```
# Microsoft Visual C++ generated build script - Do not modify

PROJ = MOTORS
DEBUG = 0
PROGTYPE = 6
CALLER =
ARGS =
DLLS =
D_RCDEFINES = -d_DEBUG
R_RCDEFINES = -dNDEBUG
ORIGIN = MSVC
ORIGIN_VER = 1.00
PROJPATH = C:\MSVC\MFC\ECHELLE\MOTOR\
USEMFC = 1
CC = cl
CPP = cl
CXX = cl
CCREATEPCHFLAG =
CPPCREATEPCHFLAG =
CUSEPCHFLAG =
CPPUSEPCHFLAG =
FIRSTC = TIMER.C
FIRSTCPP = MOTORS.CPP
RC = rc
CFLAGS_D_DEXE = /nologo /f /G2 /W3 /Zi /AM /Od /D "_DEBUG" /D "_DOS" /Fr /Fd"MOTORS.PD
B"
CFLAGS_R_DEXE = /nologo /Gs /G3 /W3 /AM /O1 /Ox /D "NDEBUG" /D "_DOS"
LFLAGS_D_DEXE = /NOLOGO /NOE /STACK:20480 /NOPACKF /EXEPACK /ONERROR:NOEXE /CO
LFLAGS_R_DEXE = /NOLOGO /NOI /STACK:5120 /ONERROR:NOEXE
LIBS_D_DEXE = mafxcrd mafxcr oldnames is_cm async
LIBS_R_DEXE = mafxcr oldnames is_cm async
RCFLAGS = /nologo
RESFLAGS = /nologo
RUNFLAGS =
OBJS_EXT =
LIBS_EXT = INSTEP\IS_CM.LIB
!if "$(DEBUG)" == "1"
CFLAGS = $(CFLAGS_D_DEXE)
LFLAGS = $(LFLAGS_D_DEXE)
LIBS = $(LIBS_D_DEXE)
MAPFILE = nul
RCDEFINES = $(D_RCDEFINES)
!else
CFLAGS = $(CFLAGS_R_DEXE)
LFLAGS = $(LFLAGS_R_DEXE)
LIBS = $(LIBS_R_DEXE)
MAPFILE = nul
RCDEFINES = $(R_RCDEFINES)
!endif
!if [if exist MSVC.BND del MSVC.BND]
!endif
SBRS = MOTORS.SBR \
    KEYBOARD.SBR \
    CTIME.SBR \
    CSERIAL.SBR \
    TIMER.SBR \
    ERROR.SBR \
    CSYSTEM.SBR \
    CMTROBL.SBR \
    STEPPER.SBR \
    DISPTCH2.SBR \
    OBJSAVE.SBR \
    CPIO.SBR \
    CLAMP.SBR \
    CERIAL.SBR \
    CFLTRDOS.SBR \
    VACUUM1.SBR \
    ION.SBR


IS_CM_DEP =

MOTORS_DEP = c:\msvc\mfc\echelle\motor\main.hpp \
  c:\msvc\mfc\echelle\motor\classes.hpp \
  c:\msvc\mfc\echelle\motor\proto.hpp \
  c:\msvc\mfc\echelle\e_vaccum\stdafx.h \
  c:\msvc\mfc\echelle\motor\cserial.hpp \
  c:\msvc\mfc\echelle\motor\cerror.hpp \
  c:\msvc\mfc\echelle\motor\opti1.hpp \
  c:\msvc\mfc\echelle\motor\ciocard.hpp \
  c:\msvc\mfc\echelle\motor\cfilter.hpp \
  c:\msvc\mfc\echelle\motor\ctimer.hpp \
  c:\msvc\mfc\echelle\motor\cmtrobl.hpp \
  c:\msvc\mfc\echelle\motor\optiparm.hpp \
  c:\msvc\mfc\echelle\motor\async.hpp \
  c:\msvc\mfc\echelle\motor\dispatch.hpp \
  c:\msvc\mfc\echelle\motor\csystem.hpp \
  c:\msvc\mfc\echelle\motor\vacuum.hpp \
  c:\msvc\mfc\echelle\motor\objsave.hpp \
  c:\msvc\mfc\echelle\motor\cpio.hpp \
  c:\msvc\mfc\echelle\motor\clamp.hpp \
  c:\msvc\mfc\echelle\motor\ion.hpp \
  c:\msvc\mfc\echelle\motor\serial.h \
  c:\msvc\mfc\echelle\motor\timer.h


KEYBOARD_DEP = c:\msvc\mfc\echelle\e_vaccum\stdafx.h \
  c:\msvc\mfc\echelle\motor\dispatch.hpp \
  c:\msvc\mfc\echelle\motor\proto.hpp


CTIME_DEP = c:\msvc\mfc\echelle\e_vaccum\stdafx.h \
  c:\msvc\mfc\echelle\motor\ctimer.hpp


CSERIAL_DEP = c:\msvc\mfc\echelle\e_vaccum\stdafx.h \
  c:\msvc\mfc\echelle\motor\cserial.hpp \
  c:\msvc\mfc\echelle\motor\async.hpp \
  c:\msvc\mfc\echelle\motor\dispatch.hpp \
  c:\msvc\mfc\echelle\motor\serial.h


TIMER_DEP = c:\msvc\mfc\echelle\motor\timer.h \
  c:\msvc\mfc\echelle\motor\interrup.h


ERROR_DEP = c:\msvc\mfc\echelle\e_vaccum\stdafx.h \
  c:\msvc\mfc\echelle\motor\optiparm.hpp \
  c:\msvc\mfc\echelle\motor\dispatch.hpp \
  c:\msvc\mfc\echelle\motor\cerror.hpp


CSYSTEM_DEP = c:\msvc\mfc\echelle\e_vaccum\stdafx.h \
  c:\msvc\mfc\echelle\motor\csystem.hpp \
  c:\msvc\mfc\echelle\motor\opti1.hpp \
  c:\msvc\mfc\echelle\motor\ciocard.hpp \
  c:\msvc\mfc\echelle\motor\cfilter.hpp \
  c:\msvc\mfc\echelle\motor\ctimer.hpp \
  c:\msvc\mfc\echelle\motor\cmtrobl.hpp \
  c:\msvc\mfc\echelle\motor\dispatch.hpp \
```

```
  c:\msvc\mfc\echelle\motor\vacuum.hpp \
  c:\msvc\mfc\echelle\motor\objsave.hpp \
  c:\msvc\mfc\echelle\motor\clamp.hpp \
  c:\msvc\mfc\echelle\motor\ion.hpp \
  c:\msvc\mfc\echelle\motor\pointers.hpp


CMTROBL_DEP = c:\msvc\mfc\echelle\e_vaccum\stdafx.h \
  c:\msvc\mfc\echelle\motor\opti1.hpp \
  c:\msvc\mfc\echelle\motor\ciocard.hpp \
  c:\msvc\mfc\echelle\motor\cfilter.hpp \
  c:\msvc\mfc\echelle\motor\ctimer.hpp \
  c:\msvc\mfc\echelle\motor\cmtrobl.hpp \
  c:\msvc\mfc\echelle\motor\dispatch.hpp \
  c:\msvc\mfc\echelle\motor\objsave.hpp \
  c:\msvc\mfc\echelle\motor\pointers.hpp


STEPPER_DEP = c:\msvc\mfc\echelle\e_vaccum\stdafx.h \
  c:\msvc\mfc\echelle\motor\optiparm.hpp \
  c:\msvc\mfc\echelle\motor\opti1.hpp \
  c:\msvc\mfc\echelle\motor\ciocard.hpp \
  c:\msvc\mfc\echelle\motor\is_cm.h


DISPTCH2_DEP = c:\msvc\mfc\echelle\e_vaccum\stdafx.h \
  c:\msvc\mfc\echelle\motor\optiparm.hpp \
  c:\msvc\mfc\echelle\motor\cserial.hpp \
  c:\msvc\mfc\echelle\motor\csystem.hpp \
  c:\msvc\mfc\echelle\motor\opti1.hpp \
  c:\msvc\mfc\echelle\motor\ciocard.hpp \
  c:\msvc\mfc\echelle\motor\cfilter.hpp \
  c:\msvc\mfc\echelle\motor\ctimer.hpp \
  c:\msvc\mfc\echelle\motor\cerror.hpp \
  c:\msvc\mfc\echelle\motor\cmtrobl.hpp \
  c:\msvc\mfc\echelle\motor\dispatch.hpp \
  c:\msvc\mfc\echelle\motor\vacuum.hpp \
  c:\msvc\mfc\echelle\motor\clamp.hpp \
  c:\msvc\mfc\echelle\motor\objsave.hpp \
  c:\msvc\mfc\echelle\motor\cpio.hpp \
  c:\msvc\mfc\echelle\motor\ion.hpp \
  c:\msvc\mfc\echelle\motor\pointers.hpp


OBJSAVE_DEP = c:\msvc\mfc\echelle\e_vaccum\stdafx.h \
  c:\msvc\mfc\echelle\motor\opti1.hpp \
  c:\msvc\mfc\echelle\motor\ciocard.hpp \
  c:\msvc\mfc\echelle\motor\cfilter.hpp \
  c:\msvc\mfc\echelle\motor\ctimer.hpp \
  c:\msvc\mfc\echelle\motor\cmtrobl.hpp \
  c:\msvc\mfc\echelle\motor\dispatch.hpp \
  c:\msvc\mfc\echelle\motor\objsave.hpp \
  c:\msvc\mfc\echelle\motor\pointers.hpp


CPIO_DEP = c:\msvc\mfc\echelle\e_vaccum\stdafx.h \
  c:\msvc\mfc\echelle\motor\cpio.hpp


CLAMP_DEP = c:\msvc\mfc\echelle\e_vaccum\stdafx.h \
  c:\msvc\mfc\echelle\motor\cpio.hpp \
  c:\msvc\mfc\echelle\motor\clamp.hpp


CERIAL_DEP = c:\msvc\mfc\echelle\motor\ctimer.hpp \
  c:\msvc\mfc\echelle\e_vaccum\stdafx.h \
  c:\msvc\mfc\echelle\motor\serial.h \
  c:\msvc\mfc\echelle\motor\timer.h


CFLTRDOS_DEP = c:\msvc\mfc\echelle\e_vaccum\stdafx.h \
  c:\msvc\mfc\echelle\motor\optiparm.hpp \
  c:\msvc\mfc\echelle\motor\optistep.hpp \
  c:\msvc\mfc\echelle\motor\opti1.hpp \
  c:\msvc\mfc\echelle\motor\ciocard.hpp \
  c:\msvc\mfc\echelle\motor\dispatch.hpp \
  c:\msvc\mfc\echelle\motor\cerror.hpp \
  c:\msvc\mfc\echelle\motor\cpio.hpp \
  c:\msvc\mfc\echelle\motor\cfilter.hpp \
  c:\msvc\mfc\echelle\motor\ctimer.hpp \
  c:\msvc\mfc\echelle\motor\winmo3\ccalibd.h


VACUUM1_DEP = c:\msvc\mfc\echelle\e_vaccum\stdafx.h \
  c:\msvc\mfc\echelle\motor\cpio.hpp \
  c:\msvc\mfc\echelle\motor\vacuum.hpp


ION_DEP = c:\msvc\mfc\echelle\e_vaccum\stdafx.h \
  c:\msvc\mfc\echelle\motor\cpio.hpp \
  c:\msvc\mfc\echelle\motor\ion.hpp


all:  $(PROJ).EXE

MOTORS.OBJ: MOTORS.CPP $(MOTORS_DEP)
  $(CPP) $(CFLAGS) $(CPPCREATEPCHFLAG) /c MOTORS.CPP

KEYBOARD.OBJ: KEYBOARD.CPP $(KEYBOARD_DEP)
  $(CPP) $(CFLAGS) $(CPPUSEPCHFLAG) /c KEYBOARD.CPP

CTIME.OBJ:  CTIME.CPP $(CTIME_DEP)
  $(CPP) $(CFLAGS) $(CPPUSEPCHFLAG) /c CTIME.CPP

CSERIAL.OBJ:  CSERIAL.CPP $(CSERIAL_DEP)
  $(CPP) $(CFLAGS) $(CPPUSEPCHFLAG) /c CSERIAL.CPP

TIMER.OBJ:  TIMER.C $(TIMER_DEP)
  $(CC) $(CFLAGS) $(CCREATEPCHFLAG) /c TIMER.C

ERROR.OBJ:  ERROR.CPP $(ERROR_DEP)
  $(CPP) $(CFLAGS) $(CPPUSEPCHFLAG) /c ERROR.CPP

CSYSTEM.OBJ:  CSYSTEM.CPP $(CSYSTEM_DEP)
  $(CPP) $(CFLAGS) $(CPPUSEPCHFLAG) /c CSYSTEM.CPP

CMTROBL.OBJ:  CMTROBL.CPP $(CMTROBL_DEP)
  $(CPP) $(CFLAGS) $(CPPUSEPCHFLAG) /c CMTROBL.CPP

STEPPER.OBJ:  STEPPER.CPP $(STEPPER_DEP)
  $(CPP) $(CFLAGS) $(CPPUSEPCHFLAG) /c STEPPER.CPP

DISPTCH2.OBJ: DISPTCH2.CPP $(DISPTCH2_DEP)
  $(CPP) $(CFLAGS) $(CPPUSEPCHFLAG) /c DISPTCH2.CPP

OBJSAVE.OBJ:  OBJSAVE.CPP $(OBJSAVE_DEP)
  $(CPP) $(CFLAGS) $(CPPUSEPCHFLAG) /c OBJSAVE.CPP

CPIO.OBJ: CPIO.CPP $(CPIO_DEP)
  $(CPP) $(CFLAGS) $(CPPUSEPCHFLAG) /c CPIO.CPP
```

```
CLAMP.OBJ:  CLAMP.CPP $(CLAMP_DEP)
  $(CPP) $(CFLAGS) $(CPPUSEPCHFLAG) /c CLAMP.CPP

CERIAL.OBJ: CERIAL.CPP $(CERIAL_DEP)
  $(CPP) $(CFLAGS) $(CPPUSEPCHFLAG) /c CERIAL.CPP

CFLTRDOS.OBJ: CFLTRDOS.CPP $(CFLTRDOS_DEP)
  $(CPP) $(CFLAGS) $(CPPUSEPCHFLAG) /c CFLTRDOS.CPP

VACUUM1.OBJ:  VACUUM1.CPP $(VACUUM1_DEP)
  $(CPP) $(CFLAGS) $(CPPUSEPCHFLAG) /c VACUUM1.CPP

ION.OBJ:  ION.CPP $(ION_DEP)
  $(CPP) $(CFLAGS) $(CPPUSEPCHFLAG) /c ION.CPP

$(PROJ).EXE:: MOTORS.OBJ KEYBOARD.OBJ CTIME.OBJ CSERIAL.OBJ TIMER.OBJ ERROR.OBJ CSYSTE
M.OBJ \
  CMTROBL.OBJ STEPPER.OBJ DISPTCH2.OBJ OBJSAVE.OBJ CPIO.OBJ CLAMP.OBJ CERIAL.OBJ CFLTR
DOS.OBJ \
  VACUUM1.OBJ ION.OBJ $(OBJS_EXT) $(DEFFILE)
  echo >NUL @<<$(PROJ).CRF
MOTORS.OBJ +
KEYBOARD.OBJ +
CTIME.OBJ +
CSERIAL.OBJ +
TIMER.OBJ +
ERROR.OBJ +
CSYSTEM.OBJ +
CMTROBL.OBJ +
STEPPER.OBJ +
DISPTCH2.OBJ +
OBJSAVE.OBJ +
CPIO.OBJ +
CLAMP.OBJ +
CERIAL.OBJ +
CFLTRDOS.OBJ +
VACUUM1.OBJ +
ION.OBJ +
$(OBJS_EXT)
$(PROJ).EXE
$(MAPFILE)
c:\msvc\lib\+
c:\msvc\mfc\lib\+
c:\msvc\mfc\hpc1\runtime\libs\+
c:\msvc\mfc\echelle\motor\instep\+
INSTEP\IS_CM.LIB+
$(LIBS)
$(DEFFILE);
<<
  link $(LFLAGS) @$(PROJ).CRF

run: $(PROJ).EXE
  $(PROJ) $(RUNFLAGS)


$(PROJ).BSC: $(SBRS)
  bscmake @<<
/o$@ $(SBRS)
<<
```

```
// CONTENTS : Motor.c -- This file contains the MAIN code for the echelle motor
//                       controllers.

// ***----------------------------------****---------------------------------***
//    SPECS --
//            Operating System: DOS 6.2
//            Language        : C++ (Microsoft Specific)
//            Compiler        : Microsoft Visual C++ 1.5 (medium Lib)
//
//    HISTORY --
//            Creation Date   : June 21, 1994
//            Edit Dates      : Aug 31, 1994 - added OBJList for referencing
//                                             objects
//                            :              - added 3 more motors to list
//                                             (gTip, gTilt, gGrating)
//                            : Sept 1, 1994 - removed CObList. Didn't work!
//                            :              - added cMtrObjL class (array of
//                                             motor object pointers)
//                            : Sept 8, 1994 - removed all timer interrupt
//                                             functions
//                            : Sept 9, 1994 - renamed motors reassigning to
//                                             final configuration
//                            : Sept 15      - added cVacuum class (reads
//                                             vacuum sensors)
//                            : Oct 30, 1994 - added gSave object...saves
//                                             state of grating, slit, tip,
//                                             tilt and CCD focus motors
//                            : Oct 31, 1994 - added gSave->read()
//                            : Nov 14, 1994 - added console I/O capability
//                            : Jan 23, 1995 - added gPIO pionter to PIO-24
//                            : Mar 23, 1995 - CCD focus obj now axisID = 4
//                            : Apr 04, 1995 - removed cShutter object
//                            : Aug 29, 1996 - addr of all motors = 0x250 **TESTING**
//                            : Sep 03, 1997 - re-addressed motors to proper values (S
ee above)
//                                           - added a filter wheel initial search and
 set in main()
//                            : 24 Sep 97 - added calibOut to startup in main() as per
 Shu-i's request
//                            : 25 Sep 97 - Changed tip/tilt axes to 2 & 3 from 1 & 2;
 axes taken from winmo3,
//                                          which works
//                            : 10 Oct 97 - Added command ME to keyboard mode (i.e. co
mmand tested in keyboard
//                                          routine, not in Dispatcher/Router
//                                          ME invokes a menu for moving motors (see m
enu() in keyboard.cpp)
//                                        - Added malloc() call for char *buffer for k
eyboard input;
//                                          just to be safe and not slowly overwrite D
OS. :)
//                            : 1 Apr 98  - Added vacuum auto-logging timer
//                            : 12 May 98 - Added cIon calls to main()
//
//            Author          : TjM (University of Chicago, Yerkes Obs)
//
// ***----------------------------------****---------------------------------***

#include <stdio.h>
#include <stdlib.h>         // included for itoa function
#include "main.hpp"

int errCode, rtnCode, stopFlag=0;

// ***-------------------------------------------------***
// -----> Define the pointters for the objects       <-----
// -----> that will be used in the motor software.   <-----
// ***-------------------------------------------------***

cSystem* gSystem;           // pointer to cSystem object

COptiStep* gOptiOne;        // pointer to OptiStep board #1
COptiStep* gOptiTwo;        // pointer to OptiStep board #2
COptiStep* gOptiThree;      // pointer to OptiStep board #3
COptiStep* gOptiFour;       // pointer to OptiStep board #4

cSerial* gIOPort;           // pointer to serial port

cDispatcher* gZephyr;       // pointer to cDispatcher object

cFilterWhl*   gEFilter;     // echelle filter wheel object pointer
cFilterWhl*   gCFilter;     // guide camera filter wheel pointer
cSlit*        gSlit;        // slit focus pointer
cTipTilt*     gTip;         // CCD tip motor pointer
cTipTilt*     gTilt;        // CCD tilt motor pointer
cGrating*     gGrating;     // grating tilt motor pointer
cCalibration* gCalMirror;   // calibration folding mirror (just needs to move)
cCCDfocus*    gCCD;         // CCD focus motor motors pointer

cMtrObL       *gList;       // object list object pointer
cVacuum       *gVac;        // object pointer to the vacuum sensors
cObjSave      *gSave;       // object pointer for saving motor positions
cError        *gError;      // obj pointer to error trapping

cPIO          *gPIO;        // obj pointer to PIO-24 i/o card
cLamp         *gLamp;       // obj pointer to Lamp Sources
cIon          *gIon;        // obj pointer to Ion pump control

void main(int argc, char *argv[]) {
  CString arg;
  int DEBUG=0;

  cTimer  *vacTime;         // pointer to local cTimer class object
  vacTime = new cTimer();

// Parse command-line opts, if any
  if(argc > 1) {
    arg = (const char *) argv[1];
    if(arg == "/d") {
      DEBUG=1;
      }
    }

// ***-----------------------------------------------------------------------***
// ----->               Start allocating the memory for the            <-----
// ----->               objects used in the motor software.            <-----
// ***-----------------------------------------------------------------------***
//  signonBanner();   // print signon banner to monitor only

  gIOPort = new cSerial(SECOND);    // open the port
  printf("Init serial port\n");
  gError = new cError;              // instantiate error object
  printf("Init error handler\n");
  gZephyr = new cDispatcher();      // instantiate dispatcher object
  printf("Init dispatcher\n");
  gSave = new cObjSave();           // instantiate Save objject
  printf("Init file I/O\n");

  gOptiOne =  new COptiStep(0x250); // instantiate OptiOne (board #1) object
  printf("Init Opti1\n");
```

```
// ******************************************************************************
// *** Now instantiate the motors that belong to gOptiOne *********************
// ******************************************************************************

  gCalMirror = new cCalibration(1,1);         // calibration motor object
  printf("\tInit cal. flat\n");
  gEFilter = new cFilterWhl(2,1,"EchFltr");   // Echelle filter motor object
  printf("\tInit Echelle filter\n");
  gCFilter = new cFilterWhl(3,1,"GCFiltr");   // Guide Camera filter motor obj
  printf("\tInit guide filter\n");

  gOptiTwo =  new COptiStep(0x260);           // OptiTwo (board #2) object
  printf("Init Opti2\n");

// ******************************************************************************
// *** Now instantiate the motors that belong to gOptiTwo *********************
// ******************************************************************************

  gGrating = new cGrating(3,2);               // Grating motor object
  printf("\tInit grating\n");
  gSlit = new cSlit(1,2);                     // Slit motor object
  printf("\tInit slit\n");

  gOptiThree =  new COptiStep(0x270);         // OptiThree (board #3) obj
  printf("Init Opti3\n");

// ******************************************************************************
// *** Now instantiate the motors that belong to gOptiThree *******************
// ******************************************************************************

  gTip = new cTipTilt(2,3, "TipMotr");          // Tip motor object
  printf("\tInit tip\n");
  gTilt = new cTipTilt(3,3,"TiltMtr");        // Tilt motor object
  printf("\tInit tilt\n");

  gOptiFour =  new COptiStep(0x280);          // OptiTwo (board #4) obj
  printf("Init Opti4\n");

// ******************************************************************************
// *** Now instantiate the motors that belong to gOptiFour ********************
// ******************************************************************************

  gCCD = new cCCDfocus(4,4);                  // CCD focus motor object
  printf("\tInit CCD (axis 1&2)\n");

// ******************************************************************************
// *** Now instantiate the rest of the objects used **************************
// ******************************************************************************

  gList = new cMtrObL;                        // the motor list object
  printf("Init object list\n");
  gPIO = new cPIO;                            // PIO-24 Board object
  printf("Init PIO\n");
  gVac = new cVacuum;                         // Vacuum sensors object
  printf("Init vacuum\n");
  gLamp = new cLamp;                          // Lamp Source object
  printf("Init lamps\n");
  gIon = new cIon;
  printf("Init Ion pump\n");
  gSave->read();                              // read the last state of system
  printf("read in last position\n");

  keyflush();                                 // flush the keyboard buffer
  printf("flush keyboard\n");
```

```
  initserialvect();                             // install the serial port
                                                // hardware intecept vectors
  printf("redirect serial interrupt\n");


// ******************************************************************************
// *** The MAIN EVENT LOOP: Exit only on reset of system or ESC char **********
// *** from the keyboard...                                          **********
// ******************************************************************************

  CString T;
  char *buffer;
  long temp;

  buffer = (char *) malloc(1024*sizeof(char));

  if(!DEBUG) {
// find the first position and then move to last known position of
// both filter wheels

    printf("\n\n\r Finding the last known filter positions....Please standby.\n\r");

    gOptiOne->cCardInit();
    temp = gEFilter->position;
    gEFilter->findHome();
    gEFilter->cMoveFilter(temp);
    printf("\tFound Echelle filter position; finding guide camera position\n");
    temp = gCFilter->position;
    gCFilter->findHome();
    gCFilter->cMoveFilter(temp);

    gZephyr->lastOS = 1;  // set the last OS board we looked at for future moves

// Move calibration mirror out, just to be safe.
