Documentation on ALFOSC_SER_DAEMON (ALFOSC low level RS-232 daemon).

Jacob Wang Clasen
Copenhagen University Observatory

July 10, 1996

Contents

Chapter 1

Description

The program ALFOSC_SER_DAEMON is responsible for sending and receiving data to/from the RS-232 serial port connected to the ALFOSC motor controller. It is also responsible for sending data to and from the IDL ALFOSC User Interface (referred to as UIF from now on).
The connection between ALFOSC_SER_DAEMON and UIF is implemented by named (FIFO) pipes. Two pipes are used, one for sending commands from UIF to ALFOSC_SER_DAEMON and one for sending replies from ALFOSC_SER_DAEMON to UIF. These pipes is located in `$UIF_DIR/io/`.
The environment variable `$UIF_DIR` is needed for the UIF. It's defined by typing the command (for tcsh) `setenv UIF_DIR /home/jacob/uif` if the UIF program is located in the directory `/home/jacob/uif`.

Chapter 2

Compilation

Change directory (command `cd`) to `$UIF_DIR/src`. The directory should contain the files

- `alfosc_ser_daemon.c`
- `makefile.ser_daemon`

Copy the file `makefile.ser_daemon` to `makefile`, and type `make`. The directory will now also contain the file

- `alfosc_ser_daemon`

The compilation is completed.

Chapter 3

Installation

When you compile the program, it is also copied to the right directory for use at runtime. The last line in the file `$UIF_DIR/src/makefile` copies the program to `$UIF_DIR/io/alfosc_ser_daemon`.
The BFOSC_SER_DAEMON program needs one environment variable to work (`ALFOSC_SER_PORT`), and three to run in the ALFOSC/FASU environment. These are as follows

- `ALFOSC_SER_PORT`
  The divice file connected to the ALFOSC motor controller (e.g. /dev/ttyS1).
- `FASU_SER_PORT`
  The device file connected to the FASU motor controller (e.g. /dev/ttyS2).
- `DEV_DIR`
  The directory which holds the FIFO pipes (see later).

These env. vars. should be set in the run command script for the shell in use (if a `tcsh` is used, the file is `~/.tcshrc`). The names of the FIFO pipes must be

- `$UIF_DIR/pipes/alfosc_serial_pipe_i`
- `$UIF_DIR/pipes/alfosc_serial_pipe_o`

These pipes is created with the command

- `mknod alfosc_serial_pipe_i p`
- `mknod alfosc_serial_pipe_o p`

Be carefull not to assign the port for the mouse to the motor controller!

Chapter 4

Usage

4.1 Normal usage

In normal operation, the usage is hidden from the user. The program is started by the `uif.csh` script and interacts quietly with the UIF. By quitting the UIF, the ALFOSC_SER_DAEMON is killed.

4.2 Debug mode

The ALFOSC_SER_PORT program can be used independently of the UIF. This is done by not defining the env. var. `$DEV_DIR`. The program then uses standard input as command input and standard output as reply output (like a terminal). This mode is usefull for testing the interface independently of the UIF.
When in "debug mode", the program accepts all the commands understood by the ASYNC multiplexer in the ALFOSC motor controller plus a few others. Following is the complete command set

- *X*I
  Initialize pos. to hall switch position.

- *X*G*nnnnnn*
  Go to position *nnnnnn*.

- *X*M*nnnnnn*
  Move relative. + or - can precede *nnnnnn* .

- *X*P
  Send position.

- *X*1, *x*2...*x*8
  Go to one of 8 positions.

- *X*q
  Quit the current operation.

- *x*
  Check driver ready.
- *X*X
  Go to hall switch init position. Used for resetting the nnn_zero value to 0.
- *X*idfoc
  Set the zero position to this position.

The *X* can be one of A (Apeture), F (Filter), G (Grism). For the Shutter and focus movement, the following commands are valid

- SI
  Initialize position to hall switches.
- SE*nnnnn*
  Start exposure for *nnnnn* sec.
- SS
  Stop exposure.
- SR
  Resume exposure for the rest of seconds started with SE.
- SA*nnnnn*
  Add *nnnnn* * 0.1 sec. to the exposure time.
- SG*nnnn*
  Go to position *nnnn*.
- SP
  Send position.
- S1,S2,S3,S4
  Go to one of 4 positions.
- Sq
  Quit the current operation.
- s
  Check driver ready.
- CI
  Initialize position to hall switches.
- CG*nnnn*
  Go to position *nnnn*.
- CM*nnnn*
  Move relative.

- CP
  Send position.
- Cq
  Quit the current operation.
- c
  Check driver ready.

Additional commands:

- PO
  Send string with all the positions in.
- q
  Quit the BFOSC_SER_DAEMON program.

