and write                                           */
/* O_NOCTTY: this program does'n want to be thecontrolling entity     */
/*           for that port.                                           */
/* O_NONBLOCK: the file is opened in non-blocking mode                */

/* open the serial port with the previous flags setted                */
  comfd=open(com,O_RDWR|O_NOCTTY|O_NONBLOCK);
  if(comfd<0) {
return;
  }
/* save current serial port setting                                   */
  tcgetattr(comfd,&oldtio);
/* clear struct for new port setting                                  */
  bzero(&newtio,sizeof(newtio));
/*
BAUDRATE: set bps rate
CS8     : 8n1 (8 bit,no parity,1 stopbit)
CLOCAL  : local connection,no modem control
CREAD   : enable receiving characters
*/
  newtio.c_cflag=BAUDRATE|CS8|CLOCAL|CREAD;

/*
ICRNL: map CR to NL
*/
  newtio.c_iflag=ICRNL;
/* Raw output*/
  newtio.c_oflag=0;
/* ICANON: enable canonical input                          */
```

```
  newtio.c_lflag=ICANON;
/* clean the  modem line and activate the setting for the port  */
  tcflush(comfd,TCIFLUSH);
  tcsetattr(comfd,TCSANOW,&newtio);
  return;
}

/* void init_parameter()************************************************
 *                                                                     *
 *This function sets the relation between TNG parameter acronym        *
 *and the two-char string for XNIR parameter notation.                 *
 *                                                                     *
 ***********************************************************************/
void init_parameter()
{


/* set the first parameter acronym and initial value            */
 sprintf(arn[0].acronym,"%s",ASR);
 sprintf(arn[0].twochar,"%s","AR");
 arn[0].type=1;/* conversion from radianti to HH.MM.SS,DD      */

/* set the second parameter acronym and initial value           */
 sprintf(arn[1].acronym,"%s",DEC);
 sprintf(arn[1].twochar,"%s","DE");
 arn[1].type=0;/* conversion from radianti to GG.MM.SS,DD      */

/* set the third parameter acronym and initial value            */
 sprintf(arn[2].acronym,"%s",EL);
 sprintf(arn[2].twochar,"%s","AL");
 arn[2].type=0;/* conversion from radianti to GG.MM.SS,DD      */

/* set the fourth parameter acronym and initial value        */
 sprintf(arn[3].acronym,"%s",OBJ);
 sprintf(arn[3].twochar,"%s","NO");
 arn[3].type=-1;/* no conversion                               */


main()
{
  fd_set readfs; /* file descriptor set*/
  char myacr[NAMELEN];/* ancillary acronym*/
  char uifacr[NAMELEN];/* system acronym*/
```

```
  struct timeval timeout;

/* opens a device, sets the port correctly and returns a file descriptor*/
  open_input_source(SERIALDEVICE);
  if(comfd <0){/* error in opening port*/
        fprintf(stderr,"ARN:can't open the port %s",SERIALDEVICE);
        exit(1);
  }

/* get the ancillary and system acronym               */
  sprintf(myacr,"%s_ARN",getenv("TNGSYSTEM"));
  uppercase(myacr);
  sprintf(uifacr,"%s_UIF",getenv("TNGSYSTEM"));
  uppercase(uifacr);

/* initialize the timeout for the select call to 0. This will trasform*/
/* the call to a polling call */
  memset(&timeout,0,sizeof(struct timeval));
/* clear the descriptor set*/
  FD_ZERO(&readfs);
/* set testing for the selected port*/
  FD_SET(comfd,&readfs);
  tngAPInit(myacr,uifacr,NULL,&readfs);
/* initialize the parameters table        */
  init_parameter();
  tngAPRegisterHandlers(cmd,alm,msg,tout,descr);
  tngAPMainLoop();
}



/** void gradi2ascii(double, char * ) ****************************
 *                                                                 *
 * Routinne to convert degree in a ascii form as  [-]GG.MM.SS,D   *
 * It can handle also   HH MM SS D if before we divide by 15      *
 *                                                                 *
 *  (C.Baffa)   *
 ******************************************************************/

void gradi2ascii(valore, /* value to convert                    */
                 buf)     /* string of result                    */
double valore;
char *buf;
```