    printf("\n\r Moving Calibration Flat out of beam path.  Please wait.\n\r");
    gCalMirror->calibOut();
  }
  else {
    printf("\nDEBUG MODE - Filter wheel/cal flat setup move skipped\n");
    }

  signonBanner();


  vacTime->cStartTimer();               // start the timer for the vacuum check

  while(stopFlag == 0) {                        // This is the main event loop

    if(chkforESC()) {
      stopFlag = 1;
      continue;
      }

//    if(!stopFlag)
//      stopFlag = menu();

    if(Ctrl_k()) {
      printf("KEYBOARD MODE...Enter Command\r\n>");  // print message
      gets(buffer);               // read the command
      T = buffer;
      if(!T.CompareNoCase("ME")) {      // For debugging in lab
        while(menu() != 1)
          ;
        printf("OK\n");
```

```
        }                          // end of if(!T.C...
      else {
        T+="\r";
        gZephyr->parser(T);            // send string to parser
        }
    } // end of if(Ctrl_k) statement

   if(gIOPort->readThePort()) {
    gZephyr->parser(gIOPort->io_str);
   }

   // now check the vacuum status, every 30 seconds
   // this also will do the logging
    if(vacTime->cChkTimer(30)) {
        if(gVac->is_vac_on[1] || gVac->is_vac_on[0]) {
          gVac->readSensors();  // This call will write the data to a file
          if(gVac->pressure1>0.004) {
            gVac->HiVacOff();   //shut off Cold Cathode sensor
            printf("\nHi Vac shutdown - reading higher than 4e-3 Torr!!\n");
            }
          if(gVac->pressure0>700) {
           // shut off the Pirani gauge
            gVac->VacOff();
            printf("\nTotal vac shutdown - Pirani reads higher than 700 Torr!!\n");
            }
        }
        vacTime->cStartTimer();
      } // end of if(vacTime->cChkTimer(30)).... statement

   // printf("POST_VACCHK: %s\n", gIOPort->io_str);

  } /* end of while loop */

//-----------------------------------****----------------------------------------
// Turn off power to everything...This will ensure that the next time the
// program is booted there is less likelihood of a voltage spike.

//-----------------------------------****----------------------------------------

  gPIO->writePort('A', 0xFF);
  gPIO->writePort('B', 0xFF);
  printf("All power shutdown.\n");


  restore_serialint();                // restore the interrupt vector to the
  printf("Restored serial interrupt vector.\n");  // original configuration
}
```

```
// CONTENTS : Keyboard.cpp -- This file has keyboard I/O functions
//                            for the PC. These functions are ANSI C
//                            but the comments are C++. Therefore the
//                            file must be compiled using C++ compiler.

// ***----------------------------------***-----------------------------***
//    SPECS --
//            Operating System: DOS 6.2
//            Language        : C++ (Microsoft Specific)
//            Compiler        : Microsoft Visual C++ 1.5 (medium Lib)
//
//    HISTORY --
//            Creation Date   : June 20, 1994
//            Edit Dates      : Nov 8, 1994   - added int Ctrl_k()
//                            : May 3, 1995   - added menu() function
//                            : Apr 29, 1996  - made Windows compatable
//                            : 10 Oct 97 - Changed version number to 1.40
//                                        - Updated menu routine for use in debug
//                                        - Added cr2lf(CString in) to translate \r to
 \n
//
//            Author          : TjM (University of Chicago, Yerkes Obs)
//
// ***----------------------------------***-----------------------------***
//

#if defined(_WINDOWS)
#include "stdafx.h"
#else
#include <afx.h>
#endif

#include <conio.h>    // for keyboard entry
#include <stdio.h>    // for the getch() function
#include <graph.h>    // for _clearscreen call
#include "dispatch.hpp" // for gZephyr references
#include "proto.hpp"  // Prototypes for keyboard routines

extern int stopFlag;  // defined in "global.h"
extern cDispatcher* gZephyr;

#define TRUE 1
#define FALSE 0

// =====================================================
//        Function keyhit
// =====================================================

int keyhit()
{ return(kbhit());  }



// =====================================================
//        Function getkey
// =====================================================

int getkey () {

 int ch;

 ch = getch();
 if (ch == 0)
  ch = 0x100 + getch();
  return(ch);
 }

// =====================================================
//        Function keyflush
// =====================================================

 void keyflush() {

  while (keyhit() != 0)
    { getkey(); }
}

// =====================================================
//        Function waitcr
// =====================================================

void waitcr() {

  while(getkey() != '\r')
    { ; }
}


// =====================================================
//        Function chkforESC
// =====================================================

// This function checks for a ESC key by the user. If the
// ESC key is hit the global flag stopFlag is set to TRUE.

int chkforESC() {

  int ch = 0x00;

  if(kbhit()) {
    ch = getkey();
    if (ch == 0x01b)      // if ch=ESC set stopFlag to TRUE
      return(TRUE);
    else {
      keyflush();
      return(FALSE);
    }
  }
  return(FALSE);
}

// =====================================================
//        Function Ctrl_k
// =====================================================

// This function checks for an Ctrl k key by the user. If the
// Ctrl k key is hit the function returns TRUE.

int Ctrl_k() {

  int ch = 0x00;

  if(kbhit()) {
    ch = getkey();
    if (ch == 0x00b)      // if ch=ALT k return TRUE
      return(TRUE);
    else {
      keyflush();
      return(FALSE);          // if not return FALSE
```

```
    }
  }
  return(FALSE);
}

// ======================================================


// ======================================================
//        Function menu
// ======================================================

int menu() {

   CString T="  ";
   CString O="  ";
   char *buffer = "";


//   printf("\r\n");
//   printf("\r\n");
//   printf("\nMake your choice...\n");
   printf("\nMOVE MOTOR:    1) Cal Mirror\r\n");
   printf("                2) Echelle Filter\r\n");
   printf("                3) Guider Filter\r\n");
   printf("                4) Grating\r\n");
   printf("                5) Slit Focus\r\n");
   printf("                6) Tip\r\n");
   printf("                7) Tilt\r\n");
   printf("                8) CCD Focus\r\n");
   printf("                9) Status\r\n");
   printf("                0) QUIT\r\n");

   printf("Motor: ");
   scanf("%s",O);

   switch(O[0]) {
    case '1': printf("\r\nEnter Position(0 = Out, 1 = In)) : ");
              scanf("%s", buffer);
              O = buffer;
              T = "mv ";
              T+= O;
              T+= "\r";
              gZephyr->parser(T);        // send string to parser
              _clearscreen(_GCLEARSCREEN);
              return(0);
              break;
    case '2': printf("\r\nEnter Position(1-6) : ");
              scanf("%s", buffer);
              O = buffer;
              T = "mv n ";
              T+= O;
              T+= "\r";
              gZephyr->parser(T);        // send string to parser
              _clearscreen(_GCLEARSCREEN);
              return(0);
              break;
    case '3': printf("\r\nEnter Position(1-6) : ");
              scanf("%s", buffer);
              O = buffer;
              T = "mv n n ";
              T+= O;
              T+= "\r";
              gZephyr->parser(T);        // send string to parser
              _clearscreen(_GCLEARSCREEN);
              return(0);
              break;
    case '4': printf("\r\nEnter Coordinate : ");
              scanf("%s", buffer);
              O = buffer;
              T = "mv n n n ";
              T+= O;
              T+= "\r";
              gZephyr->parser(T);        // send string to parser
              _clearscreen(_GCLEARSCREEN);
              return(0);
              break;
    case '5': printf("\r\nEnter Coordinate : ");
              scanf("%s", buffer);
              O = buffer;
              T = "mv n n n n ";
              T+= O;
              T+="\r";
              gZephyr->parser(T);        // send string to parser
              _clearscreen(_GCLEARSCREEN);
              return(0);
              break;
    case '6': printf("\r\nEnter Coordinate : ");
              scanf("%s", buffer);
              O = buffer;
              T = "mv n n n n n ";
              T+= O;
              T+= "\r";
              gZephyr->parser(T);        // send string to parser
              _clearscreen(_GCLEARSCREEN);
              return(0);
              break;
    case '7': printf("\r\nEnter Coordinate : ");
              scanf("%s", buffer);
              O = buffer;
              T = "mv n n n n n n ";
              T+= O;
              T+= "\r";
              gZephyr->parser(T);        // send string to parser
              _clearscreen(_GCLEARSCREEN);
              return(0);
              break;
    case '8': printf("\r\nEnter Coordinate : ");
              scanf("%s", buffer);
              O = buffer;
              T = "mv n n n n n n n ";
              T+= O;
              T+= "\r";
              gZephyr->parser(T);        // send string to parser
              _clearscreen(_GCLEARSCREEN);
              return(0);
              break;
    case '9': _clearscreen(_GCLEARSCREEN);
              O = "st\r";
              gZephyr->parser(O);        // send string to parser
              return(0);
              break;
    case '0': return(1);                 // sets stopFlag to quit
              break;
    default:  _clearscreen(_GCLEARSCREEN);
              return(0);
              break;
   }// end of switch...
return(0);
}
```

```
void signonBanner(void) {

  _clearscreen(_GCLEARSCREEN);
  printf("\n\n\n======================================================================\
r\n");
  printf("=                                                                   =\r\n");
  printf("=                   ECHELLE MOTION CONTROL                          =\r\n");
  printf("=                                                                   =\r\n");
  printf("=                      Version 1.9.9                                =\r\n");
  printf("=                                                                   =\r\n");
  printf("=                        TjM/PEG                                    =\r\n");
  printf("=                                                                   =\r\n");
  printf("=                 mcmahon@yerkes.uchicago.edu                       =\r\n");
  printf("=                   p-gettings@uchicago.edu                         =\r\n");
  printf("=                                                                   =\r\n");
  printf("=                       May 20, 1998                                =\r\n");
  printf("=                                                                   =\r\n");
  printf("=====================================================================\r\n");

}
```

```
// FILE NAME: cTimer.cpp
//
// CONTENTS : Class cTimer method definitions. cTimer subclass of CTime.

// ***------------------------------***-----------------------------***
//    SPECS --
//            Operating System: DOS 6.2
//            Language        : C++ (Microsoft Specific)
//            Compiler        : Microsoft Visual C++ 1.5 (medium Lib)
//
//    HISTORY --
//            Creation Date   : June 22, 1994
//            Edit Dates      : July 13, 1994 - added timer functions
//                            : July 14, 1994 - changed names of include files to refl
ect C or C++
//                            : Sept 9, 1994  - chnaged cChkTimer to accept float argu
ment
//
//            Author          : TjM (University of Chicago, Yerkes Obs)
//
// ***-----------------------------***-----------------------------***

#if defined(_WINDOWS)
#include "stdafx.h"
#endif
#if defined(_DOS)
#include <afx.h>
#endif

#include <stdio.h>
#include <dos.h>
#include "ctimer.hpp"   // class definitions

cTimer::cTimer() { }

void cTimer::cStartTimer() {

  time(&tThen);

}

int cTimer::cChkTimer(float cTemp)  {      // return 1 if timeout, 0 if not

  time(&tNow);
  elapsed = tNow-tThen;
  if(elapsed >= cTemp)
    return(1);
  else
    return(0);
}


void cTimer::waitNSeconds(float secsToWait) {      // will only do 32 seconds max

    clock_t atStart, now, tdiff;

    atStart = clock();
    tdiff = secsToWait*CLOCKS_PER_SEC;

    now = clock(); // get the ticks

    while( ((now = clock())-atStart) < tdiff ) {
     NULL;
    }
  return;
}

// -----------------------------------------------------
// ------ END of cTimer Class method definitions -------
// -----------------------------------------------------
```

```
// CONTENTS : cSerial -- This file contains the method definitions for
//              the cSerial class.

// ***-------------------------------***-----------------------------***
//    SPECS --
//            Operating System: DOS 6.2
//            Language        : C++ (Microsoft Specific)
//            Compiler        : Microsoft Visual C++ 1.5 (medium Lib)
//
//    HISTORY --
//            Creation Date   : June 21, 1994
//            Edit Dates      : July 13, 1994 - added serial hardware interupts
//                            : July 18, 1994 - added readThePort, writeThePort
//                            :                 and screen method definitions
//                            : July 19, 1994 - added signOn method
//                            : Oct 3,  1994  - methods now talk to dispatcher class
//                            : Aug 1,  1997  - subbed OpenComm for bios calls
//                            : Aug 5,  1997  - OpenCom only for Windows...choose DOS
//                            : 11 May 98 - Added line to readThePort() to add \0 to e
nd
//                                          of io_str.  io_str wasn't terminated befor
e!
//
//            Author          : TjM (University of Chicago, Yerkes Obs)
//
// ***------------------------------***-----------------------------***


#if defined(_WINDOWS)
#include "stdafx.h"
#endif
#if defined(_DOS)
#include <afx.h>
#endif

extern "C" {
#include <stdio.h>
#include <iostream.h>
#include <bios.h>
#include <time.h>
#include <dos.h>
#include <conio.h>
#include "cserial.hpp"  // prototype for the class
#include "async.hpp"    // header file with async definitions
#include "Dispatch.hpp" // dispatcher class definition
#include "serial.h"
}
#define TRUE 1
#define FALSE 0

extern cDispatcher* gZephyr;
extern errCode;
extern rtnCode;

// ===========================================================
// ===== Class cSerial METHODS ===============================
// ===========================================================

// ------CONSTRUCTOR -------------------

cSerial::cSerial (unsigned addr) {

  iobase = addr;
  if(iobase == PRIME)
    pCom = COM1;
  else
    pCom = COM2;

  // ** initialize the serial port **
  initSerial(96,8,0,1);

}

/***************** initSerial Method Definition ****************************/

void cSerial::initSerial(int baud=96,int databits=8,int parity=0, int stopbits=1)
{

  switch(baud) {
    case '12': pConfig = BAUD_12;
                 break;
    case '24': pConfig = BAUD_24;
                 break;
    case '48': pConfig = BAUD_48;
                 break;
    case '96': pConfig = BAUD_96;
                 break;
        default: pConfig = BAUD_96;
                 break;
  }

  switch(databits) {
    case '7': pConfig = pConfig | DATA_7;
              break;
     default: pConfig = pConfig | DATA_8;
              break;
  }

  switch(parity) {
    case '1': pConfig = pConfig | PAR_ODD;
              break;
    case '2': pConfig = pConfig | PAR_EVEN;
              break;
     default: pConfig = pConfig | PAR_NONE;
              break;
  }

  switch(stopbits) {
    case '2': pConfig = pConfig | STOP_2;
              break;
     default: pConfig = pConfig | STOP_1;
              break;
  }

//  idComDev = OpenComm( "Com1", 1024, 128 ); // open the comm using windows conformin
g routines
  _bios_serialcom(_COM_INIT, pCom, pConfig);    // configure the serial i/o port here

  if(chkPortStat() && 0x0064)
    rtnCode = 0;        // no error
  else
    rtnCode = -2;       // error on the port!

}


/***************** chkPortStat Method Definition ****************************/

unsigned cSerial::chkPortStat() {
```

```
  unsigned portStat;  // status word from the I/O port

  portStat =  _bios_serialcom(_COM_STATUS, pCom, 0);

  return(portStat);
}

/***************** readThePort Method Definition ****************************/

int cSerial::readThePort() {     // called only if there are characters to read

  int i=0;               // for index loop
  int nc;

// read the characters into the io_buf string. The string will handled 80 characters.
// Readserial() returns 0 if there are no more characters to read

  if((nc = howManyChars()))
    if(chkForRTN()) {                        // check that the last char is a "\r"
      if(nc > 80) nc = 80;
      for(i=0;i<nc;i++)                      // if not "\r" we are not ready to read t
he port
       if(!(*(io_str+i) = readSerial()))     // if yes we will read all the characters
        break;
      clear_serial_Q();
      io_str[nc] = 0;                       // terminate the string at the end of the
input
      return TRUE;
    }
    else
      return FALSE;
  else
    return FALSE;

} // --- end of readThePort Method definition ---


/***************** writeToPort Method Definition **************************/

void cSerial::writeToPort(char *str) {

  char *s;

  for (s=str; *s; s++) {
    if(writeserial(*s) == 0)
      break;
  }
}



/***************** signOn Method Definition ****************************/

void cSerial::signOn() {

  gZephyr->outStr = "[Echelle Motor Code V1.0 Beta]\r";

  gZephyr->responder(2);

}


// -------- END OF cSERIAL METHODS --------------------------------------------
```

```
/* Timer.c -- Timer interrupt routines for the echelle spectrometer */

/* ------------------------------------- *
   *** ------ Timer functions ------ ***
/  ------------------------------------- */

/* These are NOT C++! */

#include <stdio.h>
#include <dos.h>
#include "timer.h"
#include "interrupt.h"


unsigned ticker;

static void (interrupt far *oldtimer)();
static void interrupt far newtimer();
static void (*timeout_function)();

/* --- intercept the timer interrupt vector --- */
void intercept_timer(void (*fn)()) {

  if(oldtimer == NULL) {
    oldtimer = _dos_getvect(TIMER);
    _dos_setvect(TIMER, newtimer);
  }
  timeout_function = fn;
}

/* --- restore timer intercept vector --- */

void restore_timer() {

  if(oldtimer != NULL)
    _dos_setvect(TIMER, oldtimer);

  timeout_function = NULL;
}

/* --- ISR to count timer ticks --- */

static void interrupt far newtimer() {

  (*oldtimer)();
  if(ticker > 0)
    if(--ticker == 0 && timeout_function != NULL)
      (*timeout_function)();
}
```

```
// CONTENTS : cError -- Subclass of CObject. Handles all errors in the
//                        Motor.cpp applications

// ***--------------------------------***------------------------------***
//    SPECS --
//            Operating System: DOS 6.2
//            Language        : C++ (Microsoft Specific)
//            Compiler        : Microsoft Visual C++ 1.5 (medium Lib)
//
//    HISTORY --
//            Creation Date   : July 11, 1994
//            Edit Dates      : June 20, 1994 - added more error case handlers
//                            : Apr 29, 1996  - made Windows compatable
//                            : Sep 16, 1997  - Now concats outStr only
//                                             - traps errCode 7 and 8 now
//                            : 24 Sep 97 - added errCode 9 for calibOut test;
//                                          errCode 9 => mirror not moved all the way
out
//
//            Author          : TjM (University of Chicago, Yerkes Obs)
//
// ***--------------------------------***------------------------------***

#if defined(_WINDOWS)
#include "stdafx.h"
#endif
#if defined(_DOS)
#include <afx.h>
#endif

#include <iostream.h>
#include <stdlib.h>       // for the itoa conversion routine
#include "optiparm.hpp"   // include file global vars
#include "dispatch.hpp"   // class definition
#include "cerror.hpp"     // include file with class definitions

extern cDispatcher* gZephyr;

extern errCode;

// =================================================================
// ==================== cError Methods ==========================
// =================================================================

//***************** Constructor Definition ****************************

cError::cError() { }


//***************** cErrHndl Definition ****************************


void cError::cErrHndl() {

  char tempbuf[20];

    switch(errCode) {

      case errNone: gZephyr->outStr += "No OptiStep Error]\r";
                    return;
                    break;

      case errOpt1:
      case errOpt2:
      case errOpt3:
      case errOpt4: gZephyr->outStr += "OptiStep Error: ";
                    itoa(errCode,tempbuf,10);
                    gZephyr->outStr+= tempbuf;
                    gZephyr->outStr+= "]\r";
                    gZephyr->responder(2);
                    errCode = 0;  // reset errCode
                    break;

      case errPort: gZephyr->outStr += "I/O Port Error: ";
                    itoa(errCode,tempbuf,10);
                    gZephyr->outStr+= tempbuf;
                    gZephyr->outStr+= "]\r";
                    gZephyr->responder(2);
                    errCode = 0;  // reset errCode
                    break;

     case errCount: gZephyr->outStr += "Counting Error: ";
                    itoa(errCode,tempbuf,10);
                    gZephyr->outStr+= tempbuf;
                    gZephyr->outStr+= "]\r";
                    gZephyr->responder(2);
                    errCode = 0;  // reset errCode
                    break;

      case A1_BOT:  gZephyr->outStr += "Lower Limit Switch Triggered]\r";
                    errCode = 0;  // reset errCode
                    break;
      case A1_TOP:  gZephyr->outStr += "Upper Limit Switch Triggered]\r";
                    errCode = 0;  // reset errCode
                    break;
      case A2_BOT:  gZephyr->outStr += "Lower Limit Switch Triggered]\r";
                    errCode = 0;  // reset errCode
                    break;
      case A2_TOP:  gZephyr->outStr += "Upper Limit Switch Triggered]\r";
                    errCode = 0;  // reset errCode
                    break;
      case A3_BOT:  gZephyr->outStr += "Lower Limit Switch Triggered]\r";
                    errCode = 0;  // reset errCode
                    break;
      case A3_TOP:  gZephyr->outStr += "Upper Limit Switch Triggered]\r";
                    errCode = 0;  // reset errCode
                    break;
      case 7:                 // do nothing here
      case 8:       gZephyr->outStr += "Limit Switch for Axis 4]\r";
                    break;
      case 9:       gZephyr->outStr += "Mirror not fully out of beam path]\r";
                    break;

          default:  gZephyr->outStr += "Other Error: ";
                    itoa(errCode,tempbuf,10);
                    gZephyr->outStr+= tempbuf;
                    gZephyr->outStr+= "]\r";
                    gZephyr->responder(2);
                    errCode = 0;  // reset errCode
                    break;

      } // end of switch(errCode) statement

    return;

} // ***_______end of cErrHndl method definition_______***
```

```
// CONTENTS : cSystem.cpp -- Method definitions for the object class cSystem.