`-- FASU COMMAND LIST --`

See also the document "The ALFOSC motor controller".

Chapter 5

Bugs

As of july 9. 1996, no bugs has been found or reported.

Chapter 6

Program listing.

6.1 Main program.

```
/**************************************************************
* Original file: bfosc_ser_daemon                             *
*                                                             *
* Changed to:    dfosc_ser_daemon                             *
*                (Renamed to alfosc_ser_daemon)               *
*                                                             *
* Usage:         New serial port daemon for NOT/ALFOSC        *
*                to run on Linux.                             *
*                                                             *
* Interfaces:    - Serial port on Linux machine.              *
*                - FIFO pipes or STDIN/STDOUT                 *
*                                                             *
* IF program:    1) A 'socket to pipe' program.               *
*                   "daisy_chain.c".                          *
*                2) ALFOSC UIF via the pipes.                 *
*                3) STDIN/OUT for debugging.                  *
*                                                             *
* ENV VAR's :    ALFOSC_SERPORT  dev. file for FOSC           *
*                 If not set, prog. will exit.                *
*                ROT_SERPORT    dev. file for rotator         *
*                 If not set, prog. will continue.            *
*                 Not used on the NOT.                        *
*                FASU_SERPORT   dev. file for FASU            *
*                 It not set, prog. will continue.            *
*                DEV_DIR        path to pipes                 *
*                 If not set, prog. will use STDIN/STDOUT     *
*                 instead of pipes.                           *
*                All vars. should be set for normal           *
*                operation both with ALFOSC UIF and           *
*                "daisy_chain".                               *
*                                                             *
* Commands  :    All valid ASYNC commands will give answers   *
*                 according to the state of the inst.         *
*                 'q' send via STDIN or input pipe will       *
*                 quit the program.                           *
*                                                             *
* Pipe conv.:    Pipe names are seen from the user prog.'s    *
*                point of view, i.e.                          *
*                alfosc_serial_pipe_o : Output pipe from      *
*                 UIF/SOCK IF (input to this program)         *
*                alfosc_serial_pipe_i : Input pipe to         *
*                 UIF/SOCK IF (output from this program)      *
* OBS:           The motorcommands to the FASU are 'a' and    *
*                'b'. To avoid confition with the fosc slit   *
```

```
*               movement (alos 'a'), the FASU wheels are    *
*               called 'u' and 'v' in the code and only the *
*               actual motorcommands will be 'a' and 'b'.   *
*                                                           *
* Rev. hist:    Copied from "dfisc_ser_daemon"              *
*               12. apr. 1996: 1. version with ALFOSC.      *
*                                                           *
* Last ver.:    4. jul. 1996: dfosc names changed to        *
*                 alfosc.                                   *
************************************************************/

#include <stdio.h>
#include <errno.h>
#include <fcntl.h>