```
{
 int j=1;                        /* temporary                          */
 char gg[4],mm[3],ss[5];         /* partial value strings              */
 float dd=(float)0.0;            /* to handle decimals of seconds      */


/* we handle signum                                                    */
 if (valore <= 0.) {
j = -1;
valore = -valore;
 }

/* for rounding purpose                                                */
 valore += .5 / 36000.;

/* partial results by iteration of taking integer part                 */
 if (valore > 10.)
        sprintf (gg, "%d",(int)valore);
 else
        sprintf (gg,"0%d",(int)valore);

 valore = 60. * ( valore - (double)(int)valore);

 if (valore > 10.)
sprintf (mm, "%d",(int)valore);
 else
sprintf (mm,"0%d",(int)valore);

 valore = 60. * ( valore - (double)(int)valore);

 if (valore > 10.)
sprintf (ss, "%d",(int)valore);
 else
sprintf (ss,"0%d",(int)valore);

 valore = 10. * ( valore - (double)(int)valore) ;


/* we concatenate results                                              */
 if ( j >= 0 )                   /* +/- gestion                        */
        sprintf(buf,"%s.%s.%s,%1d",gg,mm,ss,(int)valore);
 else
        sprintf(buf,"-%s.%s.%s,%1d",gg,mm,ss,(int)valore);
```

```
}


int read_input(comfd,inpstring)
int comfd;
char *inpstring;
{

 char readstring[255];
 int rbyte;

 inpstring[0]='\0';
 while(TRUE){
 readstring[0]='\0';
rbyte=read(comfd,readstring,255);
if(rbyte==-1){
if(errno==EINTR){
/* read is interrupted by a signal*/
continue;
} else {
return(-1);
}
} else if(rbyte > 0){
readstring[rbyte]='\0';
strcat(inpstring,readstring);
if(readstring[rbyte-1]=='\n'){
/* if ICANON and ICRNL are set, the last character read is a NL (ASCII LF)*/
                        break;
                } else {
/* read is interrupted by a signal after it has successfully read some data*/
                        continue;
                }
        }
 }
 return(0);
}

int write_output(comfd,outstring,len)
int comfd;
int len;
char *outstring;
{
```

```
 char *writestring;
 int wbyte;

 while(TRUE){
        wbyte=write(comfd,outstring,len);
        if(wbyte==-1){
                if(errno==EINTR){
                        continue;
                 else {
                        return(-1);
                }
         } else {
                len-=wbyte;
                if(len){
                        writestring=&outstring[wbyte];
                        writestring[len]='\0';
                        outstring=writestring;
                        continue;
                } else {
                        break;
                }
        }
}
return(0);
}
```

6 Appendice B

In questa appendice riportiamo la versione modificata delle due funzioni che hanno subito le modifiche, cioè *descr()* e *main()*.

6.1 routine descr()

```
int descr(fds)
  fd_set fds;/* file descriptor set*/
  {
  int rbyte;/* number of bytes read from the serial port*/
  int wbyte;/* number of bytes written on the serial port*/
  int i;/* temporary counter*/
  int ret;/* number of bytes returned by scanf operation*/
  int len;/* length  of the string to write on serial port*/
  int firstelem;/* index of the first element required */
  int numanswer;/* number of parameter asked from xnir*/
  char inpstring[255];/* string received on serial port*/
  char outstring[255];/* string to send on serial port*/
  char header[6];
  char errmsg[256];

   rbyte=read_input(comfd,inpstring);
   if(rbyte ==-1){
sprintf(errmsg,"ARN: read failed");
tngAPShowWarn(errmsg);
return(-1);
   }
/*   inpstring[rbyte]='\0';*/
/* analyze the read string and compare the format with the expected one*/
   if(strncmp(inpstring,"INFO",4)){
/*        printf("\n Stringa Errata %s",inpstring);*/
sprintf(errmsg,"ARN:error in reading header");
tngAPShowWarn(errmsg);
return(-1);
   } else {
/*   printf("\n Stringa Giusta %s",inpstring);*/
   ret=sscanf(inpstring,"%s %d %d",header,&firstelem,&numanswer);
/* if the string header is not equal INFO -> error*/
/* if the number of data read is less than 3 -> error*/
   if(ret !=3){
sprintf(errmsg,"ARN:error in reading number of parameters");
tngAPShowWarn(errmsg);
```