//                           This class represents the system as a whole.
//
// ***--------------------------------***------------------------------***
//    SPECS --
//            Operating System: DOS 6.2
//            Language        : C++ (Microsoft Specific)
//            Compiler        : Microsoft Visual C++ 1.5 (medium Lib)
//
//    HISTORY --
//            Creation Date   : Aug 18, 1994
//            Edit Dates      : Aug 31, 1994 - changed object pointer names.
//                            :              - added gObjList pointer for
//                                             object list references
//                            : Sept 1, 1994 - CObList did not work for
//                                             CStepperMotor pointers
//                            :              - so I will now use a simple for
//                                             loop and generic ptr
//                            :              - No More CObList!
//                            : Sept 9, 1994 - changed objptr from CStepperMotor
//                                             type to cFilterWhl
//                            : Dec 4, 1994  - added printf to write status to
//                                             the screen as well as the port
//                            : Feb 9, 1995  - VacOn/VacOff removed to main()
//                            : Apr 28, 1995 - removed references to Shutter
//                            : Sep 09, 1997 - added checks for vac system on/off
//                            : Sep 16, 1997 - changed keyword vacuum0 and vacuum1
//                                             to vacuumL and vacuumH respectively
//                            : 24 Sep 97 - Added lamp status to ST/IS output
//                            : 10 Oct 97 - Changed systemStatus() to remove printf
//                                          statements - responder() now prints to the
//                                          CRT
//                            : 20 Jan 98 - Removed queryStatus() call from systemStat
us;
//                                          it was resetin the limitStatus to 0 every
time;
//                                          probably problem with OptiStep, not code
//                            : 12 May 98 - Added ion pump to status routines
//
//            Author          : TjM (University of Chicago, Yerkes Obs)
//
// ***--------------------------------***------------------------------***

//***===== constructor ==============================================***

#if defined(_WINDOWS)
#include "stdafx.h"
#endif
#if defined(_DOS)
#include <afx.h>
#endif


#include <stdio.h>            // for printf function
#include "cSystem.hpp"
#include "opti1.hpp"          // for definitions of csteppermotor
#include "cFilter.hpp"        // cFilter object definition
#include "cMtrObL.hpp"        // Object list class
#include "dispatch.hpp"     // cdispatcher definitions
#include "Vacuum.hpp"         // vacuum sensor class
#include "objsave.hpp"
#include "cLamp.hpp"
#include "ion.hpp"
#include "pointers.hpp"       // extern object pointers
extern cVacuum    *gVac;      // ===> include this later in pointers.hpp
extern cLamp   *gLamp;    // pointer to the lamp source object
extern cIon     *gIon;    // pointer to Ion pump control

cSystem::cSystem() { }

void cSystem::systemStatus() {

  int motor, value;
  cBasicMotor* objptr;    // the generic pointer used for all the motor objects

//  The following function, although it looks fine, resets the status of the motor lim
it
//  switches to off.  I think this is because the OptiStep boards don't report limit s
w
//  status unless there is a move command - anyway, status is always 0, even when a sw
 is
//  triggered.  PEG 20 Jan 98
//  queryStatus();

  CString tmp_buf = "";     // used for itoa function
  CString outStr = "";      // create string object to send to dispatcher

  // This method should retrieve the following:
  //
  //    * motor ID
  //    * which OptiStep Board
  //    * motor position
  //    * limit switch status
  //    * motor settings (speed, power, ramp)
  //

  // =============================******=====================================
  // As of Sept 9, 1994, there are 9 active stepper motors within the echelle.
  // The pointer array elemets are as follows:
  //
  //    sList[0]=gCalMirror;      calibration mirror motor
  //    sList[1]=gEFilter;        echelle instrument filter motor
  //    sList[2]=gCFilter;        guider camera filter motor
  //    sList[3]=gGrating;        grating tilt motor
  //    sList[4]=gSlit;           slit focus motor
  //    sList[5]=gTip;            CCD tip additude motor
  //    sList[6]-gTilt;           CCD tilt additude motor
  //    sList[7]=gCCD;            CCD focus motors (two motors, one object)
  //
  // =============================******=====================================


  // get the driver board status
  outStr = "{\rDriver #1: ";

  if(!gOptiOne->boardOK)
    outStr+="OK\r";
  else
    outStr+="NOT PRESENT!\r";

  outStr+= "Driver #2: ";

  if(!gOptiTwo->boardOK)
    outStr+="OK\r";
  else
    outStr+="NOT PRESENT!\r";

  outStr+= "Driver #3: ";
```

```
  if(!gOptiThree->boardOK)
    outStr+="OK\r";
  else
    outStr+="NOT PRESENT!\r";
  outStr+= "Driver #4: ";

  if(!gOptiFour->boardOK)
    outStr+="OK\r\r";
  else
    outStr+="NOT PRESENT!\r\r";

  gZephyr->outStr = outStr;            // assign gZephyr->outStr
  gZephyr->responder(2);               // send message to responder

  // get the motor parameters ---->

  outStr = "NAME       Limits(u/l)   POS (steps)   ZERO    \r";
  //        0123456789012345678901234567890123456789012345678
  //        0         1         2         3         4
  //        CalbMir      0/0          nnnnnnnnn

  gZephyr->outStr = outStr;            // assign gZephyr->outStr
  gZephyr->responder(2);               // send message to responder
//  printf(outStr);                      // also send string to screen
//  printf("\n");

  // Now do the cal mirror
  outStr = gCalMirror->name;
  outStr+= "      ";                    // add a couple of spaces
  if(gCalMirror->position == 1)
    outStr+="IN\r";
  else
    if(gCalMirror->position == 0)
      outStr+="OUT\r";
    else
      outStr+="ERROR\r";

  // now send it off to the gZephyr->responder
  gZephyr->outStr = outStr;            // assign gZephyr->outStr
  gZephyr->responder(2);                // send message to responder
//  printf(outStr);                      // also send string to screen
//  printf("\n");


  for(motor=1;motor<=2;motor++) {
     objptr = gList->getPtr(motor);    // set pointer to the proper object
     outStr = objptr->name;            // write the name of motor to outStr
     outStr+= "         ";             // add a couple of spaces

     tmp_buf = "";                     // clear the bufer fo the next item
     outStr+= "          ";            // add a couple of spaces
     ltoa(objptr->position,tmp_buf.GetBuffer(0),10);   // get the stepCount
     outStr+= tmp_buf.GetBuffer(0);    // add it to the outStr object
     outStr+= "\r";

     // now send it off to the gZephyr->responder
     gZephyr->outStr = outStr;         // make gZephyr->outStr same as this one
     gZephyr->responder(2);            // send message to responder
     printf(outStr);                   // also send string to screen
//     printf("\n");

  } // end of the for statement

  for(motor=3;motor<=7;motor++) {
     objptr = gList->getPtr(motor);    // set pointer to the proper object
     outStr = objptr->name;            // write the name of motor to outStr
     outStr+= "      ";                // add a couple of spaces

     itoa(objptr->limitStat[0],tmp_buf.GetBuffer(0),10);
     outStr+= tmp_buf.GetBuffer(0);    // add it to the outStr object

     tmp_buf = "";                     // clear the bufer fo the next item
     outStr+= "/";                     // add a couple of spaces
     itoa(objptr->limitStat[1],tmp_buf.GetBuffer(0),10);
     outStr+= tmp_buf.GetBuffer(0);    // add it to the outStr object

     tmp_buf = "";                     // clear the bufer fo the next item
     outStr+= "          ";            // add a couple of spaces
     ltoa(objptr->position,tmp_buf.GetBuffer(0),10);   // get the stepCount
     outStr+= tmp_buf.GetBuffer(0);    // add it to the outStr object
     outStr+= "          ";            // add a couple of spaces

     if(objptr->axisID == 1)
        value=0x3E;
     else
        value=0x3D;

     if(objptr->statAux==value | objptr->statAux==0x3C)
        outStr+= "Y";
     else
        outStr+= "N";
     outStr+= "\r";

     // now send it off to the gZephyr->responder
     gZephyr->outStr = outStr;         // make gZephyr->outStr same as this one
     gZephyr->responder(2);            // send message to responder
//     printf(outStr);                   // also send string to screen
//     printf("\n");

  } // end of the for statement


  // Now do the vacuum status

      if(gVac->is_vac_on[0])
        gVac->readSensors();
      else
        gVac->pressure0 = -1;

      if(!gVac->is_vac_on[1])
        gVac->pressure1 = -1;

     outStr= "\rVacuum Pressure(Torr)\r";
     outStr+="Low Vac Sensor: ";
     tmp_buf = "";
     gcvt(gVac->pressure0,4,tmp_buf.GetBuffer(0));
     outStr+= tmp_buf.GetBuffer(0);
     outStr+= "\rHigh Vac Sensor: ";
     gcvt(gVac->pressure1,4,tmp_buf.GetBuffer(0));
     outStr+= tmp_buf.GetBuffer(0);

     // now send it off to the gZephyr->responder
     gZephyr->outStr = outStr;            // assign gZephyr->outStr
     gZephyr->responder(2);               // send message to responder

     outStr= "\n\rLamp Status\r";
     outStr+= "White Lamp: ";
     outStr+= gLamp->lampW;
     outStr+= "\rTh/Ar Lamp: ";
     outStr+= gLamp->lampT;
```

```
      outStr+= "\rFe/Ne Lamp: ";
      outStr+= gLamp->lampF;
      outStr+= "\r}";

      if(gIon->IonStatus())
       outStr += "Ion HV: on\r";
      else outStr += "Ion HV: off\r";

      // now send it off to the gZephyr->responder
      gZephyr->outStr = outStr;            // assign gZephyr->outStr
      gZephyr->responder(2);               // send message to responder

}

void cSystem::interFaceStat() {

  int motor;
  cBasicMotor* objptr;        // generic pointer used for all the motor objects

  CString tmp_buf = "";     // used for itoa function
  CString outStr = "";      // create string object to send to dispatcher

  for(motor=0;motor<=7;motor++) {
    objptr = gList->getPtr(motor);    // set pointer to the proper object
    tmp_buf = "";                     // clear the bufer fo the next item
    outStr+= objptr->name;            // write the name of the motor to buffer
    outStr+="=";                      // '=' is the string delimeter
    ltoa(objptr->position,tmp_buf.GetBuffer(8),10);   // get the position
    outStr+= tmp_buf.GetBuffer(0);    // add it to the outStr object
    outStr+=" ";
  }

  outStr+="vacuumL=";
  tmp_buf = "";

  if(gVac->is_vac_on[0])
    gVac->readSensors();
  else
    gVac->pressure0 = -1;

  if(!gVac->is_vac_on[1])
    gVac->pressure1 = -1;

  gcvt(gVac->pressure0,4,tmp_buf.GetBuffer(0));
  outStr+= tmp_buf.GetBuffer(0);

// add a space to the end of the output string
  outStr+=" ";
  outStr+="vacuumH=";
  tmp_buf = "";

  gcvt(gVac->pressure1,4,tmp_buf.GetBuffer(0));
  outStr+= tmp_buf.GetBuffer(0);

  outStr+= " lampW=";
  if(gLamp->lampW == "off") outStr+= "0";
  else outStr+= "1";
  outStr+= " lampT=";
  if(gLamp->lampT == "off") outStr+= "0";
  else outStr+= "1";
  outStr+= " lampF=";
  if(gLamp->lampF == "off") outStr+= "0";
  else outStr+= "1";

  outStr+= " ionHV=";
  if(gIon->IonStatus())
    outStr+= "1";
  else outStr+= "0";
// add a space to the end of the output string
  outStr+=" ";

// now send it off to the gZephyr->responder
  gZephyr->outStr = outStr;         // assign gZephyr->outStr
  gZephyr->responder(2);            // send message to responder


}


void cSystem::systemReset() { }



void cSystem::queryStatus() {

  int status;

  gOptiTwo->cCardInit();

  status = gOptiTwo->cLimitStatus();
  status = status&0x00FF;                  // strip off high order bits
//printf("<status byte: %d>\n", status);
  gGrating->limitStat[0] = status&0x01;
  gGrating->limitStat[1] = (status&0x02)>>1;
  gSlit->limitStat[0] = (status&0x04)>>2;
  gSlit->limitStat[1] = (status&0x08)>>3;

  gOptiThree->cCardInit();

  status = gOptiThree->cLimitStatus();
  status = status&0x00FF;                  // strip off high order bits
  gTip->limitStat[0] = status&0x01;
  gTip->limitStat[1] = (status&0x02)>>1;
  gTilt->limitStat[0] = (status&0x04)>>2;
  gTilt->limitStat[1] = (status&0x08)>>3;

  gOptiFour->cCardInit();

  status = gOptiFour->cLimitStatus();
  status = status&0x00FF;                  // strip off high order bits
  gCCD->limitStat[0] = status&0x01;
  gCCD->limitStat[1] = (status&0x02)>>1;

  gZephyr->lastOS = 4;

}
```

```
// CONTENTS : cMtrObL.cpp -- A list with the motor object pointers. This will
//                           be used in conjunction with cSystem class object.

// ***------------------------------***-----------------------------***
//    SPECS --
//            Operating System: DOS 6.2
//            Language        : C++ (Microsoft Specific)
//            Compiler        : Microsoft Visual C++ 1.5 (medium Lib)
//
//    HISTORY --
//            Creation Date   : Sept 1, 1994
//            Edit Dates      : Apr 28, 1995 - eliminated gShutter from code
//                            : Feb 28, 1997 - compatable with new cBasicMotor class
//
//            Author          : TjM (University of Chicago, Yerkes Obs)
//
// ***------------------------------***-----------------------------***

// ***-------------------------------------------------------------***

#if defined(_WINDOWS)
#include "stdafx.h"
#endif
#if defined(_DOS)
#include <afx.h>
#endif

#include "opti1.hpp"
#include "cFilter.hpp"
#include "cMtrObL.hpp"
#include "Dispatch.hpp"
#include "objsave.hpp"
#include "pointers.hpp"

// -------------------------------------------
// ------->  Constructor definition ---------->
// -------------------------------------------

cMtrObL::cMtrObL() {

  int i;

  for(i=0;i<=12;i++)
    sList[i] = NULL;

  addPtrs();          // fill the pointer list

 }

// -------------------------------------------
// ------->  cMtrObL::getPtr definition ------>
// -------------------------------------------

cBasicMotor* cMtrObL::getPtr(int idx) {

  return(sList[idx]);

}

// -------------------------------------------
// ------->  cMtrObL::addPtrs definition ----->
// -------------------------------------------

void cMtrObL::addPtrs() {

// ==== As of Sept 9, 1994, there are 9 motor pointers in use =====

  sList[0]=gCalMirror;      // load the pointers
  sList[1]=gEFilter;
  sList[2]=gCFilter;
  sList[3]=gGrating;
  sList[4]=gSlit;
  sList[5]=gTip;
  sList[6]=gTilt;
  sList[7]=gCCD;            // two motors, one object
//  sList[8]=gShutter;

}
```

```
// CONTENTS : OptiStep.c -- Object oriented files for the echelle motor drives
//                          on the PC!

// ***-------------------------------***-----------------------------***
//    SPECS --
//            Operating System: DOS 6.2
//            Language        : C++ (Microsoft Specific)
//            Compiler        : Microsoft Visual C++ 1.5 (medium Lib)
//
//    HISTORY --
//            Creation Date   : June 13, 1994
//            Edit Dates      : June 13, 1994 - added Const double oscFrec and
//                                              errCode
//                            : June 20, 1994 - moved class definitions
//                                              (COptiStep & CStepperMotor)
//                                              to "proto.hpp" header file.
//                            : Aug 16, 1994  - changed def of cRampProfile
//                                              to match real world
//                            : Aug 26, 1994  - changed some parameters and
//                                              eliminated some others
//                                              cReadStepCount now returns
//                                              void but assigns obbj data
//                                              stepCount
//                            : Aug 30        - modified cReadStepCount to be
//                                              more generic (simple). Daughter
//                                              classes will handle more
//                                              specific cases.
//                            : Sept 5, 1994  - rmoved redundant cReadStepCount
//                            : Sept 6, 1994  - moved cSetMovMode from
//                                              CStepperMotor to COptiStep
//                                              class
//                            : Sept 21       - CStepperMotor::cMoveMotor now
//                                              is relative move only!
//                            :               - power setting for idle motor is
//                                              20%
//                            : Sept 26       - cMoveMotor now takes long int
//                                              as argument
//                            : Dec 2, 1994   - removed setToRamp() from
//                                              cMoveMotor()
//                            : Mar 23, 1995  - mods to CStepper::MoveMotors...
//                                              added case 4: to accomidate CCD
//                                              focus (moves 2 motors simul.
//                            : Jun 28, 1996  - Now power is left at 20% on the
//                                            - Optidrivers boards. the motors
//                                            - exper. pulses as the power shut
//                                            - down.
//                            : 7 Nov 97      - Changed cCardInit to explicitly
//                                              set movement mode to relative
//                            : 30 Apr 98 - Changed cSetSpeed() to use the parameter
//                                          "speed" and to NOT reset stpsPerSec
//
//            Author          : TjM (University of Chicago, Yerkes Obs)
//
// ***-------------------------------***-----------------------------***

#if defined(_WINDOWS)
#include "stdafx.h"
#endif
#if defined(_DOS)
#include <afx.h>
#endif

#include <stdio.h>
#include <dos.h>
#include <iostream.h>
#include "optiparm.hpp"   // include file COptiStep class
#include "opti1.hpp"      // class and function prototype definitions
extern "C" {
#include "is_cm.h"        // force "C" linkage of the instep routines
}

extern errCode;
extern rtnCode;

// ---------------------------------------------------------------------
// ----------------- COptiStep METHODS ---------------------------------
// ---------------------------------------------------------------------

/***************** Constructor Definition ****************************/

COptiStep::COptiStep() {

  whichPort = 0;
  rtnCode = 1;
  errCode = errNone;
  baseAdr = 0x250;
  oscFreq = 2000000.0;
  cnt1 = 0;               // zero the counter storage vars
  cnt2 = 0;
  cnt3 = 0;
  cnt4 = 0;

  cSetCardType(OptiSP);
  cCardInit();
  if(errCode)
    boardOK = 1;
  else
    boardOK = 0;
}

COptiStep::COptiStep(int baseAddress) {

  baseAdr = baseAddress;  // set the object data with the passes argument
  whichPort = 0;
  rtnCode = 1;
  errCode = errNone;
  oscFreq = 2000000.0;
  cnt1 = 0;               // zero the counter storage vars
  cnt2 = 0;
  cnt3 = 0;
  cnt4 = 0;

  cSetCardType(OptiSP);
  cCardInit();
  if(errCode)
    boardOK = 1;
  else
    boardOK = 0;

} /* ---------- End of Constructor definition -------------------------- */



/***************** CardInit Method Definition ****************************/

void COptiStep::cCardInit () {
  int i;

  ischeckcard( &baseAdr, &rtnCode );
  if (rtnCode == 0)
```

```
    switch(baseAdr) {
      case 0x250: errCode = errOpt1;
                  break;
      case 0x260: errCode = errOpt2;
                  break;
      case 0x270: errCode = errOpt3;
                  break;
      case 0x280: errCode = errOpt4;
                  break;
    }
  else {
    errCode = errNone;
    cResetCounters();     // reset the board counter values to the cnt1, cnt2...
  }                       // object data

  cSetMovMode(0);         // Insure that the card is in relative mode
  i=0; isfeedhold(&i);    // Insure that a shield ground will kill move
  return;
}   /* ------------ End of CardInit Method definition ---------------------- */



/***************** PortControl Method Definition ****************************/

void COptiStep::cPortControl (int whichPort, int state = 0) {

  if ((whichPort < 1) && (whichPort > 6)) {
    iscontrol(&whichPort, &state);
    errCode = errNone; }
  else
    errCode = errPort;

  return;

} /* ------------ End of PortControl Method definition ------------------- */


/***************** LimitStatus Method Definition ****************************/

// returns the optistep status word

int COptiStep::cLimitStatus () {

  int shldStat, status;

  isstatus(&ospStat,&shldStat);

  status = ospStat&0x00FF;

//  if(!status)
//    errCode = errNone;
//  else
//    errCode = errLSw1;

  return(ospStat);

} /* ----------- End of LimitStatus definition ---------------------------- */


/***************** SetCardType Method Definition ****************************/

void COptiStep::cSetCardType (int cardType) {

  issetcardtype(&cardType); // the default is the COptiStep Plus card
  return;

} /* ----------- End of SetCardType Method definition -------------------- */


void COptiStep::cReadStepCount() {

  isreadcount(&cnt1,&cnt2,&cnt3,&cnt4);

  return;

} /* ------------ End of cReadStepCount Method definition ---------- */

/***************** cResetCounters Method Definition *************************/

void COptiStep::cResetCounters() {

   isloadcount(&cnt1, &cnt2, &cnt3, &cnt4);  // this method resets the optistep
   return;                                   // board counters after a reset

} /* ------------ End of cResetCounters Method definition ---------- */

/***************** cSetMovMode Method Definition *******************/

void COptiStep::cSetMovMode (int rel_abs) {

  issetmode(&rel_abs);
  return;

} /* ------------ End of cSetMovMode Method definition -------------- */

/***************** cSetStepCount Definition ****************************/

//   Method sets the step count of any one axis to specified
//   value 'newCount'. If error occurs sets errCode to errCount.