#include <termio.h>
#include <signal.h>
#include <sys/time.h>
#include <sys/ioctl.h>
#include <sys/types.h>
#include <unistd.h>
#include <stdlib.h>
#include <string.h>

typedef enum {FALSE,TRUE} BOOLEAN;

extern void setblock(int fd,BOOLEAN on);
extern void do_timer_stuff(void);
extern void timer_routine_real(void);
extern void start_timer_real(int t);
extern void stop_timer_real(void);
extern void decode_pip_command(void);
extern void decode_ser_command_alfosc(void);
extern void decode_ser_command_rotator(void);
extern void decode_ser_command_fasu(void);

/************************************************************/
char timer_flag = 0;

char SerPort[80];
char RotSerPort[80];
char FasuSerPort[80];
int serport;
int rot_serport;
int fasu_serport;
int alfosc_serial_pipe_i;
int alfosc_serial_pipe_o;

char ser_command_alfosc[40];
int ser_command_alfosc_p = 0;

char ser_command_rotator[40];
int ser_command_rotator_p = 0;

char ser_command_fasu[40];
int ser_command_fasu_p = 0;

char pip_command[40];
int pip_command_p = 0;

int alfosc_apos, alfosc_fpos, alfosc_gpos, alfosc_cpos, alfosc_spos;
int alfosc_aok, alfosc_fok, alfosc_gok, alfosc_cok, alfosc_sok;
int rot_pos, rot_ok;
int fasu_upos, fasu_vpos;
int fasu_uok, fasu_vok;

char FASU_flag = TRUE;
char ROT_flag = TRUE;
/************************************************************/
```

```
void syserr(char *msg)
{
extern int errno,sys_nerr;
extern char *sys_errlist[];

fprintf(stderr,"ERROR: %s (%d",msg,errno);
if ( errno > 0 && errno < sys_nerr )
fprintf(stderr,";%s)\n",sys_errlist[errno]);
else
fprintf(stderr,")\n)");

setblock(0,TRUE);
system("stty sane");
exit(1);
}
/**************************************************************/
void setblock(int fd,BOOLEAN on)
{
static int blockf,nonblockf;
int flags;

if ((flags = fcntl(fd,F_GETFL,0)) == -1)
syserr("fcntl");
blockf = flags &  ~O_NDELAY;
nonblockf = flags | O_NDELAY;
if ( fcntl(fd,F_SETFL, on ? blockf : nonblockf ) == -1 )
syserr("fcntl2");
}
/**************************************************************/
void setraw(int fd)
{
struct termio tbuf;

if ( ioctl(fd,TCGETA,&tbuf) == -1 )
syserr("ioctl");
tbuf.c_iflag &= ~(INLCR | ICRNL | IUCLC | ISTRIP | IXON | BRKINT);
tbuf.c_oflag &= ~OPOST;
tbuf.c_lflag &= ~(ICANON | ISIG | ECHO);
tbuf.c_cflag = B9600 | CS8 | CREAD;
tbuf.c_cc[4] = 5;
tbuf.c_cc[5] = 2;
if ( ioctl(fd,TCSETAF,&tbuf) == -1 ) syserr("ioctl2");
}
/**************************************************************/
struct itimerval T;
/*********************************************************************/
void start_timer_real(int t)
{
t *= 1000;
T.it_interval.tv_sec = T.it_value.tv_sec = 0;
T.it_interval.tv_usec = T.it_value.tv_usec = t;

signal(SIGALRM,&timer_routine_real);
setitimer(ITIMER_REAL,&T,NULL);
}
/*********************************************************************/
void stop_timer_real(void)
{
T.it_interval.tv_sec = T.it_interval.tv_usec = \
    T.it_value.tv_sec = T.it_value.tv_usec = 0;

setitimer(ITIMER_REAL,&T,NULL);
}
/*********************************************************************/
void timer_routine_real(void)
{
stop_timer_real();
```

```
timer_flag = TRUE;

start_timer_real(250);
}
/**********************************************************/
void reset_ser_command_alfosc(void)
{
memset(ser_command_alfosc,0,40);
ser_command_alfosc_p = 0;
}
/**********************************************************/
void reset_ser_command_rotator(void)
{
memset(ser_command_rotator,0,40);
ser_command_rotator_p = 0;
}
/**********************************************************/
void reset_ser_command_fasu(void)
{
memset(ser_command_fasu,0,40);
ser_command_fasu_p = 0;
}
/**********************************************************/