```
return(-1);
   }
/* if the number of asked parameter is more then MAXPARAM -> error*/
   if(numanswer+firstelem-1 >MAXPARAM){
sprintf(errmsg,"ARN:too much parameter asked");
tngAPShowWarn(errmsg);
return(-1);
   }
/* all ok -> we ask the parameter values*/
   for(i=firstelem-1;i<firstelem+numanswer-1;i++){
/*Modifica fatta da Zacchei*/
/*tngAPReadParameter(arn[i].acronym,arn[i].value);*/
if (!tngAPReadParameter(arn[i].acronym,arn[i].value))
printf("tng_errno : %d\n",tng_errno);
/*Fine modifica fatta da Zacchei*/
printf("\n Acronimo %s \n",arn[i].acronym);
printf("\n Valori Letti %s \n",arn[i].value);
convert(arn[i].value,arn[i].type);
printf("\n Valori Convertiti %s \n",arn[i].value);
/* build the string with information for each parameter*/
sprintf(outstring,"\n\r%s= %s\n\r",arn[i].twochar,arn[i].value);
len=strlen(outstring);
/* write the string on the serial port*/
wbyte=write_output(comfd,outstring,len);
if(wbyte ==-1){
sprintf(errmsg,"ARN:write failed");
tngAPShowWarn(errmsg);
return(-1);
}
outstring[0]='\0';/* blank the output string*/
   }
 return(0);
  }
}
```

Eseguendo un confronto tra la routine sopra riportata e la corrispondente contenuta in Appendice A, si può verificare che le modifiche riguardano:

- l'aggiunta della stampa a schermo di ulteriori informazioni utilizzate durante la fase di debug dell' AP effettuata al TNG.
- il controllo del valore restituito dalla funzione della libreria TDBL *tngApReadParameter()*

6.2 routine main()

```
main()
{
  fd_set readfs;                          /* file descriptor set         */
  char myacr[NAMELEN];                    /* ancillary acronym           */
  char uifacr[NAMELEN];                   /* system acronym              */
  struct timeval timeout;

/* opens a device, sets the port correctly and returns a file descriptor*/
  open_input_source(SERIALDEVICE);
  if(comfd <0){                           /* error in opening port        */
fprintf(stderr,"ARN:can't open the port %s",SERIALDEVICE);
exit(1);
  }

/* get the ancillary and system acronym*/
/* sprintf(myacr,"%s_ARN",getenv("TNGSYSTEM")); */
  sprintf(myacr,"ARN");
  uppercase(myacr);
/*  sprintf(uifacr,"%s_UIF",getenv("TNGSYSTEM"));*/
  sprintf(uifacr,"UIF");
  uppercase(uifacr);

/* initialize the timeout for the select call to 0. This will trasform*/
/* the call to a polling call */
  memset(&timeout,0,sizeof(struct timeval));
/* clear the descriptor set*/
  FD_ZERO(&readfs);
/* set testing for the selected port*/
  FD_SET(comfd,&readfs);
  tngAPInit(myacr,uifacr,NULL,&readfs);
/* initialize the parameters table*/
  init_parameter();
  tngAPRegisterHandlers(cmd,alm,msg,tout,descr);
  tngAPMainLoop();
}
```

Le variazioni presenti nella funzione principale sono legate esclusivamente alle modifiche effettuate dai responsabili del software WSS riguardanti la costruzione dell' acronimo del sistema contenente l' unità costituita dall' ancillary process.