void COptiStep::cSetStepCount (int StepMtr, long newCount) {

  isreadcount(&cnt1, &cnt2, &cnt3, &cnt4);  // the counts are read

  switch (StepMtr) {
    case 1: cnt1 = newCount;
            isloadcount(&cnt1, &cnt2, &cnt3, &cnt4);
            errCode = errNone;
            break;
    case 2: cnt2 = newCount;
            isloadcount(&cnt1, &newCount, &cnt3, &cnt4);
            errCode = errNone;
            break;
    case 3: cnt3 = newCount;
            isloadcount(&cnt1, &cnt2, &newCount, &cnt4);
            errCode = errNone;
            break;
    case 4: cnt4 = newCount;
            isloadcount(&cnt1, &cnt2, &cnt3, &newCount);
            errCode = errNone;
            break;
    default: errCode = errCount;
             break;
    }
  isreadcount(&cnt1, &cnt2, &cnt3, &cnt4);  // the counts are read

} /* ------------ End of cSetStepCount Method definition ------------ */
```

```
// ****************** END of COptiStep METHODS ******************************



// --------------------------------------------------------------------
// ----------------- CStepperMotor METHODS -----------------------------
// --------------------------------------------------------------------

/***************** Constructor Definition ***************************/

CStepperMotor::CStepperMotor() { } // default constructor

CStepperMotor::CStepperMotor(int axis, int optib) {

  axisID = axis;
  whichOpti = optib;

  switch(whichOpti)  {
    case 1: baseAdr = 0x250;
            break;
    case 2: baseAdr = 0x260;
            break;
    case 3: baseAdr = 0x270;
            break;
    case 4: baseAdr = 0x280;
            break;
  } // end of switch statement

  start = 100;
  end = 600;
  accel = 300;
  stpsPerSec = 1000;

  powrSetting = pwr0;                               // default power = 0%
  CStepperMotor::cRampProfile(start,end,accel);     // initial ramping params
  CStepperMotor::cReadStepCount();                  // read count of the motors

} /* ---------- End of Constructor definition ------------------- */

/***************** MoveMotor Method Definition *********************/

void CStepperMotor::cMoveMotor (long numofSteps) {

  //
  // This method has been modified to make relative moves only!
  //

  long stp1=0,   // temp storage vars for the steps to move
       stp2=0,
       stp3=0,
       stp4=0;

  switch(axisID) {
      case 1: stp1 = numofSteps;    // moves one axis only at a time
              break;
      case 2: stp2 = numofSteps;
              break;
      case 3: stp3 = numofSteps;
              break;
      case 4: stp1 = numofSteps;
              stp2 = numofSteps;
              break;
      default:  break;
  }

  CStepperMotor::cPowerCtrl(pwr100);          // enable the power to the motors
  ismover(&stp1,&stp2,&stp3,0,&errCode);      // do the move
  CStepperMotor::cPowerCtrl(pwr0);            // disable the power
                                              // changed from pwr20 to pwr0 on 22 Apr
98


} /* ---------- End of cMoveMotor Method definition ---------------- */



/***************** cRampProfile Method Definition *******************/

void CStepperMotor::cRampProfile (int bottom, int top, int acl) {

  isprofile(bottom,top,(long)acl); // may have to set this each move?

} /* ------------ End of cRampProfile Method definition ------------ */



/***************** cSetSpeed Method Definition *******************/

void CStepperMotor::cSetSpeed (int speed) {

  clkDelay = (int)(oscFreq/(double)speed);
  issetspeed(&clkDelay);

} /* ------------ End of cSetSpeed Method definition -------------- */


/***************** cSetToRamp Method Definition *******************/

void CStepperMotor::cSetToRamp () {   // Enables the ramping for the next
  isrampset();                        // move of the motor

} /* ------------ End of cSetRamp Method definition -------------- */


/***************** cPowerCtrl Method Definition *******************/

void CStepperMotor::cPowerCtrl (int powrSetting) {

  ispowerinit(&powrSetting);

} /* ------------ End of cPowerCtrl Method definition -------------- */



// ****************** END of CStepperMotor METHODS **************************
```

```
// CONTENTS : Disptch2.cpp -- This file contains the function prototypes
//                            for the class cDispatcher.

// ***--------------------------------***------------------------------***
//    SPECS --
//           Operating System: DOS 6.2
//           Language        : C++ (Microsoft Specific)
//           Compiler        : Microsoft Visual C++ 1.5 (medium Lib)
//
//    HISTORY --
//           Creation Date   : July 20, 1994
//           Edit Dates      : July 25, 1994 - major mods including name
//                           :                 changes
//                           : Aug 11, 1994 - major mods including name
//                           :                 changes
//                           : Aug 17, 1994 - pass by ref the pointer to
//                           :                 cSerial port object
//                           : Aug 19, 1994 - now use pointers to access
//                           :                 object methods
//                           : Aug 23, 1994 - added MVA and MVR to command
//                           :                 list, removed MV command
//                           : Aug 30       - added assignment of movHow for
//                           :                 each motor
//                           : Aug 31       - changed pointer names(gEFilter,
//                           :                 gCFilter, gSlit)
//                           : Sept 9, 1994 - extensive mods adding many new
//                           :                 motor objects
//                           : Sept 15      - added cVacuum class
//                           : Sept 21      - motor moves are relative only!
//                           :              - added ramp command.
//                           : Sept 28      - restructured router()
//                           : Nov 2, 1994  - added ZM (zero motor count)
//                           : Dec 10, 1994 - added FH command
//                           : Jan 26, 1995 - added glamp pointer refs
//                           : Apr 04, 1995 - removed shutter commands
//                           : Apr 29, 1996 - made Windows compatable
//                           : Jul 31, 1996 - removed SO and SC commands
//                           : Aug 29, 1996 - replaced -1 with 0 for ommition
//                                            in MV command
//                           : Sep 03, 1997 - added findHome parsing for filter
//                                            wheels
//                                          - changed command ZM to ZC
//                                          - change indices of ZC and FH from
//                                            0-7 to 1-8
//                           : Sep 08, 1997 - trapped more ivalid responses in
//                                            parser()
//                                          - changed atoi to atol in
//                                            prs_mv_cmd()
//                           : Sep 09, 1997 - re-arranged parser() and fixed
//                                            many small bugs
//                           : Sep 10, 1997 - added checks in router() to
//                                            confirm motor move was completed
//                                          - added new commands VA and VO
//                                            for Vacuum Activate and Vacuum Off
//                           : Sep 16, 1997 - added LX command and lamp parameter
//                                            'A' for ALL lamps
//                                          - added gError->cErrHndl() calls in
//                                            router
//                           : 24 Sep 97 - Modified CO/CI to return error msgs in
//                                         same style as for MV command
//                           : 10 Oct 97 - Changed responder() to print near-identica
l
//                                         messages to the screen when anything is se
nt
//                                         out over the serial port
//                           : 13 Oct 97 - Fixed bug in router that gave spurious err
or msgs
//                                         by introducing target_pos[] - array of whe
re each
//                                         motor should be moving to (NOT # of steps)
.
//                           : 1 Apr 98  - Added WV, WO and timestamp for vacuum logg
ing
//                           : 6 Apr 98  - Added DeGas() private fn and called by DG
console cmd
//                           : 27 Apr 98 - Fixed ZC problem with Tip & Tilt - wrong a
xes in code
//                                       - Added console commands "PA" & "PO" to powe
r/depower boards
//                           : 12 May 98 - Added cIon calls - IA, IO
//                           : 19 May 98 - Added KL command for system shutdown
//                                         Shutdown lasts until next command received
//
//           Author          : TjM (University of Chicago, Yerkes Obs)
//
// ***--------------------------------***------------------------------***

#if defined(_WINDOWS)
#include "stdafx.h"
#endif
#if defined(_DOS)
#include <afx.h>
#endif

#include <stdlib.h>       // used for atoi function
#include <time.h>         // used for vacuum logging timestamp
#include <dos.h>
#include "optiparm.hpp"
#include "cSerial.hpp"
#include "cSystem.hpp"
#include "Opti1.hpp"
#include "cFilter.hpp"
#include "cError.hpp"
#include "cMtrObL.hpp"    // motor object list
#include "dispatch.hpp"   // class definition
#include "Vacuum.hpp"
#include "cLamp.hpp"
#include "objsave.hpp"
#include "cPIO.hpp"
#include "ion.hpp"
#include "pointers.hpp"   // list of extern pointer definitions

extern cSerial *gIOPort;
extern cSystem *gSystem;
extern cVacuum *gVac;     // pointer to the vacuum sensors
extern cLamp   *gLamp;    // pointer to the lamp source object
extern cError  *gError;   // pointer to error handling object
extern cPIO    *gPIO;     // for PA, PO commands
extern cIon    *gIon;     // Ion pump object

extern int errCode;

// ===================================================
// ***===== Prototypes for cDispatcher Methods ======***
// ===================================================

// ************* Constructor ***********************

cDispatcher::cDispatcher() {
```

```
  int i;

  stat=0;
  cmd = 0;
  obj = 0;

  for(i=0;i<=7;i++) {
    nsteps[i][0] = 0;       // 0 in second element means don't move motor
    nsteps[i][1] = 0;
  }

  lastOS = 0;     // the last optistep board used

  inStr = "";
  outStr = "";

  ShutDown = FALSE;

}

// ===================================================
// ************ parser Method Definition *************
// ===================================================

//* ---> The command str should have the following format:
//*
//*        cmd_specifier parm1 parm2 parm3 ... NULL
//*
//* ---> A MOVE command would look like:
//*
//*        MV m1 m2 m3 m4 m5 m6 m7 m8 \r
//*
//* ---> A 'N' or 'n' in any parameter will skip the motor (DON'T MOVE)
//*
//* ---> The following commands are valid:
//*
//*       KL = Total power shutdown
//*       MV = move relative
//*       RS = reset system
//*       ST = system status
//*       AC = ramp parmeters
//*       CI = Calibration Mirror In
//*       CO = Cal Mirror Out
//*       LA = Lamp Activate
//*       LO = Lamps Off
//*       LX = Exclusive Lamp Activate
//*       IS = Interface Status
//*       IA = Ion pump activate
//*       IO = Ion pump off
//*       ZC = Zero Motor Count
//*       FH = Find Home
//*       VA = Vacuum Sensor Activate
//*       VO = Vacuum Sensor Off
//*       PA = Power Activate
//*       PO = Power Off
//*       WV = Write Vacuum Readings (CONSOLE)
//*       WO = Write Off (CONSOLE)
//*       DG = DeGas camera tank (CONSOLE)
//*
//* ---> All moves are absolute moves

void cDispatcher::parser(const char* str) {

  time_t stamp;
  struct tm *lstamp;

  int idx;                // string index
  int i = 0;

  cmd = 0;                // make sure the object data
  obj = 0;                // has been reset for the

  inStr = str;            // assign the string to the inStr object for parsing

  if(inStr.IsEmpty()) {   // first test for characters in the string
    stat=1;
    responder(0);         // if there are none tell user.
    return;               // This should never happen...hah!
  }

  inStr.MakeUpper();      // convert all string elements to uppercase.
                          // this way we only test for uppercase characters.

  for(idx=0;idx<=80;idx++)      // find the first non-space character or <rtn>
    if(inStr.GetAt(idx)!=0x20)
      break;
    else
      if(idx>=80) {
        stat=1;             // flag for an error!
        responder(0);       // send error condition to user
    }

  printf("CMD: %s\n", inStr.GetBuffer(0));

  if(ShutDown) {
    gPIO->pwr24V(1,1);            // Bring back power
    gPIO->pwr24V(2,1);
    gIon->IonOn();                // Make sure ion pump is on
    ShutDown = FALSE;
    }

  switch(inStr[idx]) {      // determine the cmd specifier string

    case 'K': if(inStr[idx+1] == 'L') {
                cmd = KIL;
                responder(0);
                router();
                }
              else {
                stat=1;
                responder(1);
                }
              responder(1);
              return;
              break;

    case 'A': if(inStr.GetAt(idx+1)=='C' && inStr.GetAt(idx+2)== 0x20) {
                cmd = AC;   // set acceleration and speed of ramp
                responder(0);
                router();
              }
              else  {
                stat = 1;
                responder(1);
              }
              responder(1);
              return;
              break;

    case 'C': if(inStr[idx+1] == 'I' &&  (inStr.GetAt(idx+2)== 0x20 || inStr.GetAt(idx
```

```
+2)== 0x0D)) {
                cmd = CI;               // Cal mirror in
                responder(0);
                router();
              }
              else
                if(inStr[idx+1] == 'O' &&  (inStr.GetAt(idx+2)== 0x20 || inStr.GetAt(i
dx+2)== 0x0D)) {
                  cmd = CO;                  // cal mirror out
                  responder(0);
                  router();
                }
                else  {
                  stat = 1;
                  responder(1);
                }
              responder(1);
              return;
              break;

    case 'F': if(inStr.GetAt(idx+1)=='H'&& inStr.GetAt(idx+2)== 0x20) {
                cmd = FH;          // Find Home...Move motor until
                responder(0);
                router();
                if(!gSave->write())    // Save the last location
                  errCode = 1;
              }
              else  {
                stat = 1;
                responder(1);
              }
              responder(1);
              break;

    case 'I':  if(inStr.GetAt(idx+1)=='S'&& (inStr.GetAt(idx+2)== 0x20 || inStr.GetAt(
idx+2)== 0x0D)) {
                cmd = IS;               // interface status command
                responder(0);
                router();
              }
              else if(inStr[idx+1] == 'A') {
                cmd = IA;
                responder(0);
                router();
                }
              else if(inStr[idx+1] == 'O') {
                cmd = IO;
                responder(0);
                router();
                }
              else  {
                stat = 1;
                responder(1);
              }
              responder(1);
              return;
              break;

    case 'L': if(inStr[idx+1] == 'A' && inStr.GetAt(idx+2)== 0x20) {
                cmd = LA;               // Lamp activate
                responder(0);
                router();
               }
              else
                if(inStr[idx+1] == 'O'&& inStr.GetAt(idx+2)== 0x20) {
                  cmd = LO;
                  responder(0);
                  router();
                 }
                else
                  if(inStr[idx+1] == 'X'&& inStr.GetAt(idx+2)== 0x20) {
                    cmd = LX;
                    responder(0);
                    router();
                   }
                  else {
                    stat = 1;
                    responder(1);
                  }
              responder(1);
              return;
              break;

    case 'M': if(inStr[idx+1] == 'V' && inStr.GetAt(idx+2)== 0x20) {
                cmd = MVR;
                i = prs_mv_cmd();    // the MV command
              }
              else
                stat = 1;

              if(i==1) {
                responder(0);      // command acknowledged
                router();
                if(!gSave->write())
                  errCode = 1;
              }                    // route the command to the appropriate obj
              else {               // else the command was not valid
               stat=1;             // set the error flag
               responder(1);       // send the invalid message back to user
              }
              responder(1);
              return;
              break;

    case 'S': if(inStr[idx+1] == 'T' && (inStr.GetAt(idx+2)== 0x20 || inStr.GetAt(idx+
2)== 0x0D)) {
                 cmd = ST;               // status command
                 responder(0);           // command ack
                 router();               // the ST command
              }
              else  {
                stat=1;                // set the error flag
                cmd = 0;               // status command
                responder(1);
              }
              responder(1);
              return;
              break;

    case 'R': if(inStr[idx+1] == 'S' && (inStr.GetAt(idx+2)== 0x20 || inStr.GetAt(idx+
2)== 0x0D)) {
                cmd = RS;                // the RS command
                responder(0);
                router();              // the ST command
              }
              else {
                stat = 1;
                responder(1);            // send "NAK"
              }
              responder(1);
```

```
              return;
              break;

    case 'V': if(inStr[idx+1] == 'A' && inStr.GetAt(idx+2)== 0x20) {
                cmd = VA;              // Lamp activate
                responder(0);
                router();
               }
              else
                if(inStr[idx+1] == 'O'&& inStr.GetAt(idx+2)== 0x20) {
                  cmd = VO;
                  responder(0);
                  router();
                 }
                else {
                  stat = 1;
                  responder(1);
                }
              responder(1);
              return;
              break;


    case 'Z': if(inStr[idx+1] == 'C' && inStr.GetAt(idx+2)== 0x20){
                cmd = ZC;               // Zero motor Counter command
                responder(0);
                router();
              }
              else  {
                stat = 1;
                responder(1);
              }
              responder(1);
              return;
              break;
// PA - activate power on an Optistep board
// PO - deactivate power on an OptiStep board
    case 'P': if(inStr[idx+1] == 'A') {
                cmd=PA;
                responder(0);
                router();

                gPIO->drivePower(i,1);        // turn OptiStep Board ON
                }
              else if(inStr[idx+1] == 'O') {
                cmd=PO;
                responder(0);
                router();

                gPIO->drivePower(i,0);        // turn OptiStep Board OFF
                }
              else {
                stat = 1;
                responder(1);
                }
              responder(1);
              return;
              break;

// The following are console-only commands
    // WV - Write Vacuum pressures to file
    case 'W': if(inStr[idx+1] == 'V') {
                cmd = CC;
                printf("writing vacuum to vacuum.log\n");
                if(!RecordVac) {
                  if((vacfp = fopen("vacuum.log", "at")) == NULL) {
                    printf("cannot open vacuum.log, abort record vacuum\n");
                    RecordVac=0;
                    break;
                    }
                  stamp = time(NULL);
                  lstamp = localtime(&stamp);
                  fprintf(vacfp, "%s", asctime(lstamp));
                  fprintf(vacfp, "Pirani\t\tCC Gauge\n");
                  fclose(vacfp);
                  RecordVac = 1;
                  printf("ready to record vacuum\n");
                  }
                else {
                  printf("already recording vacuum\n");
                  }
                }
    // WO - stop recording vacuum
              else if(inStr[idx+1] == 'O') {
                RecordVac=0;
                printf("stopped recording vacuum\n");
                }
              else {
                printf("unknown console command: %s\n", inStr);
                }
              break;
    case 'D': if(inStr[idx+1] == 'G') {
                cmd = CC;
                printf("DeGas camera tank motors\n");
                //printf("not yet implemented\n");
                i = DeGas();
                if(i) {
                  printf("Find home failed for:\n");
                  if(i && 1) {
                    printf("Tip\n");
                    }
                  if(i && 2) {
                    printf("Tilt\n");
                    }
                  if(i && 4) {
                    printf("CCD Focus\n");
                    }
                  }
                }
              else {
                printf("unknown console command: %s\n", inStr);
                }
              break;


    default:  stat=1;                 // flag for an error
              responder(1);           // invalid response
              responder(1);           // prompt
              return;
              break;

  } // ** end of switch(inStr[idx]) **

  return;

} // **--- end of cDispatcher::interpret definition ---**


// ****************************************************************
//                  router method definition
```

```
// *****************************************************************

void cDispatcher::router() {

  // =============================******=====================================
  // As of Sept 9, 1994, there are 9 active stepper motors within the echelle.