void reset_pip_command(void)
{
memset(pip_command,0,40);
pip_command_p = 0;
}
/**********************************************************/

void main(void)
{
char cs,cp;
char *SP;  /* ALFOSC ser port */
char *RSP; /* Rotator ser port */
char *FSP; /* FASU ser port */
char dev_dir[80],pipe[80];

fprintf(stderr,"Starting DFOSC serial port controller daemon\n");

SP  = getenv("ALFOSC_SERPORT");
RSP = getenv("ROT_SERPORT");
FSP = getenv("FASU_SERPORT");

if ( SP == NULL )
{
fprintf(stderr,"No 'ALFOSC_SERPORT' defined\n");
exit(1);
}
if ( RSP == NULL )
{
fprintf(stderr,"Warning! No 'ROT_SERPORT' defined\n");
ROT_flag = FALSE;
}
if ( FSP == NULL )
{
fprintf(stderr,"Warning! No 'FASU_SERPORT' defined\n");
FASU_flag = FALSE;
}
strcpy(SerPort,SP);
if ( ROT_flag == TRUE ) strcpy(RotSerPort,RSP);
if ( FASU_flag == TRUE ) strcpy(FasuSerPort,FSP);

fprintf(stderr,"Serial port: %s\n",SerPort);
if ( ROT_flag == TRUE )
fprintf(stderr,"Rotator port: %s\n",RotSerPort);
if ( FASU_flag == TRUE )
fprintf(stderr,"FASU port: %s\n",FasuSerPort);
```

```
if ( ( serport = open( SerPort, O_RDWR | O_NDELAY )) == -1)
syserr("serport open");

if ( ROT_flag == TRUE )
{
if ( ( rot_serport = open( RotSerPort, O_RDWR | O_NDELAY )) == -1)
syserr("Rotator serport open");
}

if ( FASU_flag == TRUE )
{
if ( ( fasu_serport = open( FasuSerPort, O_RDWR | O_NDELAY )) == -1)
syserr("FASU serport open");
}

SP = getenv("DEV_DIR"); /* env.var for the FIFO dir. */
if ( SP == NULL )
    {
alfosc_serial_pipe_o = 1;
alfosc_serial_pipe_i = 0;
    }
else
{
strcpy(dev_dir,SP);
strcpy(pipe,dev_dir); strcat(pipe,"/alfosc_serial_pipe_o");
alfosc_serial_pipe_o = open(pipe,O_RDONLY );
strcpy(pipe,dev_dir); strcat(pipe,"/alfosc_serial_pipe_i");
alfosc_serial_pipe_i = open(pipe,O_WRONLY );
fprintf(stdout,"Pipes opened...\n");
}


if ( ( alfosc_serial_pipe_i <= -1 ) || ( alfosc_serial_pipe_o <= -1 ))
    {
    fprintf(stderr,"\n1. Error opening a pipe\n");
    close(alfosc_serial_pipe_i); close(alfosc_serial_pipe_o);
    exit(-1);
    }

setblock(alfosc_serial_pipe_o, FALSE);
setblock(alfosc_serial_pipe_i, FALSE);

alfosc_aok = alfosc_fok = alfosc_gok = alfosc_cok = alfosc_sok = FALSE;
alfosc_apos = alfosc_fpos = alfosc_gpos = alfosc_cpos = alfosc_spos = -99999;
rot_ok = FALSE;
rot_pos = -99999;
fasu_uok = fasu_vok = FALSE;
fasu_upos = fasu_vpos = -99999;

start_timer_real(250);

cs = cp = 0;
while ( cp != 'q' )
{
while (read(alfosc_serial_pipe_o,&cp,1) == 1 )
{
if ( ( cp == '\n' ) || ( cp == '\r' ) || ( pip_command_p == 79 ) )
{
decode_pip_command();
reset_pip_command();
}
else
{
pip_command[pip_command_p++] = cp;
}
}
```

```
while (read(serport,&cs,1) == 1 )
{
if ( ( cs == '\n' ) || ( cs == '\r' ) || (ser_command_alfosc_p == 79) )
{
decode_ser_command_alfosc();
reset_ser_command_alfosc();
}
else
{
ser_command_alfosc[ser_command_alfosc_p++] = cs;
}
}

if ( ROT_flag == TRUE )
{
while (read(rot_serport,&cs,1) == 1 )
{
if ( ( cs == '\n' ) || ( cs == '\r' ) || \
( ser_command_rotator_p == 79 ) )
{
decode_ser_command_rotator();
reset_ser_command_rotator();
}
else
{
ser_command_rotator[ser_command_rotator_p++] = cs;
}
}
}

if ( FASU_flag == TRUE )
{
while (read(fasu_serport,&cs,1) == 1 )
{
if ( ( cs == '\n' ) || ( cs == '\r' ) || \
( ser_command_fasu_p == 79 ) )
{
decode_ser_command_fasu();
reset_ser_command_fasu();
}
else
{
ser_command_fasu[ser_command_fasu_p++] = cs;
}
}
}

if ( timer_flag )
{
do_timer_stuff();
timer_flag = FALSE;
}
usleep(100);
}
stop_timer_real();
close(serport);
close(rot_serport);
close(fasu_serport);
close(alfosc_serial_pipe_i);
close(alfosc_serial_pipe_o);
}
/**********************************************************/
void do_timer_stuff(void)
{
char buf[40];

if ( alfosc_aok == FALSE ) strcpy(buf,"a");