  // The pointer array elemets are as follows:
  //
  //    sList[0]=gCalMirror;      calibration mirror motor
  //    sList[1]=gEFilter;        echelle instrument filter motor
  //    sList[2]=gCFilter;        guider camera filter motor
  //    sList[3]=gGrating;        grating tilt motor
  //    sList[4]=gSlit;           slit focus motor
  //    sList[5]=gTip;            CCD tip addidude motor
  //    sList[6]-gTilt;           CCD tilt addidude motor
  //    sList[7]=gCCD;            CCD focus motors (two motors, one object)
  //
  // =============================******=====================================

  int n;

  switch(cmd) {

    case KIL:
//printf("Killing System\n");
      gVac->VacOff();
      gLamp->lampOff('A');
      gIon->IonOn();
      gPIO->drivePower(1,0);
      gPIO->drivePower(2,0);
      gPIO->drivePower(3,0);
      gPIO->drivePower(4,0);
      gPIO->pwr24V(1,0);
      gPIO->pwr24V(2,0);
      ShutDown = TRUE;
//printf("System killed.\n");
      break;

    case MVR:

      for(n=0;n<=7;n++) {                       // Move through the 8 mtrs
        if(nsteps[n][1] != 0)                      // if steps value=0, DON'T MOVE!
          switch(n) {
            case 0: if(lastOS != 1) {         // did we last talk with this
                      if(lastOS == 2)         // driver? If no, check which was

                        gOptiTwo->cReadStepCount();     // the last
                      if(lastOS == 3)                   // and read the board
                        gOptiThree->cReadStepCount();   // counters
                      if(lastOS == 4)
                        gOptiFour->cReadStepCount();
                      gOptiOne->cCardInit();            // next, reset board
                      lastOS = 1; }                     // to route commands
                    if(nsteps[n][0] <= 0)
                      gCalMirror->calibOut();           // now, move mirror out
                    else
                      gCalMirror->calibIn();            // or move mirror in
                   if(gCalMirror->position == -1) {
                    outStr = "[Calibration Mirror Move Error! :: ";
                    gError->cErrHndl();
                    responder(2);
                   } // end of if(gCalMirror->position == -1)... statement
                    break;

            case 1: if(lastOS != 1) {         // did we last talk with this
                      if(lastOS == 2)         // driver? If no, check which was

                        gOptiTwo->cReadStepCount();   // the last
                      if(lastOS == 3)                 // and read the board
                        gOptiThree->cReadStepCount(); // counters
                      if(lastOS == 4)
                        gOptiFour->cReadStepCount();
                      gOptiOne->cCardInit();          // next, reset board
                      lastOS = 1; }                   // to route commands
                    gEFilter->cMoveFilter(nsteps[n][0]); // then, move echelle ftr
                    break;

            case 2: if(lastOS != 1) {         // did we last talk with this
                      if(lastOS == 2)         // driver? If no, check which was

                        gOptiTwo->cReadStepCount();   // the last
                      if(lastOS == 3)
                        gOptiThree->cReadStepCount(); // and read the board
                      if(lastOS == 4)
                        gOptiFour->cReadStepCount();  // counters
                      gOptiOne->cCardInit();          // next, reset board
                      lastOS = 1; }                   // to route commands
                    gCFilter->cMoveFilter(nsteps[n][0]); // then, move guider ftr
                    break;

            case 3: if(lastOS != 2) {         // did we last talk with this
                      if(lastOS == 1)         // driver? If no, check which was

                        gOptiOne->cReadStepCount();   // the last
                      if(lastOS == 3)                 // and read the board
                        gOptiThree->cReadStepCount(); // counters
                      if(lastOS == 4)
                        gOptiFour->cReadStepCount();
                      gOptiTwo->cCardInit();          // next, reset board
                      lastOS = 2; }                   // to route commands
                    gGrating->cMoveMotor(nsteps[n][0]);  // then, move grating
                    if(gGrating->position != target_pos[n]) {
//printf("\n<gGrating->pos: %ld  nsteps[n][0]: %ld>\n", gGrating->position, nsteps[n][
0]);
//printf("\n<limitStat[0]: %d\tlimitStat[1]: %d>\n", gGrating->limitStat[0], gGrating-
>limitStat[1]);
                      outStr = "[Grating Move Error! :: ";
                      gError->cErrHndl();
                      responder(2);
                   } // end of if(gGrating->position != nsteps[n][0])... statement
                    break;

            case 4: if(lastOS != 2) {         // did we last talk with this
                      if(lastOS == 1)         // driver? If no, check which was

                        gOptiOne->cReadStepCount();   // the last
                      if(lastOS == 3)                 // and read the board
                        gOptiThree->cReadStepCount(); // counters
                      if(lastOS == 4)
                        gOptiFour->cReadStepCount();
                      gOptiTwo->cCardInit();          // next, reset board
                      lastOS = 2; }                   // to route commands
                    gSlit->cMoveMotor(nsteps[n][0]);      // then, move slit
                    if(gSlit->position != target_pos[n]) {
                      outStr = "[Slit Move Error! :: ";
                      gError->cErrHndl();
                      responder(2);
                   } // end of if(gSlit->position != nsteps[n][0])... statement
                    break;
```

```
        case 5: if(lastOS != 3) {        // did we last talk with this
                  if(lastOS == 1)        // driver? If no, check which was

                    gOptiOne->cReadStepCount();   // the last
                  if(lastOS == 2)                 // and read the board
                    gOptiTwo->cReadStepCount();   // counters
                  if(lastOS == 4)
                    gOptiFour->cReadStepCount();
                  gOptiThree->cCardInit();        // next, reset board
                  lastOS = 3; }                   // to route commands
                gTip->cMoveMotor(nsteps[n][0]);      // lastly, move tip
                if(gTip->position != target_pos[n]) {
                  outStr = "[Tip Move Error! :: ";
                  gError->cErrHndl();
                  responder(2);
               } // end of if(gTip->position != nsteps[n][0])... statement
                break;

        case 6: if(lastOS != 3) {        // did we last talk with this
                  if(lastOS == 1)        // driver? If no, check which was

                    gOptiOne->cReadStepCount();   // the last
                  if(lastOS == 2)                 // and read the board
                    gOptiTwo->cReadStepCount();   // counters
                  if(lastOS == 4)
                    gOptiFour->cReadStepCount();
                  gOptiThree->cCardInit();        // next, reset board
                  lastOS = 3; }                   // to route commands
                gTilt->cMoveMotor(nsteps[n][0]);      // lastly, move tilt
                if(gTilt->position != target_pos[n]) {
                  outStr = "[Tilt Move Error! :: ";
                  gError->cErrHndl();
                  responder(2);
               } // end of if(gTilt->position != nsteps[n][0])... statement
                break;

        case 7: if(lastOS != 4) {        // did we last talk with this
                  if(lastOS == 1)        // driver? If no, check which was
                    gOptiOne->cReadStepCount();   // the last
                  if(lastOS == 2)                 // and read the board
                    gOptiTwo->cReadStepCount();   // counters
                  if(lastOS == 3)
                    gOptiThree->cReadStepCount();
                  gOptiFour->cCardInit();         // next, reset board
                  lastOS = 4; }                   // to route commands
                gCCD->cMoveMotor(nsteps[n][0]);       // now, move CCDfocus
                if(gCCD->position != target_pos[n])  {
                  outStr = "[CCD Focus Move Error! :: ";
                  gError->cErrHndl();
                  responder(2);
               } // end of if(gCCD->position != nsteps[n][0])... statement
                break;

        default: stat=1;
                 break;
      }  // end of switch(parm1)
  } // end of FOR loop
return;
break; // end of case MVR:


case ST: gSystem->systemStatus();
         break;

case RS: gSystem->systemReset();
         break;

case AC: prs_ac_cmd();
         break;

case IS: gSystem->interFaceStat();
         break;

case CI: if(lastOS != 1) {        // did we last talk with this
          if(lastOS == 2)        // driver? If no, check which was
            gOptiTwo->cReadStepCount();     // the last
          if(lastOS == 3)                   // and read the board
            gOptiThree->cReadStepCount();   // counters
          if(lastOS == 4)
            gOptiFour->cReadStepCount();
          gOptiOne->cCardInit();            // next, reset board
          lastOS = 1;                     // to route commands
         }
         gCalMirror->calibIn();           // or move mirror in
         if(gCalMirror->position == -1) {
            outStr = "[Calibration Mirror Move Error! :: ";
            gError->cErrHndl();
            responder(2);
         } // end of if(gCalMirror->position == -1)... statement

         break;

case CO: if(lastOS != 1) {        // did we last talk with this
          if(lastOS == 2)        // driver? If no, check which was
            gOptiTwo->cReadStepCount();     // the last
          if(lastOS == 3)                   // and read the board
            gOptiThree->cReadStepCount();   // counters
          if(lastOS == 4)
            gOptiFour->cReadStepCount();
          gOptiOne->cCardInit();            // next, reset board
          lastOS = 1;                     // to route commands
         }
         gCalMirror->calibOut();            // or move mirror in
         if(gCalMirror->position == -1) {
            outStr = "[Calibration Mirror Move Error! :: ";
            gError->cErrHndl();
            responder(2);
         } // end of if(gCalMirror->position == -1)... statement

         break;

case FH: switch(inStr[3]) {
          case '1': break;

          case '2': if(lastOS != 1) {        // did we last talk with this
                      if(lastOS == 2)        // driver? If no, check which was

                        gOptiTwo->cReadStepCount();   // the last
                      if(lastOS == 3)                 // and read the board
                        gOptiThree->cReadStepCount(); // counters
                      if(lastOS == 4)
                        gOptiFour->cReadStepCount();
                      gOptiOne->cCardInit();          // next, reset board
                      lastOS = 1; }                   // to route commands

                    gEFilter->findHome();
                    break;

          case '3': if(lastOS != 1) {        // did we last talk with this
                      if(lastOS == 2)        // driver? If no, check which was
```

```
                gOptiTwo->cReadStepCount();   // the last
              if(lastOS == 3)                 // and read the board
                gOptiThree->cReadStepCount(); // counters
              if(lastOS == 4)
                gOptiFour->cReadStepCount();
              gOptiOne->cCardInit();          // next, reset board
              lastOS = 1; }                   // to route commands

            gCFilter->findHome();
            break;

 case '4': if(lastOS != 2) {         // did we last talk with this
              if(lastOS == 1)        // driver? If no, check which was

                gOptiOne->cReadStepCount();   // the last
              if(lastOS == 3)                 // and read the board
                gOptiThree->cReadStepCount(); // counters
              if(lastOS == 4)
                gOptiFour->cReadStepCount();
              gOptiTwo->cCardInit();          // next, reset board
              lastOS = 2; }                   // to route commands

            gGrating->findHome();
            break;

 case '5': if(lastOS != 2) {         // did we last talk with this
              if(lastOS == 1)        // driver? If no, check which was

                gOptiOne->cReadStepCount();   // the last
              if(lastOS == 3)                 // and read the board
                gOptiThree->cReadStepCount(); // counters
              if(lastOS == 4)
                gOptiFour->cReadStepCount();
              gOptiTwo->cCardInit();          // next, reset board
              lastOS = 2; }                   // to route commands

            gSlit->findHome();
            break;

 case '6': if(lastOS != 3) {         // did we last talk with this
              if(lastOS == 1)        // driver? If no, check which was

                gOptiOne->cReadStepCount();   // the last
              if(lastOS == 2)                 // and read the board
                gOptiTwo->cReadStepCount();   // counters
              if(lastOS == 4)
                gOptiFour->cReadStepCount();
              gOptiThree->cCardInit();        // next, reset board
              lastOS = 3; }                   // to route commands

            gTip->findHome();
            break;

 case '7': if(lastOS != 3) {         // did we last talk with this
              if(lastOS == 1)        // driver? If no, check which was

                gOptiOne->cReadStepCount();   // the last
              if(lastOS == 2)                 // and read the board
                gOptiTwo->cReadStepCount();   // counters
              if(lastOS == 4)
                gOptiFour->cReadStepCount();
              gOptiThree->cCardInit();          // next, reset board
              lastOS = 3; }                   // to route commands
```

```
                      gTilt->findHome();
                      break;

            case '8': if(lastOS != 4) {         // did we last talk with this
                        if(lastOS == 1)         // driver? If no, check which was

                          gOptiOne->cReadStepCount();   // the last
                        if(lastOS == 2)                 // and read the board
                          gOptiTwo->cReadStepCount();   // counters
                        if(lastOS == 3)
                          gOptiThree->cReadStepCount();
                        gOptiFour->cCardInit();         // next, reset board
                        lastOS = 4; }                   // to route commands

                      gCCD->findHome();
                      break;

           } // end of switch statement
           break;

  case LA: n = 3;
           while(inStr[n] == 0x20 && inStr[n] != 0x0D)
              n++;
           switch(inStr[n]) {
              case 'W':
              case 'T':
              case 'F': gLamp->lampOn(inStr[n]);
                        break;
              case 'A': gLamp->lampOn(inStr[n]);
                        break;
              default:  outStr = "Invalid Lamp ID";
                        responder(2);
                        break;
           }  // end of swtich....

           break;

  case LO: n = 3;
           while(inStr[n] == 0x20 && inStr[n] != 0x0D)
              n++;
           switch(inStr[n]) {
              case 'W': gLamp->lampOff(inStr[n]);
                        break;
              case 'T': gLamp->lampOff(inStr[n]);
                        break;
              case 'F': gLamp->lampOff(inStr[n]);
                        break;
              case 'A': gLamp->lampOff(inStr[n]);
                        break;
              default:  outStr = "Invalid Lamp ID";
                        responder(2);
                        break;
           }  // end of swtich....

           break;

  case LX: n = 3;
           while(inStr[n] == 0x20 && inStr[n] != 0x0D)
              n++;
           switch(inStr[n]) {
              case 'W': gLamp->lampOff('A');
                        gLamp->lampOn(inStr[n]);
                        break;
              case 'T': gLamp->lampOff('A');
                        gLamp->lampOn(inStr[n]);
```

```
                    break;
          case 'F': gLamp->lampOff('A');
                    gLamp->lampOn(inStr[n]);
                    break;
          default:  outStr = "Invalid Lamp ID";
                    responder(2);
                    break;
       }  // end of swtich....

       break;
case VA: n = 3;
         while(inStr[n] == 0x20 && inStr[n] != 0x0D)
            n++;
         switch(inStr[n]) {
            case 'H': gVac->LoVacOn();
                      gVac->HiVacOn();
                      break;
            case 'L': gVac->LoVacOn();
                      break;
            case 'A': gVac->VacOn();
                      break;
            default:  outStr = "Invalid Vacuum ID";
                      responder(2);
                      break;
         }  // end of swtich....

         break;

case VO: n = 3;
         while(inStr[n] == 0x20 && inStr[n] != 0x0D)
            n++;
         switch(inStr[n]) {
            case 'H': gVac->HiVacOff();
                      break;
            case 'L': gVac->VacOff();
                      break;
            case 'A': gVac->VacOff();
                      break;
            default:  outStr = "Invalid Vacuum ID";
                      responder(2);
                      break;
         }  // end of swtich....

         break;

case ZC: switch(inStr[3]) {
          case '1': break;
          case '2': break;
          case '3': break;
          case '4': gOptiTwo->cCardInit();
                    gOptiTwo->cSetStepCount(3,0);
                    gOptiTwo->cReadStepCount();
                    gGrating->position = gOptiTwo->cnt3;
                    gGrating->stepCount = gOptiTwo->cnt3;
                    break;
          case '5': gOptiTwo->cCardInit();
                    gOptiTwo->cSetStepCount(1,0);
                    gOptiTwo->cReadStepCount();
                    gSlit->position = gOptiTwo->cnt1;
                    gSlit->stepCount = gOptiTwo->cnt1;
                    break;
          case '6': gOptiThree->cCardInit();
                    gOptiThree->cSetStepCount(2,0);
                    gOptiThree->cReadStepCount();
                    gTip->position = gOptiThree->cnt2;
                    gTip->stepCount = gOptiThree->cnt2;
                    break;
          case '7': gOptiThree->cCardInit();
                    gOptiThree->cSetStepCount(3,0);
                    gOptiThree->cReadStepCount();
                    gTilt->position = gOptiThree->cnt3;
                    gTilt->stepCount = gOptiThree->cnt3;
                    break;
          case '8': gOptiFour->cCardInit();
                    gOptiFour->cSetStepCount(1,0);
                    gOptiFour->cSetStepCount(2,0);
                    gOptiFour->cReadStepCount();
                    gCCD->position = gOptiFour->cnt1;
                    gCCD->stepCount = gOptiFour->cnt1;
                    break;
          default:  stat = 1;
                    break;
        } // end of switch statement
        gSave->write();
        break;

case PA: n = 3;
         while(inStr[n] == 0x20 && inStr[n] != 0x0D)
            n++;
         switch(inStr[n]) {
            case '1': gPIO->drivePower(1,1);        // turn OptiStep Board ON
                      break;
            case '2': gPIO->drivePower(2,1);        // turn OptiStep Board ON
                      break;
            case '3': gPIO->drivePower(3,1);        // turn OptiStep Board ON
                      break;
            case '4': gPIO->drivePower(4,1);        // turn OptiStep Board ON
                      break;
            case '5': gPIO->pwr24V(1,1);        // turn box #1 on
                      gPIO->pwr24V(2,1);        // turn box #2 on
                      break;
            default:  outStr = "Invalid Board ID";
                      responder(2);
                      break;
         }  // end of swtich....

         break;

case PO: n = 3;
         while(inStr[n] == 0x20 && inStr[n] != 0x0D)
            n++;
         switch(inStr[n]) {
            case '1': gPIO->drivePower(1,0);        // turn OptiStep Board Off
                      break;
            case '2': gPIO->drivePower(2,0);        // turn OptiStep Board Off
                      break;
            case '3': gPIO->drivePower(3,0);        // turn OptiStep Board Off
                      break;
            case '4': gPIO->drivePower(4,0);        // turn OptiStep Board Off
                      break;
            case '5': gPIO->pwr24V(1,0);        // turn box #1 off
                      gPIO->pwr24V(2,0);        // turn box #2 off
                      break;
            default:  outStr = "Invalid Board ID";
                      responder(2);
                      break;
         }  // end of swtich....

         break;
```

```
    case IA: gIon->IonOn();
            break;

    case IO: gIon->IonOff();
            break;

    case CC: printf("console-only command processed\n");
             break;

  } // end of switch(cmd) sequence

  return;
}

// ****************************************************************
//                 responder method definition
// ****************************************************************

void cDispatcher::responder(int respType) {

  int i;  // counter for cr->lf translation
  // this method is the interface method between the objects and serial port

  switch(respType) {
    case 0: gIOPort->writeToPort("\006");
            break;
    case 1: if(!stat) {
              gIOPort->writeToPort("\r>");
              printf("OK\n");
              }
            else {
              stat=0;
              gIOPort->writeToPort("\25[Invalid Command]");
              printf("[Invalid Command]\n");
            }
            break;

    case 2: gIOPort->writeToPort(outStr.GetBuffer(0));
            for(i=0; i<outStr.GetLength(); i++) {
              if(outStr[i] == '\r')
                outStr.SetAt(i, '\n');
              }
            printf("%s", outStr);
            break;                  // write any string sent to object to port

    default: break;

  } // ---- end of the switch(respType) statement ----

  return;
}



// ====================================================
// ********** prs_mv_cmd Method Definition ***********
// ====================================================

// **** The prs_mv_cmd method interprets each of the incomming
// **** MOVE commamnds as ABSOLUTE moves. Each of the step
// **** values in the inStr is a positive integer value.

// Commamd Format
//
// -> MV motor# step# CR
//
// or MV nnnnn nnnnn nnnnn nnnnn nnnnn nnnnn nnnnn nnnnn
//    01234567890123456789012345678901234567890123456789012345678901234567  (idx numbers)
//    0          1         2         3         4         5        (tens)

// **-- returns 1 if successful, 0 if error


int cDispatcher::prs_mv_cmd() {

  int p = 0, idx;
  cBasicMotor* objptr;      // this will be the pointer used for all motor objs
  CString tmpStr;           // create a temporary string object
  int move_a_motor = 0;       // local to indicate if motor will be moved

  for(idx=0;idx<=7;idx++) {  // the second element signifies whether to move (1)
    nsteps[idx][0] = 0;       // or not to move (0)
    nsteps[idx][1] = 0;
  }

  switch(inStr[1]) {        // check the next char in the string. should tell
    case 0x56:  cmd = MVR;  // if command is valid move (0x56 = 'V')
                break;
    default:    return(0);  // unexpected char = error condition!
                break;
  }

  for(idx=2;idx<=80;idx++) {
    if(idx>=inStr.GetLength())      // if it ever tries to read past the end of
      return(0);                    // inStr an error has occured

    if(p>7 || inStr[idx] == 0x0D) {
      if(move_a_motor)
        return(1);
      else
        return(0);
    } // end of if(p>7 || inStr[idx] == 0x0D)... statement
    else
      if(inStr[idx] == 0x20 && inStr[idx+1] != 0x0D) {   // look for the spaces
        idx++;
        tmpStr = inStr.Right(inStr.GetLength() - idx);

      if(tmpStr[0]=='N'||tmpStr[0]=='n')  // if the character is N set the don't move
flag
        nsteps[p][1] = 0;
      else {
        if(tmpStr[0]==0x2D || tmpStr[0]==0x2B || (tmpStr[0]<=0x39&&tmpStr[0]>=0x30)) {
          nsteps[p][0] = atol(tmpStr);
          if(chkrange(p,nsteps[p][0])) {
            nsteps[p][1]=1;              // set the move flag true
            move_a_motor = 1;            // this was a valid command after all
          } // end of if(chkrange(p,nsteps[p][0])) ... statement
          else {
            nsteps[p][1] = 0;
            return(0);                  // found that command was invalid!
          } // end of if(chkrange... else ...
        } // end of if(tmpStr[0]==0x2D || tmpS... loop
        else
          return(0);   // drop out... we found commnd not valid
      }  // end of else statement

      target_pos[p] = nsteps[p][0];
      if(nsteps[p][1] == 1 && p>2 ) {       // the first 3 motors move by index
        objptr = gList->getPtr(p);
        nsteps[p][0] = -(objptr->stepCount - nsteps[p][0]);   // do the absolute conve
```

```
rsion here
      }
      p++;
    } // end of if(inStr[idx] == 0x20 && inStr....   loop
  } // end of for(idx=2;idx<=80;idx++) loop
  return(1);
}

// ===================================================
// ********** prs_ac_cmd Method Definition ***********
// ===================================================

// Commamd structure
//
// -> AC motor# start end accel
//
// or AC n nnn nnnn nnnn
//    0123456789012345678 (idx numbers)

void cDispatcher::prs_ac_cmd() {

  int i;
  CString tmpStr;                 // create a temporary string object
  char tmp[10] = "";              // temporary storage for the parm value
  cBasicMotor* objptr;            // this will be the pointer used for
                                  // reference motor objects
  if(inStr[3]!=0x0D) {
    tmp[0] = inStr[3];
    objptr = gList->getPtr(atoi(tmp)); }
  else {
    stat = 1;
    return; }

  for(i=5;i<=7;i++)              // read the start speed (1st parm)
    if(inStr[i]!=0x0D)
      tmpStr+= inStr[i];
  else {
    stat = 1;
    return; }

  objptr->start = atoi(tmpStr);  // got the value of the 2nd parm
  tmpStr = "";

  for(i=9;i<=12;i++)             // read the end speed (3rd parm)
    if(inStr[i]!=0x0D)
      tmpStr+= inStr[i];
  else {
    stat = 1;
    return; }

  objptr->end = atoi(tmpStr);     // got the value of the 3rd parm
  tmpStr = "";

  for(i=14;i<=16;i++)             // read the accel (4th parm)
    if(inStr[i]!=0x0D)
      tmpStr+= inStr[i];
  else {
    stat = 1;
    return; }

  objptr->accel = atoi(tmpStr);   // got the value of the 4th parm
  tmpStr = "";

  return;
}

// ===================================================
// ************ chkrange Method Definition ************
// ===================================================

// This method checks the range of the incomming move parameter
// to see if it is a valid request. Returns True if valid request

BOOL cDispatcher::chkrange(int idx, long value) {  // idx is the index of the motor

   switch(idx) {

    case 0: if( value == 1 || value == 0 )
              return(TRUE);
            break;

    case 1: if( value >= 1 && value <= 6 )
              return(TRUE);
            break;

    case 2: if( value >= 1 && value <= 6 )
              return(TRUE);
            break;

    case 3: if( labs(value)<=999999 )
              return(TRUE);
            break;

    case 4: if( labs(value)<=999999 )
              return(TRUE);
            break;

    case 5: if( labs(value)<=999999 )
              return(TRUE);
            break;

    case 6: if( labs(value)<=999999 )
              return(TRUE);
            break;

    case 7: if( labs(value)<=999999 )
              return(TRUE);
            break;
   } // end of switch(value)...