else strcpy(buf,"AP");
```

```
if ( alfosc_fok == FALSE ) strcat(buf,"f");
else strcat(buf,"FP");
if ( alfosc_gok == FALSE ) strcat(buf,"g");
else strcat(buf,"GP");
if ( alfosc_cok == FALSE ) strcat(buf,"c");
else strcat(buf,"CP");
if ( alfosc_sok == FALSE ) strcat(buf,"s");
else strcat(buf,"SP");
write(serport,buf,strlen(buf));


if ( ROT_flag == TRUE )
{
if ( rot_ok == FALSE ) strcpy(buf,"r");
else strcpy(buf,"RP");
write(rot_serport,buf,strlen(buf));
}

if ( FASU_flag == TRUE )
{
if ( fasu_uok == FALSE ) strcpy(buf,"a");
else strcpy(buf,"AP");
if ( fasu_vok == FALSE ) strcat(buf,"b");
else strcat(buf,"BP");
write(fasu_serport,buf,strlen(buf));
}
}
/*************************************************************/
void decode_pip_command(void)
{
char CR=0x0d;
char sb[80];

/*fprintf(stdout,"pip_comm: [%s]\n",pip_command);fflush(stdout);*/

if ( !strncmp("PO",pip_command,2) )
{
if ( alfosc_apos < 0 ) strcpy(sb," *******");
else sprintf(sb," %7d",alfosc_apos);
if ( alfosc_fpos < 0 ) strcat(sb," *******");
else sprintf(&sb[8]," %7d",alfosc_fpos);
if ( alfosc_gpos < 0 ) strcat(sb," *******");
else sprintf(&sb[16]," %7d",alfosc_gpos);
if ( alfosc_cpos < 0 ) strcat(sb," *******");
else sprintf(&sb[24]," %7d",alfosc_cpos);
if ( alfosc_spos < 0 ) strcat(sb," *******");
else sprintf(&sb[32]," %7d",alfosc_spos);

if (rot_pos < 0 ) strcat(sb," *******\r\n");
else sprintf(&sb[40]," %7d\r\n",rot_pos);

write(alfosc_serial_pipe_i, sb, strlen(sb));
return;
}

if ( !strncmp("q",pip_command,1) )
{
stop_timer_real();
close(serport);
close(rot_serport);
close(fasu_serport);
close(alfosc_serial_pipe_o);
close(alfosc_serial_pipe_i);
exit(1);
}
if ( !strncmp("IN",pip_command,2) )
{
strcpy(sb,"AIFIGICISI");
```

```
write(serport,sb,10);
alfosc_aok = alfosc_fok = alfosc_gok = alfosc_cok = alfosc_sok = FALSE;
alfosc_apos = alfosc_fpos = alfosc_gpos = alfosc_cpos = alfosc_spos = -99999;
return;
}
/* Commands for the Aperture movement */
if ( !strncmp("AG",pip_command,2) )
{
write(serport,pip_command,strlen(pip_command));
write(serport,&CR,1);
alfosc_aok = FALSE;
alfosc_apos = -99999;
return;
}
if ( !strncmp("AI",pip_command,2) )
{
write(serport,pip_command,strlen(pip_command));
    alfosc_aok = FALSE;
    alfosc_apos = -99999;
return;
}
if ( !strncmp("AP",pip_command,2) )
{
sprintf(sb,"%d\r\n",alfosc_apos);
write(alfosc_serial_pipe_i, sb, strlen(sb));
return;
}
if ( !strncmp("Aq",pip_command,2) )
{
write(serport,pip_command,strlen(pip_command));
    alfosc_aok = FALSE;
    alfosc_apos = -99999;
return;
}
if ( !strncmp("a",pip_command,1) )
{
if (alfosc_aok) strcpy(sb,"ay\r\n");
else
strcpy(sb,"an\r\n");
write(alfosc_serial_pipe_i, sb, strlen(sb));
return;
}
if ( !strncmp("A",pip_command,1) )
{
write(serport,pip_command,strlen(pip_command));
write(serport,&CR,1);
alfosc_aok = FALSE;
alfosc_apos = -99999;
return;
}
/* Commands for the Filter movement */
if ( !strncmp("FG",pip_command,2) )
    {
    write(serport,pip_command,strlen(pip_command));
    write(serport,&CR,1);
    alfosc_fok = FALSE;
    alfosc_fpos = -99999;
    return;
    }
if ( !strncmp("FI",pip_command,2) )
    {
    write(serport,pip_command,strlen(pip_command));
    alfosc_fok = FALSE;
    alfosc_fpos = -99999;
    return;
    }
if ( !strncmp("FP",pip_command,2) )
    {
```

```
    sprintf(sb,"%d\r\n",alfosc_fpos);
    write(alfosc_serial_pipe_i, sb, strlen(sb));
    return;
    }
if ( !strncmp("Fq",pip_command,2) )
{
write(serport,pip_command,strlen(pip_command));
    alfosc_fok = FALSE;
    alfosc_fpos = -99999;
return;
}
if ( !strncmp("f",pip_command,1) )
    {
    if (alfosc_fok) strcpy(sb,"fy\r\n");
    else
    strcpy(sb,"fn\r\n");
    write(alfosc_serial_pipe_i, sb, strlen(sb));
    return;
    }
if ( !strncmp("F",pip_command,1) )
    {
    write(serport,pip_command,strlen(pip_command));
    write(serport,&CR,1);
    alfosc_fok = FALSE;
    alfosc_fpos = -99999;
    return;
    }
/* Commands for the Grism movement */
if ( !strncmp("GG",pip_command,2) )
    {
    write(serport,pip_command,strlen(pip_command));
    write(serport,&CR,1);
    alfosc_gok = FALSE;
    alfosc_gpos = -99999;
    return;
    }
if ( !strncmp("GI",pip_command,2) )
    {
    write(serport,pip_command,strlen(pip_command));
    alfosc_gok = FALSE;
    alfosc_gpos = -99999;
    return;
    }
if ( !strncmp("GP",pip_command,2) )
    {
    sprintf(sb,"%d\r\n",alfosc_gpos);
    write(alfosc_serial_pipe_i, sb, strlen(sb));
    return;
    }
if ( !strncmp("Gq",pip_command,2) )
{
write(serport,pip_command,strlen(pip_command));
    alfosc_gok = FALSE;
    alfosc_gpos = -99999;
return;
}
if ( !strncmp("g",pip_command,1) )
    {
    if (alfosc_gok) strcpy(sb,"gy\r\n");
    else
    strcpy(sb,"gn\r\n");
    write(alfosc_serial_pipe_i, sb, strlen(sb));
    return;
    }
if ( !strncmp("G",pip_command,1) )
    {
    write(serport,pip_command,strlen(pip_command));
    write(serport,&CR,1);
```

```
    alfosc_gok = FALSE;
    alfosc_gpos = -99999;
    return;
    }
/* Commands for the Focus movement */
if ( !strncmp("CI",pip_command,2) )
    {
    write(serport,pip_command,strlen(pip_command));
    alfosc_cok = FALSE;
    alfosc_cpos = -99999;
    return;
    }
if ( !strncmp("CP",pip_command,2) )
    {
    sprintf(sb,"%d\r\n",alfosc_cpos);
    write(alfosc_serial_pipe_i, sb, strlen(sb));
    return;
    }
if ( !strncmp("CG",pip_command,1) )
    {
    write(serport,pip_command,strlen(pip_command));
    write(serport,&CR,1);
    alfosc_cok = FALSE;
    alfosc_cpos = -99999;
    return;
    }
if ( !strncmp("Cq",pip_command,2) )
{
write(serport,pip_command,strlen(pip_command));
    alfosc_cok = FALSE;
    alfosc_cpos = -99999;
return;
}
if ( !strncmp("c",pip_command,1) )
    {
    if (alfosc_cok) strcpy(sb,"cy\r\n");
    else
    strcpy(sb,"cn\r\n");
    write(alfosc_serial_pipe_i, sb, strlen(sb));
    return;
    }
/* Commands for the shutter*/
if ( !strncmp("SI",pip_command,2) )
    {
    write(serport,pip_command,strlen(pip_command));
    alfosc_sok = FALSE;
    alfosc_spos = -99999;
    return;
    }
if ( !strncmp("SP",pip_command,2) )
    {
    sprintf(sb,"%d\r\n",alfosc_spos);
    write(alfosc_serial_pipe_i, sb, strlen(sb));
    return;
    }
if ( !strncmp("SE",pip_command,2) )
    {
    write(serport,pip_command,strlen(pip_command));
    write(serport,&CR,1);
    alfosc_sok = FALSE;
    alfosc_spos = -99999;
    return;
    }
if ( !strncmp("Sq",pip_command,2) )
{
write(serport,pip_command,strlen(pip_command));
    alfosc_sok = FALSE;
    alfosc_spos = -99999;
```

```
return;
}
if ( !strncmp("s",pip_command,1) )
    {
    if (alfosc_sok) strcpy(sb,"sy\r\n");
    else
    strcpy(sb,"sn\r\n");
    write(alfosc_serial_pipe_i, sb, strlen(sb));
    return;
    }
/* Commands for the rotator */

if ( ROT_flag == TRUE )
{
if ( !strncmp("RG",pip_command,2) )
{
write(rot_serport, pip_command, strlen(pip_command));
write(rot_serport, &CR,1);
rot_ok = FALSE;
rot_pos = -99999;
return;
}
if ( !strncmp("RP",pip_command,2) )
{
sprintf(sb,"%d\r\n",rot_pos);
write(alfosc_serial_pipe_i, sb, strlen(sb) );
return;
}
if ( !strncmp("Rq",pip_command,2) )
{
write(rot_serport,pip_command,strlen(pip_command) );
rot_ok = FALSE;
rot_pos = -99999;
return;
}
if ( !strncmp("r",pip_command,1) )
{
if (rot_ok) strcpy(sb,"ry\r\n");
else
strcpy(sb,"rn\r\n");
write(alfosc_serial_pipe_i, sb, strlen(sb) );
return;
}
}
/* Commands for the Filter and shutter unit (FASU) */
/* FASU filterwheel #1, physically called "filter A" */