  return(FALSE); // if we get this far it was not valid

} /* end of cDispaher::chkrange() method definition   */


// This fn moves the camera tank motors to degas the motors themselves
// It is very stupid, and simply moves each motor back and forth 10,000
// steps N times.  N is hard-wired @ 4 to start
// After moving the motor N cycles, it then does a findHome()
// There is no error handling beyond status reporting
// This should be issued from console only, sine it will destroy
// calibration/alignment of the tank!
int cDispatcher::DeGas(void)
{
  int i, j=0, N;

  printf("# of cycles: ");
  scanf("%d", &N);
```

```
  printf("Degassing camera tank.  This will take a while, so get some coffee\n");
  // Move tip, tilt, then CCD
  // N is 4
  printf("Moving tip...");
  for(i=0; i<N; i++) {
    gTip->cMoveMotor(-20000);
    waitNSeconds(60); // wait 1 minute for pumps to catch up
    gTip->cMoveMotor(20000);
    waitNSeconds(60);
    printf("%d ", i+1);
    }
  printf ("\nFindhome()...");
  j = gTip->findHome();
  if(j) printf("Find home failed!!!!!\n");
  else printf("Home found\n");

  printf("\nMoving tilt...");
  for(i=0; i<N; i++) {
    gTilt->cMoveMotor(-20000);
    waitNSeconds(60); // wait 1 minute for pumps to catch up
    gTilt->cMoveMotor(20000);
    waitNSeconds(60);
    printf("%d ", i+1);
    }
  printf ("\nFindhome()...");
  j += 2 * gTilt->findHome();
  if(j && 2) printf("Find home failed!!!!!\n");
  else printf("Home found\n");

  printf("\nMoving CCD...");
  for(i=0; i<N; i++) {
    gCCD->cMoveMotor(-10000);
    waitNSeconds(60); // wait 1 minute for pumps to catch up
    gCCD->cMoveMotor(10000);
    waitNSeconds(60);
    printf("%d ", i+1);
    }
  printf ("\nFindhome()...");
  j += 4 * gCCD->findHome();
  if(j &&  4) printf("Find home failed!!!!!\n");
  else printf("Home found\n");

  printf("\nDone with motor degas.\n");
  return(j);
}

void cDispatcher::waitNSeconds(float secsToWait) {      // will only do 32 seconds max

    clock_t atStart, now, tdiff;

    atStart = clock();
    tdiff = secsToWait*CLOCKS_PER_SEC;

    now = clock(); // get the ticks

    while( ((now = clock())-atStart) < tdiff ) {
     NULL;
    }

  return;
}
```

```
// CONTENTS : objSave.cpp --  This file contains the method definitions
//                            for cObjSave, a class used for file control
//                            of obj saves.
// ***------------------------------***-----------------------------***
//    SPECS --
//            Operating System: DOS 6.2
//            Language        : C++ (Microsoft Specific)
//            Compiler        : Microsoft Visual C++ 1.5 (medium Lib)
//
//    HISTORY --
//            Creation Date   : Oct 17, 1994
//            Edit Dates      : Nov 2, 1994  - now sets gZephyr->lastOS
//
//            Author          : TjM (University of Chicago, Yerkes Obs)
//
// ***------------------------------***-----------------------------***

#if defined(_WINDOWS)
#include "stdafx.h"
#else
#include <afx.h>
#endif

#include <iostreams.h>
#include <stdlib.h>
#include "opti1.hpp"          // for definitions of csteppermotor
#include "cFilter.hpp"        // cFilter object definition
#include "cMtrObL.hpp"        // Object list class
#include "dispatch.hpp"     // cdispatcher definitions
#include "objsave.hpp"
#include "pointers.hpp"       // extern object pointers

CStdioFile* gFile;

// ------------- Constructor Definitions -------------------------------------

cObjSave::cObjSave() {

  fileName = "init.mtr";      // the file name
  gFile = new CStdioFile();   // create the object (file is not opened yet)
}


// ------------- Method Definitions ------------------------------------------

BOOL cObjSave::read() {

  int motor, idx = 0;
  cBasicMotor* objptr;        // generic pointer used for all the motor objects
  CString inStr = "";       // create string object to send to dispatcher


  if(!gFile->Open("init.mtr",CFile::modeRead | CFile::typeText))
    return(0);

  gFile->SeekToBegin();     // start at the beginning of file

  for(motor=0;motor<=7;motor++) {
    gFile->ReadString(inStr.GetBuffer(0),30);
    objptr = gList->getPtr(motor);        // set pointer to the proper object
    objptr->position = atol(inStr.GetBuffer(0));// assign motor position
    objptr->stepCount=objptr->position;
    inStr="";                             // NULL the string for the next round

    switch(motor) {
      case 0: gCalMirror->position = 0;
              break;
      case 1: gEFilter->position = objptr->position;
              break;
      case 2: gCFilter->position = objptr->position;
              break;
      case 3: gOptiTwo->cCardInit();
              gOptiTwo->cSetStepCount(1,objptr->position);
              break;
      case 4: gOptiTwo->cSetStepCount(2,objptr->position);
              break;
      case 5: gOptiThree->cCardInit();
              gOptiThree->cSetStepCount(2,objptr->position);
              break;
      case 6: gOptiThree->cSetStepCount(3,objptr->position);
              break;
      case 7: gOptiFour->cCardInit();
              gOptiFour->cSetStepCount(1,objptr->position);
              gOptiFour->cSetStepCount(2,objptr->position);
              break;
    } // end of switch(motor)
  } // end of for statement

  gZephyr->lastOS = 4;      // sets this cDispatcher data so we won't lose
  gFile->Close();           // data later when the first MV command occurs
  return(1);
}

BOOL cObjSave::write() {

  int motor;
  cBasicMotor* objptr;        // generic pointer used for all the motor objects
  CString tmp_buf = "";     // used for itoa function
  CString outStr = "";      // create string object


  if(!gFile->Open("init.mtr",CFile::modeCreate|CFile::modeRead|CFile::typeText))
    return(0);

  for(motor=0;motor<=7;motor++) {
    tmp_buf = "";                         // clear the bufer fo the next item
    outStr= "";                           // add a couple of spaces
    objptr = gList->getPtr(motor);    // set pointer to the proper object
    ltoa(objptr->position,tmp_buf.GetBuffer(0),10);   // get the position
    outStr+= tmp_buf.GetBuffer(0);    // add it to the outStr object
    outStr+= "\r\n";
    gFile->Write(outStr.GetBuffer(outStr.GetLength()),outStr.GetLength());
  }

  gFile->Close();
  return(1);

}
```

```
// CONTENTS : cPIO.cpp -- This file contains the class definition
//                       for cPIO class. cPIO controls the power
//                       distribution for the motor enclosures.

// ***-----------------------------***----------------------------***
//    SPECS --
//            Operating System: DOS 6.2
//            Language        : C++ (Microsoft Specific)
//            Compiler        : Microsoft Visual C++ 1.5 (medium Lib)
//
//    HISTORY --
//            Creation Date   : Jan 16, 1995
//            Edit Dates      : Jan 19, 1995 - added generic read and write port
//                            :                methods
//                            : Jan 30, 1995 - mods to pwr24V method
//                            : Feb 08, 1995 - added pwr24V(1,1)to constructor
//                            : Apr 28, 1995 - added pwr24V(2,1)to constructor
//                            : Apr 28, 1995 - commented out pwr24V() in
//                                             drivePower().
//                            : Apr 29, 1996 - added #include "stdafx.h" for
//                                           - Windows compatability
//                            : Feb 26, 1997 - changed OR and & values for Box 2
//                                           - ON/OFF...could be wiring error?
//            Author          : TjM (University of Chicago, Yerkes Obs)
//
// ***-----------------------------***----------------------------***

#if defined(_WINDOWS)
#include "stdafx.h"
#endif

#if defined(_DOS)
#include <afx.h>
#endif

#include <conio.h>      // needed for port i/o
#include "cPIO.hpp"     // class definitions

//-----------------------------------------------------------------------
//---  ***NOTE*** The relays in the Enclosures are ACTIVE LOW!!!    ---
//-----------------------------------------------------------------------

//-----------------------------------------------------------------------
//--- The PIO-24 Control Register must be configured for our use. The
//--- Control Register has the following format:
//---
//---   D7  D6  D5    D4      D3      D2      D1      D0
//---   X   X   X     PtA     PtC(U)  X       PtB     PtC(L)
//---
//--- Input=1, Output=0. Port C can be split between input and output.
//--- For our routine:
//---
//---      PortA=Output   PortB=Output    PortC=Input
//-----------------------------------------------------------------------

/**********************************************************************
/
/ PIO Pinout 2/26/97
/
/ Pin     Port    I/O     Discription
  1                       N/C
  2                       N/C
  3       B7      O       N/C
  4       B6      O       N/C
  5       B5      O       N/C
  6       B4      O       N/C
  7       B3      O       Fe/Ne Lamp control
  8       B2      O       Th/Ar Lamp control
  9       B1      O       White Lamp control
  10      B0      O       Ion Pump Control
  .
  .
  .
  22      C7      I       Spare Hall Effect Switch
  23      C6      I       Cal Flat Hall Effect Switch
  24      C5      I       CCD Focus Hall Effect Switch
  25      C4      I       Tilt Hall Effect Switch
  26      C3      I       Tip Hall Effect Switch
  27      C2      I       Ion Pump H.V. Status
  28      C1      I       Grating Hall Effect Switch
  29      C0      I       Slit Hall Effect Switch
  30      A7      O       Hi Vacuum Sensor Power
  31      A6      O       Lo Vacuum Sensor Power
  32      A5      O       Driver Power (Board #1, Enclosure #2)
  33      A4      O       24V Power (Enclosure #1)
  34      A3      O       Driver Power (Board #2, Enclosure #2)
  35      A2      O       24V Power (Enclosure #1)
  36      A1      O       Driver Power (Board #2, Enclosure #1)
  37      A0      O       Driver Power (Board #1, Enclosure #1)

  */

//***--------------------------------------------------------------***
//                      CONSTRUCTOR DEFINITION
//***--------------------------------------------------------------***

cPIO::cPIO() {

  BaseAddr = 0x300;
  outp(BaseAddr+3,0xFF);         // reset of the card, BaseAddr+3 = Control Reg
printf("\t\tPIO reset\n");
  outp(BaseAddr+3,0x09);         // Port A & B = output, C = input
printf("\t\tPIO ctrl reg setup\n");
  outp(BaseAddr, 0xFF);          // pull port A High
printf("\t\tPull Port A High\n");
  outp(BaseAddr+1, 0xFF);        // pull port B High
printf("\t\tPull Port B High\n");

  pwr24V(1,1);                   // turn on box #1 & #2...This will stay on
printf("\t\tBox 1 24V power on\n");
  pwr24V(2,1);                   // indefinatly.
printf("\t\tBox 2 24V power on\n");

} // end of constructor

cPIO::~cPIO() {

//  pwr24V(1,0);
//  pwr24V(2,0);
  // New as of 4/2/97...Pull the bus high (Active Low) to shutdown systems properly
  outp(BaseAddr, 0xFF);          // pull port A High
  outp(BaseAddr+1, 0xFF);        // pull port B High


}


void cPIO::pwr24V(int box, int onoff) { // turns +24V on/off. "onoff" is
                                        // 1 or 0. Box is 1 or 2.
  int port;
```

```
  port=readPort('A');                    // read the present port value

  switch(box) {
    case 1: if(onoff)
              outp(BaseAddr,port&0xEF);  // Box 1 Power On
            else
              outp(BaseAddr,port|0xD3);  // Box 1 Power Off
            break;                       // also turn off pwr to brds in enclosure
    case 2: if(onoff)
              outp(BaseAddr,port&0xFB);  // Box 2 Power On ***Note*** was DF
            else
              outp(BaseAddr,port|0x2C);  // Box 2 Power Off ***Note was 0x2C
            break;                       // also turn off pwr to brds in enclosure
    default:break;
  }

  return;
}

void cPIO::drivePower(int brd, int onoff) {

  int port;

  port=readPort('A');              // read the present port value

  switch(brd) {
    case 1: if(onoff==1)
              outp(BaseAddr,port&0xFE);  // Board 1-1 ON
            else
              outp(BaseAddr,port|0x01);  // Board 1-1 OFF
            break;
    case 2: if(onoff==1)
              outp(BaseAddr,port&0xFD);  // Board 1-2 ON
            else
              outp(BaseAddr,port|0x02);  // Board 1-2 OFF
            break;
    case 3: if(onoff==1) {
//              pwr24V(2,1);               // turn box #2 ON
              outp(BaseAddr,port&0xDF);  // Board 2-1 ON
            }
            else  {
              outp(BaseAddr,port|0x20);  // Board 2-1 OFF
//              pwr24V(2,0);               // turn box #2 OFF
            }
            break;
    case 4: if(onoff==1)  {
//              pwr24V(2,1);               // turn box #2 ON
              outp(BaseAddr,port&0xF7);  // Board 2-2 ON
            }
            else  {
              outp(BaseAddr,port|0x08);  // Board 2-2 OFF
//              pwr24V(2,0);               // turn box #2 OFF
            }
            break;
    default: // errCode = 32;     // need to add a new error code for this case
             break;
  }
}


//-----------------------------------------------------------------------------
//---     cPIO::readPort
//---
//--- Generic read of a PIO Board Port.
//---
//-----------------------------------------------------------------------------

int cPIO::readPort(char port) {        // generic read of the PIO Board Ports

  int portValue;

  switch(port) {
    case 'A': portValue=inp(BaseAddr);
              break;
    case 'B': portValue=inp(BaseAddr+1);
              break;
    case 'C': portValue=inp(BaseAddr+2);
              break;
  }
  return(portValue);

}

//-----------------------------------------------------------------------------
//---     cPIO::writePort
//---
//--- Generic write to a PIO Board Port.
//---
//-----------------------------------------------------------------------------

void cPIO::writePort(char port, int value) {

  switch(port) {
    case 'A': outp(BaseAddr,value);
              break;
    case 'B': outp(BaseAddr+1, value);
              break;
    case 'C': outp(BaseAddr+2,value);
              break;
  }
  return;
}
```

```
// CONTENTS : cLamp.cpp -- This file contains the method definition
//                         for class cLamp. cLamp class will control
//                         the white light, Fe/Ne and Th/Ar lamps.
// ***------------------------------***------------------------------***
//    SPECS --
//            Operating System: DOS 6.2
//            Language        : C++ (Microsoft Specific)
//            Compiler        : Microsoft Visual C++ 1.5 (medium Lib)
//
//    HISTORY --
//            Creation Date   : Jan 23, 1995
//            Edit Dates      : Sep 16, 1997 - added new command LX and new
//                                           - parameter 'A' for all
//                            : 24 Sep 97 - fixed readPort in cPIO => lamps work
//                            : 25 Sep 97 - Changed lampName to lampW/T/F for status
//
//            Author          : TjM (University of Chicago, Yerkes Obs)
//
// ***------------------------------***------------------------------***

#if defined(_WINDOWS)
#include "stdafx.h"
#endif
#if defined(_DOS)
#include <afx.h>
#endif

#include "cPIO.hpp"
#include "cLamp.hpp"

extern cPIO*  gPIO;
extern int errCode;

// ***------------------------------***------------------------------***
//
//    There are 3 lamps that will be used with the Echelle Spectrograph.
//    The lamps are: a white light source; a Fe/Ne lamp; and a Th/Ar
//    lamp. The lamps will be controlled via this class. The Clamp
//    class will control the lamps via the PIO-24 card and the cPIO
//    class.
//
//    The lamps are actuated by bringing bits B1-B3 of the PIO-24 low.
//
//    B1 = White Light Source
//    B2 = Fe/Ne Light Source
//    B3 = Th/Ar Light Source
//
//    The lampStat is an integer array with information indicating the
//    state of each lamp. 0 = OFF and 1 = ON.
//
// ***------------------------------***------------------------------***



cLamp::cLamp() {

    lampW="off";
    lampT="off";
    lampF="off";
}

void cLamp::lampOn(char lamp) {

  int port;

  port = gPIO->readPort('B');   // read the present state of port


  switch(lamp) {
    case 'W': gPIO->writePort('B',port&0xFD);
              lampW="on";
              break;
    case 'F': gPIO->writePort('B',port&0xFB);
              lampF="on";
              break;
    case 'T': gPIO->writePort('B',port&0xF7);
              lampT="on";
              break;
    case 'A': lampOn('W');     // turn on all lamps with recursive calls
              lampOn('F');
              lampOn('T');
              break;
    default:  errCode = 100;      // syntax error
              break;
  }
  return;
}

void cLamp::lampOff(char lamp) {

  int port;

  port = gPIO->readPort('B');   // read the present state of port

  switch(lamp) {
    case 'W': gPIO->writePort('B',port|0x02);
              lampW="off";
              break;
    case 'T': gPIO->writePort('B',port|0x04);
              lampT="off";
              break;
    case 'F': gPIO->writePort('B',port|0x08);
              lampF="off";
              break;
    case 'A': lampOff('W');     // turn off all the lamps with recursive calls
              lampOff('F');
              lampOff('T');
              break;
    default:  errCode = 1;
              break;
  }
  return;
}
```

```
/* CONTENTS :  Serial -- This file contains the function definitions for
               the serial i/o routines.

   ***-------------------------------***-----------------------------***
      SPECS --
              Operating System: DOS 6.2
              Language        : C (Microsoft Specific)
              Compiler        : Microsoft Visual C++ 1.5 (medium Lib)

      HISTORY --
              Creation Date   : June 21, 1994
              Edit Dates      : July 13, 1994 - added serial hardware interupts
                              : Sept 8, 1994  - removed use of timeout interrupt vecto
r
                              :               - now use cTimer class in readSerial sub
routine

              Author          : TjM (University of Chicago, Yerkes Obs)

   ***------------------------------***-----------------------------***/

#include <stdio.h>
#include <conio.h>
#include <dos.h>
#include "cTimer.hpp"
extern "C" {
#include "serial.h"   /* header file with async definitions */
#include "timer.h"

#define TRUE  1
#define FALSE 0

static char serial_buffer[SERIALBUFSIZE];
char *nextserialin = serial_buffer;
char *nextserialout = serial_buffer;
int buffer_count;

int SERIALPORT = 2;
int PARITY = 0;
int STOPBITS = 1;
int WORDLEN = 8;
int BAUD = 2400;
int timeOut = 1;

int xonxoff_enabled = TRUE;
static int XON_input_pending;
static XON_output_pending;

/* --- serial port init parameter byte ----- */
static union {
  struct {
    unsigned wordlen  : 2;
    unsigned stopbits : 1;
    unsigned parity   : 3;
    unsigned brk      : 1;
    unsigned divlatch : 1;
  } serial_initial_bits;
  char serial_initial_char;
} initserial;

static void (interrupt far *oldserialint)();
static void interrupt far newserialint();

/* ----- initialize the serial port interrupt vectors ----- */
void  initserialvect() {

/* ---- attach to the serial interupt vector ---- */
  if(oldserialint == NULL)
    oldserialint = _dos_getvect(SERIALINT);

  _dos_setvect(SERIALINT, newserialint);
  outp(MODEMCTL, (inp(MODEMCTL) | DTR | RTS | OUT2));
  outp(PIC01, (inp(PIC01) & SERIALIRQ));
  outp(INTENABLE, DATAREADY);
  outp(PIC00, EOI);

/* -- flush existing serial interrupts ---- */
  inp(RXDATA);
  inp(INTIDENT);
  inp(LINESTATUS);
  inp(MODEMSTATUS);

} /* --- End of intiserialport Definition --- */

/***************** restore_serialint ****************************/

void restore_serialint() {

  if(oldserialint)
    _dos_setvect(SERIALINT, oldserialint);
}

/***************** clear_serial_Q ******************************/

void clear_serial_Q () {

  nextserialin = nextserialout = serial_buffer;
  buffer_count = 0;

} /* -- end of clear_serial_Q definition ----- */

/****************** newserialint ******************************/

static void interrupt far newserialint() {

  int chr;
  outp(PIC00,EOI);
  if(nextserialin == serial_buffer+SERIALBUFSIZE)
    nextserialin = serial_buffer;
  chr = inp(RXDATA);
  if(xonxoff_enabled)
    if(chr == XOFF)
      XON_input_pending = 1;
    else if(chr == XON)
      XON_input_pending = 0;
  if(!xonxoff_enabled || (chr != XON && chr != XOFF)) {
    *nextserialin++ = (char) chr;
    buffer_count++;
  }
  if(xonxoff_enabled && !XON_output_pending && buffer_count > THRESHLD)
  { while((inp(LINESTATUS) & XMIT_DATA_READY) == 0)
      ;  /* wait here for conditions */
    outp(TXDATA, XOFF);
    XON_output_pending = 1;
  }
} /* -- end of newserialint definition ----------  */


int howManyChars() {
```

```
  return buffer_count;
}


/***************** readserial Method *******************************/

int readSerial() {

  cTimer pTimer;                /* instantiate an object to do the time out checks */

  pTimer.cStartTimer();         /* start the timer here */
  while (!input_char_ready())
    if(pTimer.cChkTimer(.01))   /* check the timer...if more than 1 millisecond has pa
ssed return */
      return 0;

  if(nextserialout == serial_buffer+SERIALBUFSIZE)
    nextserialout = serial_buffer;
    --buffer_count;

  if(XON_output_pending && buffer_count < SAFETYLVL) {
    XON_output_pending = 0;
    writeserial(XON);
  }

  return *nextserialout++;

} /* -- end of readSerial function ----- */


/***************** writeSerial Method *******************************/

int writeserial(int chr) {

  while(XON_input_pending)
    ;     /* wait here while XON pending to clear */

  set_timer(timeOut);
  while((inp(LINESTATUS) &  XMIT_DATA_READY) == 0)
    if(timed_out())
      return FALSE;
  outp(TXDATA, chr);
  return TRUE;

} /* --- end of writeSerial function --- */


/******************* chkForRTN Method *****************************/

int chkForRTN() {

  if(serial_buffer[buffer_count-1] == 0x0d)     // 0x0d is hex code for RETURN key
    return(1);
  else
    return(0);

} /* --- end of chkForRTN Method --- */

} /* end of the extern "C" linkage directive */
```

```
// CONTENTS : cfltrdos.cpp -- This file contains the class definition
//                              for cFilter(MS-DOS version only).