if ( FASU_flag == TRUE )
{

if ( !strncmp("UP",pip_command,2) )
{
sprintf(sb,"%d\r\n",fasu_upos);
write(alfosc_serial_pipe_i, sb, strlen(sb));
return;
}
if ( !strncmp("Uq",pip_command,2) )
{
strcpy(sb,"Aq");
write(fasu_serport, sb, strlen(sb));
fasu_uok = FALSE;
fasu_upos = -99999;
return;
}
if ( !strncmp("u",pip_command,1) )
{
if (fasu_uok) strcpy(sb,"uy\r\n");
else
```

```
strcpy(sb,"un\r\n");
write(alfosc_serial_pipe_i, sb, strlen(sb));
return;
}
if ( !strncmp("U",pip_command,1) )
{
sprintf(sb,"A%c",pip_command[1]);
write(fasu_serport, sb, strlen(sb));
write(fasu_serport,&CR,1);
fasu_uok = FALSE;
fasu_upos = -99999;
return;
}
/* Commands for FASU filterwheel #2 */
if ( !strncmp("VP",pip_command,2) )
         {
         sprintf(sb,"%d\r\n",fasu_vpos);
         write(alfosc_serial_pipe_i, sb, strlen(sb));
         return;
         }
if ( !strncmp("Vq",pip_command,2) )
         {
         strcpy(sb,"Bq");
         write(fasu_serport, sb, strlen(sb));
         fasu_vok = FALSE;
         fasu_vpos = -99999;
         return;
         }
if ( !strncmp("v",pip_command,1) )
         {
         if (fasu_vok) strcpy(sb,"vy\r\n");
         else
         strcpy(sb,"vn\r\n");
         write(alfosc_serial_pipe_i, sb, strlen(sb));
         return;
         }
if ( !strncmp("V",pip_command,1) )
         {
         sprintf(sb,"B%c",pip_command[1]);
         write(fasu_serport, sb, strlen(sb));
         write(fasu_serport,&CR,1);
         fasu_vok = FALSE;
         fasu_vpos = -99999;
         return;
         }
} /* end FASU_flag == TRUE */
}
/************************************************************/
void decode_ser_command_alfosc(void)
{

if ( ser_command_alfosc[1] == 'G' ) return;
if ( ser_command_alfosc[1] == 'I' ) return;

/*fprintf(stdout,"ser: [%s]\n",ser_command);fflush(stdout);*/

/* Aperture movement strings  */
if ( !strncmp("ay",ser_command_alfosc,2) )
{
alfosc_aok = TRUE;
return;
}
if ( !strncmp("an",ser_command_alfosc,2) )
{
alfosc_apos = -99999;
alfosc_aok = FALSE;
return;
}
```

```
if ( !strncmp("AP",ser_command_alfosc,2) )
{
if ( ser_command_alfosc[2] == '?' )
{
if ( alfosc_aok == TRUE )
{
/*strcpy(sb,"AI");
write(serport,sb,2);
fprintf(stderr,"Aperture is being initialized\r\n");*/
alfosc_apos = -99999;
}
else alfosc_aok = FALSE;
}
else
alfosc_apos = atoi(&ser_command_alfosc[2]);
return;
}

/* Filter movement strings  */
if ( !strncmp("fy",ser_command_alfosc,2) )
    {
    alfosc_fok = TRUE;
return;
    }
if ( !strncmp("fn",ser_command_alfosc,2) )
    {
alfosc_fpos = -99999;
    alfosc_fok = FALSE;
return;
    }
if ( !strncmp("FP",ser_command_alfosc,2) )
    {
    if ( ser_command_alfosc[2] == '?' )
        {
        if ( alfosc_fok == TRUE )
{
/*strcpy(sb,"FI");
write(serport,sb,2);
fprintf(stderr,"Filter is being initialized\r\n");*/
alfosc_fpos = -99999;
}
        else alfosc_fok = FALSE;
        }
else
     alfosc_fpos = atoi(&ser_command_alfosc[2]);
return;
    }

/* Grism strings */
if ( !strncmp("gy",ser_command_alfosc,2) )
    {
    alfosc_gok = TRUE;
return;
    }
if ( !strncmp("gn",ser_command_alfosc,2) )
    {
alfosc_gpos = -99999;
    alfosc_gok = FALSE;
return;
    }
if ( !strncmp("GP",ser_command_alfosc,2) )
    {
    if ( ser_command_alfosc[2] == '?' )
        {
        if ( alfosc_gok == TRUE )
{
/*strcpy(sb,"GI");
write(serport,sb,2);
```

```
fprintf(stderr,"Grism is being initialized\r\n");*/
alfosc_gpos = -99999;
}
        else alfosc_gok = FALSE;
        }
else
     alfosc_gpos = atoi(&ser_command_alfosc[2]);
return;
    }

/* Focus  strings */
if ( !strncmp("cy",ser_command_alfosc,2) )
    {
    alfosc_cok = TRUE;
return;
    }
if ( !strncmp("cn",ser_command_alfosc,2) )
    {
alfosc_cpos = -99999;
    alfosc_cok = FALSE;