// ***-------------------------------***-----------------------------***
//    SPECS --
//            Operating System: DOS 6.2
//            Language        : C++ (Microsoft Specific)
//            Compiler        : Microsoft Visual C++ 1.5 (medium Lib)
//
//    HISTORY --
//            Creation Date   : Aug 19, 1994
//            Edit Dates      : Aug 30, 1994  - added overload method
//                                              cReadStepCount() and
//                                              cMoveMotor(numofSteps)
//                            : Sept 26, 1994 - cMoveMotor now takes long int
//                            : Oct 29, 1994  - findHome now checks for detent
//                            : Oct 31        - added new cMoveMotor overload
//                                              methods to cSlit, cCCDFocus,
//                                              cGrating and cTipTilt classes
//                            : Dec 2, 1994   - added cSetToRamp() to all
//                                              motor move methods
//                                            - added findHome to cCCDfocus
//                                              and cTipTilt
//                            : Dec 10, 1994  - FindHome() method added to
//                                              cGrating class
//                            : Jan 4, 1995   - added cFilterWhl::updateCount()
//                                              to all cFilterWhl daughter objs
//                            : Jan 28, 1995  - added power method, box assign.
//                            : Feb 06, 1995  - moved ramping into daughter obj
//                            : Feb 10, 1995  - modified chkLimits() to read
//                                              Port C of PIO-24 card for HOME
//                                              status.
//                            : Mar 02, 1995  - eliminated spaces from "name"
//                            : Apr 04, 1995  - overloaded cReadStepCount() of
//                                              cCCDfocus class
//                            : Apr 28, 1995  - commented out part of power()
//                            : Apr 28, 1995  - gPIO->drivePower() now called in
//                                              object constructors
//                            : Jun 28, 1996  - took the ramping out of all the
//                                            - motor methods. This is due to
//                                            - resonance that the vacuum rated
//                                            - exibit in ramping up.
//                            : Feb 13, 1997  - change overdrv on calibin meth.
//                                            - chnged calib mir mtr spd = 150
//                            : Feb 14, 1997  - re-introduced ramping to calib
//                            : Feb 17, 1997  - changes to calibIn (Hall Effects)
//                            : Feb 25, 1997  - re-installed findHome() in
//                                            - cFilterWhl constructor, added
//                                            - error check on cFilterWhl::findHome
//                            : Feb 27, 1997  - cGrating::findHome now rtns Bool
//                            : Feb 28, 1997  - created new base motor class
//                                              cBasicMotor from which all motor
//                                              classes are daughters.
//                            : Mar 03, 1997  - changes in findHome() due to
//                                              Hall Effects being reverse logic
//                            : Jul 29, 1997  - added cursor changes to cmovemotor
//                            : Aug 19, 1997  - mods to BasicMotor::findHome()..
//                                              refinements
//                                            - re-installed ramping on CCD
//                                              Focus motors
//                                            - added cBasicMotor::FHMoveMotor()
//                                              used in findHome()
//                            : Aug 29, 1997  - cFilterWhl::findHome now uses
//                                              cBasicMotor::FHMoveMotor
//                                            - modified to run on MS-DOS ONLY!
//                                            - FilterWhl::findHome works if
//                                              hardware is adj properly
//                                            - added cGrating::findHome()
//                                              overload method
//                                            - added ramping back to motors
//                                              (grating, slit, tip/tilt)
//                            : Sep 08, 1997  - re-arranged cMoveFilter()
//                            : Sep 16, 1997  - removed gError->errHndl() and
//                                              placed in cDispatcher::router()
//                                            - overdrive the calflat after
//                                              hall switch engages
//                            : 24 Sep 97 - added errCode 9 and limitStat check to cCa
libration::calibOut()
//                                        - errCode 9 set if mirror didn't make it all
 the way out
//                            : 06 Oct 97 - changed if(pos==0) to if(n<0) in cFilterWh
l::cMoveFilter()
//                                          for cases 4 & 5
//                            :           - added test to calibIn() to check
//                                          if already in before 400 step move
//                            : 13 Oct 97 - Changed chkLimits() to not test errcode an
d return early.
//                            : 23 Apr 98 - Added CCDfocus:findHome() method to remove
 search by 1 from CCD find home
//                            : 30 Apr 98 - Changed all findHome() & cBasicMotor::cMov
eMotor() to reset motor speed to
//                                          standard speed set in constructor
//                            : 19 May 98 - Changed Cal Flat Hall bit to install Ion p
ump remote control
//                            : 20 May 98 - Changed findHome() routines to use 20, 30,
 or 40k step moves
//
//            Author          : TjM (University of Chicago, Yerkes Obs)
//
// ***-------------------------------***-----------------------------***

#if defined(_WINDOWS)
#include "stdafx.h"
#endif
#if defined(_DOS)
#include <afx.h>
#endif

#include <stdio.h>
#include <stdlib.h>
#include <time.h>
#include "optiparm.hpp"
#include "optistep.hpp"
#include "opti1.hpp"
#include "dispatch.hpp"
#include "cError.hpp"
#include "cPIO.hpp"
#include "cFilter.hpp"

#if defined(_WINDOWS)
#include "ccalibd.h"
#endif

extern COptiStep*   gOptiOne;
extern COptiStep*   gOptiTwo;
extern COptiStep*   gOptiThree;
extern COptiStep*   gOptiFour;
extern cDispatcher* gZephyr;
extern cError*      gError;
extern cPIO*        gPIO;
```

```
#if defined(_WINDOWS)
extern cCalibd      *gCalibd;   // used for status (debuging)
#endif

extern int errCode;

// ***---------------------------------------------------------------------***
// ----->           Definitions for cBasicMotor methods                 <-----
// ***---------------------------------------------------------------------***

cBasicMotor::cBasicMotor() { m_hall_bit = 0xFF;}           // default constructor

cBasicMotor::~cBasicMotor() { }          // default distructor

  /* ---------- End of Constructor definition ------------------- */


void cBasicMotor::cReadStepCount() {

  CStepperMotor::cReadStepCount();    // read all motor counts of OptiStep card
  switch(axisID) {
    case 1: stepCount = cnt1;
            break;
    case 2: stepCount = cnt2;
            break;
    case 3: stepCount = cnt3;
            break;
    case 4: stepCount = cnt1;
            break;

    default: stepCount = 0;
             break;

  }  // ---------------- end of switch(axisID) --------------------
  return;
} // ---------- end of cBasicMotor::cReadStepCount() definition ----


void cBasicMotor::cMoveMotor(long numofSteps) {

  gPIO->drivePower(whichOpti,1);        // turn OptiStep Board ON
  cSetSpeed(stpsPerSec);
  cReadStepCount();
//  cRampProfile(start,end,accel);     ->>> moved this line to daughter objects
  CStepperMotor::cMoveMotor(numofSteps);
//  gError->cErrHndl();    ->>> removed from here and placed in cDispather::router()
  chkLimits();
  cReadStepCount();
  gPIO->drivePower(whichOpti,0);        // turn OptiStep Board OFF
  return;

} // ----------- end of cBasicMotor::cMoveMotor() definition --------

// stripped down version of cBasicMotor::cMoveMotor() for findHome method

void cBasicMotor::FHMoveMotor(long numofSteps) {

  cReadStepCount();
  CStepperMotor::cMoveMotor(numofSteps);
//  cBasicMotor::cMoveMotor(numofSteps);
  chkLimits();
  cReadStepCount();
  return;
} // ----------- end of cBasicMotor::cFHmoveMotor() definition --------

BOOL cBasicMotor::findHome() {

  BOOL done = FALSE,           // marker to flag when the Home is found
       failed = FALSE;         // signal of failure

  int tries=0;                 // absolute bailout counter

  printf("start find home\n");
  gPIO->drivePower(whichOpti,1);        // turn OptiStep Board ON
  cSetSpeed(stpsPerSec);
  if(!limitStat[1])            // check fo a limit...if not then move to it
    FHMoveMotor(30000);        // chkLimits() called in cMoveMotor
  printf("finish 30,000 step move\n");
  printf("start hunt for hall effect [dx=100]\n");
  tries=0;
  while(!done && !failed){    // do this until success or failure
    FHMoveMotor(-100);         // move in steps of 100 toward other limit
    tries++;
    if(limitStat[0])           // if we hit the other limit switch we have failed
      failed = TRUE;
    if((hallStat|m_hall_bit)==m_hall_bit)  // check the hall effect switch (reverse lo
gic!)
      done = TRUE;             // drop out if the switch was detected
    if(tries > MAX_TRIES)
      failed=TRUE;
  } // end of while loop

  if(failed) {                 // we are at the limit...go back and look again
    printf("hall effect not tripped; trying again [dx=10]\n");
    failed = FALSE;
    tries=0;
    FHMoveMotor(30000);        // move back to same limit we started at.
    printf("finish 30,000 step move\n");
    while(!done && !failed){    // do this until success or failure
      FHMoveMotor(-10);           // move in steps of 10 toward other limit
      tries++;
      if(limitStat[0])
        failed = TRUE;
      if((hallStat|m_hall_bit)==m_hall_bit)
        done = TRUE;
      if(tries > MAX_TRIES)
        failed=TRUE;
    } // end of while loop
  } // end of if(failed) statement

  if(done) {                   // we have found home...must refine home again
    printf("hall effect tripped; refining position [dx=10]\n");
    done = FALSE;
    tries=0;

    while(!done && !failed){    // do this until success or failure
      FHMoveMotor(100);           // move in steps of 10 toward other limit
      tries++;
      if(limitStat[0])          // stop and check for Hall switch to turn off
        failed = TRUE;          // if it did not there is may be a bad switch
      if((hallStat|m_hall_bit)!=m_hall_bit)
        done = TRUE;            // hall switch is off now turn around and refine pos
      if(tries > MAX_TRIES)
        failed=TRUE;
    } // end of this while loop
//    cMoveMotor(5000);           // move away from switch on side we approched before
    done = FALSE;
    tries=0;
```

```
    while(!done && !failed){    // do this until success or failure
      FHMoveMotor(-10);           // move in steps of 10 toward other limit
      tries++;
      if(limitStat[0])
        failed = TRUE;
      if((hallStat|m_hall_bit)==m_hall_bit)
        done = TRUE;
      if(tries > MAX_TRIES)
        failed=TRUE;
    } // end of while loop
  } // end of if(done) statement

  if(done) {                  // found home...must refine home again
    printf("hall effect tripped; refining position [dx=5]\n");
    done = FALSE;
    tries=0;
    while(!done && !failed){    // do this until success or failure
      FHMoveMotor(10);          // move in steps of 10 toward other limit
      tries++;
      if(limitStat[0])          // stop and check for Hall switch to turn off
        failed = TRUE;          // if it did not there is may be a bad switch
      if((hallStat|m_hall_bit)!=m_hall_bit)
        done = TRUE;            // hall switch is off now turn around and refine pos
      if(tries > MAX_TRIES)
        failed=TRUE;
    } // end of this while loop
//    cMoveMotor(5000);           // move away from switch on side we approched before
    done = FALSE;
    tries=0;
    while(!done && !failed){    // do this until success or failure
      FHMoveMotor(-5);          // move in steps of 10 toward other limit
      tries++;
      if(limitStat[0])
        failed = TRUE;
      if((hallStat|m_hall_bit)==m_hall_bit)
        done = TRUE;
      if(tries > MAX_TRIES)
        failed=TRUE;
    } // end of while loop
  } // end of if(done) statement

  if(done) {                  // found home...must refine home again
    printf("hall effect tripped; refining position [dx=1]\n");
    done = FALSE;
    tries=0;
    cSetSpeed(70);
    while(!done && !failed){    // do this until success or failure
      FHMoveMotor(5);           // move in steps of 10 toward other limit
      tries++;
      if(limitStat[0])          // stop and check for Hall switch to turn off
        failed = TRUE;          // if it did not there is may be a bad switch
      if((hallStat|m_hall_bit)!=m_hall_bit)
        done = TRUE;            // hall switch is off now turn around and refine pos
      if(tries > MAX_TRIES)
        failed=TRUE;
    } // end of this while loop
//    cMoveMotor(5000);           // move away from switch on side we approched before
    done = FALSE;
    while(!done && !failed){    // do this until success or failure
      waitNSeconds(.5);         // this slows the motors down for more acc.
      FHMoveMotor(-1);          // move in steps of 10 toward other limit
      tries++;
      if(limitStat[0])
        failed = TRUE;
      if((hallStat|m_hall_bit)==m_hall_bit)
        done = TRUE;
      if(tries > MAX_TRIES)
        failed=TRUE;
    } // end of while loop
  } // end of if(done) statement

  cSetSpeed(stpsPerSec);                // Reset motor speed to normal operating value
  gPIO->drivePower(whichOpti,0);        // turn OptiStep Board OFF
  if(failed) printf("!!!!!!!find home failed\n");
  else printf("successfully found home\n");
  return(failed);
}

// method chkLimits() returns 0 for open limit switches or 1 for switch #1
// closed or 2 for switch #2 closed.

void cBasicMotor::chkLimits() {

  int statLmt, status;

  status = cLimitStatus();      // get the status of the switches

  statLmt = status&0x00FF;      // strip off high order bits

//--------------------------------****--------------------------------------
//---> Check the Hall Effect Switch status next.
//
//---> The Hall Effect switches are used as a "HOME" position sensor.
//---> The switch status is read using Port C of the PIO-24 card. The
//---> configuration is:
//--->
//--->    bit1: Slit Focus
//--->    bit2: Grating
//--->    bit3: Calibration Mirror
//--->    bit4: CCD Tip
//--->    bit5: CCD Tilt
//--->    bit6: CCD Focus
//--->    bit7: spare
///-->    bit8: spare
//--------------------------------****--------------------------------------

  hallStat = gPIO->readPort('C');   // read the Hall Effect's Status

  statAux = status>>6;  // strip off lower order bits that don't mater
                        // only interested in bit 6(M4-) and 7(M4+)
                        // for the filter wheels
//  if(!errCode) {
    limitStat[0] = 0;           // the switches are open!
    limitStat[1] = 0;
//    return;
//  }

  switch(axisID) {

    case 1: if(statLmt&0x01) {
              limitStat[0] = 1;
              limitStat[1] = 0; }
            else
              if(statLmt&0x02) {
                limitStat[0] = 0;
                limitStat[1] = 1; }
            return;
            break;
    case 2: if(statLmt&0x04) {
              limitStat[0] = 1;
```

```
            limitStat[1] = 0; }
          else
            if(statLmt&0x08) {
              limitStat[0] = 0;
              limitStat[1] = 1; }
          return;
          break;
    case 3: if(statLmt&0x10) {
            limitStat[0] = 1;
            limitStat[1] = 0; }
          else
            if(statLmt&0x20) {
              limitStat[0] = 0;
              limitStat[1] = 1; }
          return;
          break;
    case 4: if(statLmt&0x01) {
            limitStat[0] = 1;
            limitStat[1] = 0; }
          else
            if(statLmt&0x02) {
              limitStat[0] = 0;
              limitStat[1] = 1; }
          return;
          break;
  } // end of switch(axisID) statement

  return;
}

// UpdateCount method updates the OptiStep Board motor counts
// to the accurate position;

void cBasicMotor::updateCount() {

  switch(whichOpti) {

    case 1: switch(axisID) {
            case 1: gOptiOne->cnt1=cnt1;
                    break;
            case 2: gOptiOne->cnt2=cnt2;
                    break;
            case 3: gOptiOne->cnt3=cnt3;
                    break;
          }
          break;
    case 2: switch(axisID) {
            case 1: gOptiTwo->cnt1=cnt1;
                    break;
            case 2: gOptiTwo->cnt2=cnt2;
                    break;
            case 3: gOptiTwo->cnt3=cnt3;
                    break;
          }
          break;
    case 3: switch(axisID) {
            case 1: gOptiThree->cnt1=cnt1;
                    break;
            case 2: gOptiThree->cnt2=cnt2;
                    break;
            case 3: gOptiThree->cnt3=cnt3;
                    break;
          }

          break;
    case 4: gOptiFour->cnt1 = cnt1;
            gOptiFour->cnt2 = cnt2;
            break;

  } // end switch statement
  return;

} // end of cBasicMotor::updateCount method


void cBasicMotor::power(int onoff) {    // onoff: 1 = ON; 0 = OFF

//  pValue = gPIO->readPort('A');

//  switch(onoff) {
//    case 0: if(box == 1)
//              if(pValue&0x08)         // if true box #1 is off
//                gPIO->pwr24V(1,0);    // turn power off
//              else
//                if(pValue&0x10)       // if true box #2 is off
//                  gPIO->pwr24V(2,0);  // turn off box #2
//            break;
//    case 1: if(box == 1)
//              if(pValue&0x08)         // if true box #1 is off
//                gPIO->pwr24V(1,1);    // turn power on
//              else
//                if(pValue&0x10)       // if true box #2 is off
//                  gPIO->pwr24V(2,1);  // turn on box #2
//            break;
//    default: errCode =  100;          // syntax error (wrong arguments)
//
//  } // end of switch statement

  // Now we turn on the appropriate OptiStep board

  gPIO->drivePower(whichOpti,onoff);    // turn on-off the board

  return;
}  // end of cFilterWhl::power method




// ***----------------------------------------------------------------------***
// ----->           Definitions for cFilterWhl methods                  <-----
// ***----------------------------------------------------------------------***

cFilterWhl::cFilterWhl() { }           // default constructor

cFilterWhl::cFilterWhl(int axis, int optib, const char* n) {

  axisID = axis;
  whichOpti = optib;
  name = n;
  position = 1;            // this may need to be changed!!!
  box = 1;

  start = 400;
  end = 400;
  accel = 300;
  stpsPerSec = 300;       // make it sound better
  cSetSpeed(stpsPerSec);

  if(axis == 2)
    trigaddr = 0x0001;      // echelle filter bit is: 0x00FD
```

```
  else
    if(axis == 3)
      trigaddr = 0x0002;      // Guider filter bit is: 0x00FE

  gPIO->drivePower(whichOpti,1);  // turn the driver board on
  chkLimits();              // won't work unless power is on to Box #1
  gPIO->drivePower(whichOpti,0);  // turn the driver board off

  powrSetting = pwr0;                               // default power = 0%

//  findHome();      // wrong place for this call

} /* ---------- End of Constructor definition ------------------- */



void cFilterWhl::cMoveFilter(long pos) {

  long n, steps;

  n = position-pos;

  switch(labs(n)) {

    case 0: break;
    case 1: if(n<0)
              steps = 400;
            else
              steps = -400;

            cMoveMotor(steps);
            if(!errCode)
              position = pos;
            updateCount();
            break;
    case 2: if(n<0)
              steps = 800;
            else
              steps = -800;

            cMoveMotor(steps);
            if(!errCode)
              position = pos;
            updateCount();
            break;
    case 3: steps = 1200;
            cMoveMotor(steps);
            if(!errCode)
              position = pos;
            updateCount();
            break;
    case 4: if(pos == 1)
              steps = 800;
            else
              steps = -800;

            cMoveMotor(steps);
            if(!errCode)
              position = pos;
            updateCount();
            break;
    case 5: if(pos == 1)
              steps = 400;
            else
              steps = -400;

            cMoveMotor(steps);
            if(!errCode)
              position = pos;
            updateCount();
            break;
    default:break;
  } // end of switch(n) statement


//*** moved the following lines out or to another location in method ***

//  cRampProfile(start,end,accel);
//  cSetToRamp();                          // set ramping on move
//  cMoveMotor(steps);
//  gError->cErrHndl();
//  if(!errCode)
//    position = pos;
//  updateCount();

  return;

} // ----------- end of cFilterWhl::cMoveFilter() definition --------

void cFilterWhl::findHome() {

  int i, pos, count = 0, done = 0;           // count will keep track of error conditi
ons

  cSetSpeed(300);
  gPIO->drivePower(whichOpti,1);  // turn the driver board on
  waitNSeconds(.25);                 // wait here for switch to trigger
  chkLimits();              // won't work unless power is on to Box #1

  while (!done&&!(statAux&trigaddr)&&(count<500)) // 400 steps is filter setting
  {                                // if not in the detent
    FHMoveMotor(5);                // move 10 step
    waitNSeconds(.25);             // wait here for switch to trigger
    chkLimits();                   // won't work unless power is on to Box #1
    if(statAux&trigaddr)
      done = 1;
    count=count+5;                    // increment the counter
  }

  if(count>=500) {                 // do this in case of error..try again the other way
    while (!(statAux&trigaddr)&&(count>0))  // 400 steps is filter setting
    {                                // if not in the detent
      FHMoveMotor(-5);                // move 10 step
      waitNSeconds(.25);                  // wait here for switch to trigger
      chkLimits();              // won't work unless power is on to Box #1
      count=count-5;                      // increment the counter
    }
    if(count<=0) {                 // do this in case of error
      position = -1;                 // indeterminate position
      gPIO->drivePower(whichOpti,0);  // turn the driver board off
      return;
    }
  } // end of if(count>=500....