return;
    }
if ( !strncmp("CP",ser_command_alfosc,2) )
    {
    if ( ser_command_alfosc[2] == '?' )
        {
        if ( alfosc_cok == TRUE )
{
/*strcpy(sb,"CI");
write(serport,sb,2);
fprintf(stderr,"Focus is being initialized\r\n");*/
alfosc_cpos = -99999;
}
        else alfosc_cok = FALSE;
        }
else
     alfosc_cpos = atoi(&ser_command_alfosc[2]);
return;
    }

/* Shutter commands */
if ( !strncmp("sy",ser_command_alfosc,2) )
    {
    alfosc_sok = TRUE;
return;
    }
if ( !strncmp("sn",ser_command_alfosc,2) )
    {
alfosc_spos = -99999;
    alfosc_sok = FALSE;
return;
    }
if ( !strncmp("SP",ser_command_alfosc,2) )
    {
    if ( ser_command_alfosc[2] == '?')
        {
        if ( alfosc_sok == TRUE )
{
/*strcpy(sb,"SI");
write(serport,sb,2);
fprintf(stderr,"Shutter is being initialized\r\n");*/
alfosc_spos = -99999;
}
        else alfosc_sok = FALSE;
        }
else
{
     alfosc_spos = atoi(&ser_command_alfosc[2]);
```

```
if ( ( alfosc_spos == 0 ) || ( alfosc_spos == 4000 ) )
alfosc_spos = 0;
else if ( ( alfosc_spos == 2000 ) || ( alfosc_spos == 6000 ) )
alfosc_spos = 1;
else alfosc_spos = -99999;
}
return;
    }

}
/************************************************************/
void decode_ser_command_rotator(void)
{

if ( ser_command_rotator[1] == 'G' ) return;
if ( ser_command_rotator[1] == 'I' ) return;

/*fprintf(stdout,"ser: [%s]\n",ser_command);fflush(stdout);*/

/* Rotator commands */
if ( !strncmp("ry",ser_command_rotator,2) )
    {
    rot_ok = TRUE;
    return;
    }
if ( !strncmp("rn",ser_command_rotator,2) )
    {
    rot_pos = -99999;
    rot_ok = FALSE;
    return;
    }
if ( !strncmp("RP",ser_command_rotator,2) )
    {
    if ( ser_command_rotator[2] == '?' )
        {
        if ( rot_ok == TRUE )
            {
            fprintf(stderr,"Rotator failure...\r\n");
            rot_pos = -99999;
            }
        else rot_ok = FALSE;
        }
    else
        rot_pos = atoi(&ser_command_rotator[2]);
    return;
    }
}
/************************************************************/
void decode_ser_command_fasu(void)
{

if ( ser_command_fasu[1] == 'G' ) return;
if ( ser_command_fasu[1] == 'I' ) return;

/*fprintf(stdout,"ser: [%s]\n",ser_command);fflush(stdout);*/
/* FASU commands, wheel 'U' */
if ( !strncmp("ay",ser_command_fasu,2) )
    {
    fasu_uok = TRUE;
    return;
    }
if ( !strncmp("an",ser_command_fasu,2) )
    {
    fasu_upos = -99999;
    fasu_uok = FALSE;
    return;
    }
if ( !strncmp("AP",ser_command_fasu,2) )
```

```
    {
    if ( ser_command_fasu[2] == '?' )
        {
        if (fasu_uok == TRUE )
            {
            fprintf(stderr,"FASU wheel 'u' failure...\r\n");
            fasu_upos = -99999;
            }
        else fasu_uok = FALSE;
        }
    else
        fasu_upos = atoi(&ser_command_fasu[2]);
    return;
    }

/* FASU commands, wheel 'V' */
if ( !strncmp("by",ser_command_fasu,2) )
    {
    fasu_vok = TRUE;
    return;
    }
if ( !strncmp("bn",ser_command_fasu,2) )
    {
    fasu_vpos = -99999;
    fasu_vok = FALSE;
    return;
    }
if ( !strncmp("BP",ser_command_fasu,2) )
    {
    if ( ser_command_fasu[2] == '?' )
        {
        if (fasu_vok == TRUE )
            {
            fprintf(stderr,"FASU wheel 'V' failure...\r\n");
            fasu_vpos = -99999;
            }
        else fasu_vok = FALSE;
        }
    else
        fasu_vpos = atoi(&ser_command_fasu[2]);
    return;
    }

}
/***************************************************************/
```

6.2 Makefile.

```
alfosc_ser_daemon :  alfosc_ser_daemon.c
gcc -O2 -m486 -finline -pedantic -Wall -Wstrict-prototypes -o alfosc_ser_daemon alfosc_ser_daemon.c

cp alfosc_ser_daemon ../io/alfosc_ser_daemon
```