  done = 0;
  FHMoveMotor(-100);               // move it out of the detent
  for(i=0;i<4;i++) {
    FHMoveMotor(-10);                // move back to see if there is a detent
    waitNSeconds(.25);                 // wait here for switch to trigger
```

```
    chkLimits();              // won't work unless power is on to Box #1
    if(statAux&trigaddr)
      break;
  }
  if(!(statAux&trigaddr))           // if detent then wheel is now on a filter
    FHMoveMotor(i*10+100);                // if not, it was on a filter...move back
  else {
    FHMoveMotor(i*10+100);
    position = 1;                   // wheel is now at filter zero
    gPIO->drivePower(whichOpti,0);  // turn the driver board off
    return;
  }


  FHMoveMotor(100);                // move it out of the detent
  for(i=0;i<4;i++) {
    FHMoveMotor(10);                // move back to see if there is a detent
    waitNSeconds(.25);                // wait here for switch to trigger
    chkLimits();              // won't work unless power is on to Box #1
    if(statAux&trigaddr)
      break;
  }
  if(!(statAux&trigaddr))           // if detent then wheel is now on a filter
    FHMoveMotor(-i*10-100);               // if not, it was on a filter...move back
  else {
//    FHMoveMotor(-i*10-100);
    position = 1;                   // wheel is now at filter zero
    gPIO->drivePower(whichOpti,0);  // turn the driver board off
    return;
  }

/*  cMoveMotor(130);                  // move forward to see if there is a detent
  waitNSeconds(1);                // wait here for switch to trigger
  gPIO->drivePower(whichOpti,1);  // turn the driver board on
  for(i=0;i<10;i++){
    chkLimits();              // won't work unless power is on to Box #1
  }
  gPIO->drivePower(whichOpti,0);  // turn the driver board off
  if(!(statAux&trigaddr))           // if detent then wheel is now on a filter
    cMoveMotor(-130);                // if not, it was on a filter...move back
  else {
    position = 0;                   // wheel is now at filter zero
    return;
  }
//  cMoveMotor(270);                  // see if home indent is found
*/
  for(pos=0;pos<=6;pos++) {         // there are 6 posibilities to check
    FHMoveMotor(400);               // move to next filter position
    FHMoveMotor(-100);               // move it out of the detent
    for(i=0;i<4;i++) {
      FHMoveMotor(-10);               // move back to see if there is a detent
      waitNSeconds(.25);                // wait here for switch to trigger
      chkLimits();              // won't work unless power is on to Box #1
      if(statAux&trigaddr)
        break;
    }
    if(!(statAux&trigaddr))           // if detent then wheel is now on a filter
      FHMoveMotor(i*10+100);                // if not, it was on a filter...move back
    else {
      FHMoveMotor(i*10+100);
      position = 1;                   // wheel is now at filter zero
      gPIO->drivePower(whichOpti,0);  // turn the driver board off
      return;
    }
  } // end of the for loop
  if(pos>5)                         // if we did not find the filter
    position = -1;                  // error condition...position undetermined

  gPIO->drivePower(whichOpti,0);  // turn the driver board off
  return;
}

// ***---- End of cFilterWhl::findHome() method ----------------------------***


// ***---------------------------------------------------------------------***
// ----->                     cCCDfocus class definitions               <-----
// ***---------------------------------------------------------------------***

cCCDfocus::cCCDfocus(int axis, int optib) {

  axisID = axis;
  whichOpti = optib;
  name = "CCDFocs";
  position = 0;
  box = 2;
  m_hall_bit = 0xDF;          // PIO bit PC5 is the CCD Focus switch

  start = 100;
  end = 300;
  accel = 100;
  stpsPerSec = 300;
  cSetSpeed(stpsPerSec);

  gPIO->drivePower(whichOpti,1);  // turn the driver board on
  chkLimits();              // won't work unless power is on to Box #1
  gPIO->drivePower(whichOpti,0);  // turn the driver board off

  powrSetting = pwr0;                   // default power = 0%
  cRampProfile(start,end,accel);        // set initial ramping parameters
  cReadStepCount();                     // overloaded method of CStepperMotor

}

void cCCDfocus::cMoveMotor(long numofSteps) {  // relative moves

  cRampProfile(start,end,accel);
  cSetToRamp();                         // set ramping on move
  cBasicMotor::cMoveMotor(numofSteps);
  position = stepCount;
  updateCount();
  return;
}

void cCCDfocus::FHMoveMotor(long numofSteps) {

  cRampProfile(start,end,accel);
  cSetToRamp();                         // set ramping on move
  CStepperMotor::cMoveMotor(numofSteps);
  chkLimits();
  position = stepCount;
  updateCount();
  return;
}


void cCCDfocus::cReadStepCount() {     // overlaoded method of cFilter class

  CStepperMotor::cReadStepCount();   // read the board count
```

```
  stepCount = cnt1;                  // get the count of the motor
  return;
}

BOOL cCCDfocus::findHome() {

  BOOL done = FALSE,           // marker to flag when the Home is found
       failed = FALSE;         // signal of failure

  int tries=0;                 // attempt counter

  printf("start find home\n");
  gPIO->drivePower(whichOpti,1);        // turn OptiStep Board ON
  cSetSpeed(stpsPerSec);

  if(!limitStat[1])            // check fo a limit...if not then move to it
    FHMoveMotor(20000);        // chkLimits() called in cMoveMotor
  printf("finish 20,000 step move\n");
  printf("start hunt for hall effect [dx=100]\n");
  tries=0;
  while(!done && !failed){     // do this until success or failure
    FHMoveMotor(-100);          // move in steps of 100 toward other limit
    tries++;
    if(limitStat[0])           // if we hit the other limit switch we have failed
      failed = TRUE;
    if((hallStat|m_hall_bit)==m_hall_bit)  // check the hall effect switch (reverse lo
gic!)
      done = TRUE;             // drop out if the switch was detected
    if(tries > MAX_TRIES)
        failed=TRUE;
  } // end of while loop

  if(failed) {                 // we are at the limit...go back and look again
    printf("hall effect not tripped; trying again [dx=10]\n");
    failed = FALSE;
    tries=0;
    FHMoveMotor(20000);        // move back to same limit we started at.
    printf("finish 20,000 step move\n");
    while(!done && !failed){    // do this until success or failure
      FHMoveMotor(-10);            // move in steps of 10 toward other limit
      tries++;
      if(limitStat[0])
        failed = TRUE;
      if((hallStat|m_hall_bit)==m_hall_bit)
        done = TRUE;
      if(tries > MAX_TRIES)
        failed=TRUE;
    } // end of while loop
  } // end of if(failed) statement

  if(done) {                   // we have found home...must refine home again
    printf("hall effect tripped; refining position [dx=10]\n");
    done = FALSE;
    tries=0;

    while(!done && !failed){    // do this until success or failure
      FHMoveMotor(100);            // move in steps of 10 toward other limit
      tries++;
      if(limitStat[0])           // stop and check for Hall switch to turn off
        failed = TRUE;           // if it did not there is may be a bad switch
      if((hallStat|m_hall_bit)!=m_hall_bit)
        done = TRUE;             // hall switch is off now turn around and refine pos
      if(tries > MAX_TRIES)
        failed=TRUE;
    } // end of this while loop
//    cMoveMotor(5000);           // move away from switch on side we approched before
    done = FALSE;
    tries=0;
    while(!done && !failed){    // do this until success or failure
      FHMoveMotor(-10);            // move in steps of 10 toward other limit
      tries++;
      if(limitStat[0])
        failed = TRUE;
      if((hallStat|m_hall_bit)==m_hall_bit)
        done = TRUE;
      if(tries > MAX_TRIES)
        failed=TRUE;
    } // end of while loop
  } // end of if(done) statement

  if(done) {                   // found home...must refine home again
    printf("hall effect tripped; refining position [dx=5]\n");
    done = FALSE;
    tries=0;
    while(!done && !failed){    // do this until success or failure
      FHMoveMotor(10);           // move in steps of 10 toward other limit
      tries++;
      if(limitStat[0])           // stop and check for Hall switch to turn off
        failed = TRUE;           // if it did not there is may be a bad switch
      if((hallStat|m_hall_bit)!=m_hall_bit)
        done = TRUE;             // hall switch is off now turn around and refine pos
      if(tries > MAX_TRIES)
        failed=TRUE;
    } // end of this while loop
//    cMoveMotor(5000);           // move away from switch on side we approched before
    done = FALSE;
    tries=0;
    while(!done && !failed){    // do this until success or failure
      FHMoveMotor(-5);           // move in steps of 10 toward other limit
      tries++;
      if(limitStat[0])
        failed = TRUE;
      if((hallStat|m_hall_bit)==m_hall_bit)
        done = TRUE;
      if(tries > MAX_TRIES)
        failed=TRUE;
    } // end of while loop
  } // end of if(done) statement

  if(done) {                   // found home...must refine home again
    printf("hall effect tripped; refining position [dx=5, hall effect detrigger]\n");
    done = FALSE;
    tries=0;
    cSetSpeed(70);

    while(!done && !failed){    // do this until success or failure
      FHMoveMotor(5);            // move in steps of 10 toward other limit
      tries++;
      if(limitStat[0])           // stop and check for Hall switch to turn off
        failed = TRUE;           // if it did not there is may be a bad switch
      if((hallStat|m_hall_bit)!=m_hall_bit)
        done = TRUE;             // hall switch is off now turn around and refine pos
      if(tries > MAX_TRIES)
        failed=TRUE;
    } // end of this while loop
  } // end of if(done) statement

  cSetSpeed(stpsPerSec);                 // Reset motor speed to normal operating value
  gPIO->drivePower(whichOpti,0);         // turn OptiStep Board OFF
  if(failed) printf("!!!!!! find home failed !!!!!!\n");
```

```
  else printf("successfully found home\n");
  return(failed);
}

// ***---------------------------------------------------------------------***
// ----->                    cSlit class definitions                   <-----
// ***---------------------------------------------------------------------***

cSlit::cSlit(int axis, int optib) {

  axisID = axis;
  whichOpti = optib;
  name = "SlitFcs";
  position = 0;
  box = 1;
  m_hall_bit = 0xFE;          // PIO bit PC0 is the Slit Focus switch

  start = 70;
  end = 400;
  accel = 400;
  stpsPerSec = 400;
  cSetSpeed(stpsPerSec);

  gPIO->drivePower(whichOpti,1);  // turn the driver board on
  chkLimits();                // won't work unless power is on to Box #1
  gPIO->drivePower(whichOpti,0);  // turn the driver board off

  powrSetting = pwr0;                   // default power = 0%
  cRampProfile(start,end,accel);        // set initial ramping parameters
  cReadStepCount();                     // overloaded method of CStepperMotor
  return;

}

void cSlit::cMoveMotor(long numofSteps) {

  cRampProfile(start,end,accel);
  cSetToRamp();                         // set ramping on move
  cBasicMotor::cMoveMotor(numofSteps);
  position = stepCount;
  updateCount();
  return;
}

BOOL cSlit::findHome() {

  BOOL done = FALSE,          // marker to flag when the Home is found
       failed = FALSE;        // signal of failure

  int tries=0;                // absolute bailout counter

  printf("start find home\n");
  gPIO->drivePower(whichOpti,1);        // turn OptiStep Board ON
  cSetSpeed(stpsPerSec);
  if(!limitStat[1])           // check fo a limit...if not then move to it
    FHMoveMotor(30000);       // chkLimits() called in cMoveMotor
  printf("finish 30,000 step move\n");
  printf("start hunt for hall effect [dx=100]\n");
  tries=0;
  while(!done && !failed){    // do this until success or failure
    FHMoveMotor(-100);         // move in steps of 100 toward other limit
    tries++;
    if(limitStat[0])          // if we hit the other limit switch we have failed
      failed = TRUE;
    if((hallStat|m_hall_bit)==m_hall_bit)  // check the hall effect switch (reverse lo
gic!)
      done = TRUE;            // drop out if the switch was detected
    if(tries > MAX_TRIES)
      failed=TRUE;
  } // end of while loop

  if(failed) {                // we are at the limit...go back and look again
    printf("hall effect not tripped; trying again [dx=10]\n");
    failed = FALSE;
    tries=0;
    FHMoveMotor(30000);       // move back to same limit we started at.
    printf("finish 30,000 step move\n");
    while(!done && !failed){    // do this until success or failure
      FHMoveMotor(-10);           // move in steps of 10 toward other limit
      tries++;
      if(limitStat[0])
        failed = TRUE;
      if((hallStat|m_hall_bit)==m_hall_bit)
        done = TRUE;
      if(tries > MAX_TRIES)
        failed=TRUE;
    } // end of while loop
  } // end of if(failed) statement

  if(done) {                  // we have found home...must refine home again
    printf("hall effect tripped; refining position [dx=10]\n");
    done = FALSE;
    tries=0;

    while(!done && !failed){    // do this until success or failure
      FHMoveMotor(100);           // move in steps of 10 toward other limit
      tries++;
      if(limitStat[0])          // stop and check for Hall switch to turn off
        failed = TRUE;          // if it did not there is may be a bad switch
      if((hallStat|m_hall_bit)!=m_hall_bit)
        done = TRUE;            // hall switch is off now turn around and refine pos
      if(tries > MAX_TRIES)
        failed=TRUE;
    } // end of this while loop
//    cMoveMotor(5000);           // move away from switch on side we approched before
    done = FALSE;
    tries=0;
    while(!done && !failed){    // do this until success or failure
      FHMoveMotor(-10);           // move in steps of 10 toward other limit
      tries++;
      if(limitStat[0])
        failed = TRUE;
      if((hallStat|m_hall_bit)==m_hall_bit)
        done = TRUE;
      if(tries > MAX_TRIES)
        failed=TRUE;
    } // end of while loop
  } // end of if(done) statement

  if(done) {                  // found home...must refine home again
    printf("hall effect tripped; refining position [dx=5]\n");
    done = FALSE;
    tries=0;
    while(!done && !failed){    // do this until success or failure
      FHMoveMotor(10);          // move in steps of 10 toward other limit
      tries++;
      if(limitStat[0])          // stop and check for Hall switch to turn off
        failed = TRUE;          // if it did not there is may be a bad switch
      if((hallStat|m_hall_bit)!=m_hall_bit)
        done = TRUE;            // hall switch is off now turn around and refine pos
```

```
      if(tries > MAX_TRIES)
        failed=TRUE;
    } // end of this while loop
//    cMoveMotor(5000);           // move away from switch on side we approched before
    done = FALSE;
    tries=0;
    while(!done && !failed){    // do this until success or failure
      FHMoveMotor(-5);          // move in steps of 10 toward other limit
      tries++;
      if(limitStat[0])
        failed = TRUE;
      if((hallStat|m_hall_bit)==m_hall_bit)
        done = TRUE;
      if(tries > MAX_TRIES)
        failed=TRUE;
    } // end of while loop
  } // end of if(done) statement

  if(done) {                 // found home...must refine home again
    printf("hall effect tripped; refining position [dx=1]\n");
    done = FALSE;
    tries=0;
    cSetSpeed(70);
    while(!done && !failed){    // do this until success or failure
      FHMoveMotor(5);           // move in steps of 10 toward other limit
      tries++;
      if(limitStat[0])          // stop and check for Hall switch to turn off
        failed = TRUE;          // if it did not there is may be a bad switch
      if((hallStat|m_hall_bit)!=m_hall_bit)
        done = TRUE;            // hall switch is off now turn around and refine pos
      if(tries > MAX_TRIES)
        failed=TRUE;
    } // end of this while loop
//    cMoveMotor(5000);           // move away from switch on side we approched before
    done = FALSE;
    while(!done && !failed){    // do this until success or failure
      //waitNSeconds(.5);         // this slows the motors down for more acc.
      FHMoveMotor(-1);          // move in steps of 10 toward other limit
      tries++;
      if(limitStat[0])
        failed = TRUE;
      if((hallStat|m_hall_bit)==m_hall_bit)
        done = TRUE;
      if(tries > MAX_TRIES)
        failed=TRUE;
    } // end of while loop
  } // end of if(done) statement

  cSetSpeed(stpsPerSec);               // Reset motor speed to normal operating value
  gPIO->drivePower(whichOpti,0);       // turn OptiStep Board OFF
  if(failed) printf("!!!!!!!find home failed\n");
  else printf("successfully found home\n");
  return(failed);
}


// ***--------------------------------------------------------------------------***
// ----->               cCalibration class definitions                    <-----
// ***--------------------------------------------------------------------------***

cCalibration::cCalibration(int axis, int optib) {

  axisID = axis;
  whichOpti = optib;
  name = "CalbMir";
  position = 0;
  box = 1;
  m_hall_bit = 0x7F;          // PIO bit PC7 is the Cal flat switch

  start = 65;
  end = 100;
  accel = 100;
  stpsPerSec = 100;
  cSetSpeed(stpsPerSec);

  gPIO->drivePower(whichOpti,1);  // turn the driver board on
  chkLimits();                // won't work unless power is on to Box #1
  gPIO->drivePower(whichOpti,0);  // turn the driver board off

  powrSetting = pwr0;                    // default power = 0%
  cRampProfile(start,end,accel);         // set initial ramping parameters
  cReadStepCount();                      // overloaded method of CStepperMotor
  return;

}

void cCalibration::calibIn() {

// 440 steps should be 1.5 revolutions of the motor 2/13/97 - Heimsath
  int i;

// next commented out...should already be on in cPIO contstructor
//  gPIO->pwr24V(2,1);            // turn on power to Hall Effects Switches

  chkLimits();
  if((hallStat|m_hall_bit)==m_hall_bit) { // check to see if already in
    cMoveMotor(50);   // Now overdrive the motor to engage the magnet
    position = 1;
    updateCount();
//      gPIO->pwr24V(2,0);            // turn off power to Hall Effects Switches
    return;
    }

  cMoveMotor(400);  // move 400 steps...not quite there (150 steps to go)

  for(i=0;i<=75;i++)  { // move in steps of 2 (75x)= 150 steps total
    if((hallStat|m_hall_bit)==m_hall_bit) {
      cMoveMotor(50);   // Now overdrive the motor to engage the magnet
      position = 1;
      updateCount();
//      gPIO->pwr24V(2,0);            // turn off power to Hall Effects Switches
      return;
    }
    else {
      cMoveMotor(2);
//      waitNSeconds(.5); // wait 1/2 seconds between moves
    }
  }
  // didn't make it there!

  position = -1;
//  gPIO->pwr24V(2,0);            // turn off power to Hall Effects Switches

//  updateCount();
  return;

}

void cCalibration::calibOut() {
```

```
  cMoveMotor(-440);   // One revolution of the motor
  gPIO->drivePower(whichOpti,0);        // turn OptiStep Board OFF
  position = 0;
  if(limitStat[0] != 1) {
    position = -1;
    errCode = 9;
    }
  updateCount();
  return;

}

void cCalibration::cMoveMotor(long numofSteps) {

//  gPIO->drivePower(whichOpti,1);        // turn OptiStep Board ON
  cReadStepCount();
  cRampProfile(start,end,accel);
  cSetToRamp();
  cBasicMotor::cMoveMotor(numofSteps);    // why was it CStepperMotor:: before?
//  chkLimits();
  cReadStepCount();
  return;

} // ----------- end of cFilterWhl::cMoveMotor() definition --------


void cCalibration::lampOn(int lamp) { return; }

void cCalibration::lampOff() { return; }




// ***-----------------------------------------------------------------------***
// ----->               cGrating class definitions                     <-----
// ***-----------------------------------------------------------------------***

cGrating::cGrating (int axis, int optib) {

  axisID = axis;
  whichOpti = optib;
  name = "Grating";
  position = 0;
  box = 1;
  m_hall_bit = 0xFD;          // PIO bit PC1 is the Grating switch

  start = 70;
  end = 180;
  accel = 100;
  stpsPerSec = 180;
  cSetSpeed(stpsPerSec);

  gPIO->drivePower(whichOpti,1);  // turn the driver board on
  chkLimits();              // won't work unless power is on to Box #1
  gPIO->drivePower(whichOpti,0);  // turn the driver board off

  powrSetting = pwr0;                   // default power = 0%
  cRampProfile(start,end,accel);        // set initial ramping parameters
  cReadStepCount();                     // overloaded method of CStepperMotor
  return;

}

void cGrating::cMoveMotor(long numofSteps) {
//  cRampProfile(start,end,accel);        // set initial ramping parameters
//  cSetToRamp();                         // set ramping on move
  cBasicMotor::cMoveMotor(numofSteps);
  position = stepCount;
  updateCount();
  return;
}

BOOL cGrating::findHome() {

  BOOL done = FALSE,          // marker to flag when the Home is found
       failed = FALSE;        // signal of failure

  int tries;

  printf("start find home\n");
  gPIO->drivePower(whichOpti,1);        // turn OptiStep Board ON
  cSetSpeed(stpsPerSec);
  if(!limitStat[1])           // check fo a limit...if not then move to it
    FHMoveMotor(40000);       // chkLimits() called in cMoveMotor
  printf("finish 40,000 step move\n");
  printf("start hunt for hall effect [dx=100]\n");
  tries=0;
  while(!done && !failed){    // do this until success or failure
    FHMoveMotor(-100);         // move in steps of 100 toward other limit
    tries++;
    if(limitStat[0])          // if we hit the other limit switch we have failed
      failed = TRUE;
    if((hallStat|m_hall_bit)==m_hall_bit)  // check the hall effect switch (reverse lo
gic!)
      done = TRUE;            // drop out if the switch was detected
    if(tries > MAX_TRIES)
        failed=TRUE;
  } // end of while loop

  if(failed) {                // we are at the limit...go back and look again
    printf("hall effect not tripped; trying again [dx=10]\n");
    failed = FALSE;
    tries=0;
    FHMoveMotor(40000);       // move back to same limit we started at.
    printf("finish 40,000 step move\n");
    while(!done && !failed){    // do this until success or failure
      FHMoveMotor(-10);           // move in steps of 10 toward other limit
      tries++;
      if(limitStat[0])
        failed = TRUE;
      if((hallStat|m_hall_bit)==m_hall_bit)
        done = TRUE;
      if(tries > MAX_TRIES)
        failed=TRUE;
    } // end of while loop
  } // end of if(failed) statement

  if(done) {                  // we have found home...must refine home again
    printf("hall effect tripped; refining position [dx=10]\n");
    done = FALSE;
    tries=0;

    while(!done && !failed){    // do this until success or failure
      FHMoveMotor(100);           // move in steps of 10 toward other limit
      tries++;
      if(limitStat[0])          // stop and check for Hall switch to turn off
        failed = TRUE;          // if it did not there is may be a bad switch
      if((hallStat|m_hall_bit)!=m_hall_bit)
        done = TRUE;            // hall switch is off now turn around and refine pos
```

```
      if(tries > MAX_TRIES)
        failed=TRUE;
    } // end of this while loop
    done = FALSE;
    tries=0;
    while(!done && !failed){    // do this until success or failure
      FHMoveMotor(-10);           // move in steps of 10 toward other limit
      tries++;
      if(limitStat[0])
        failed = TRUE;
      if((hallStat|m_hall_bit)==m_hall_bit)
        done = TRUE;
      if(tries > MAX_TRIES)
        failed=TRUE;
    } // end of while loop
  } // end of if(done) statement

  if(done) {                  // found home...must refine home again
    printf("hall effect tripped; refining position [dx=5]\n");
    done = FALSE;
    tries=0;
    while(!done && !failed){    // do this until success or failure
      FHMoveMotor(10);          // move in steps of 10 toward other limit
      tries++;
      if(limitStat[0])          // stop and check for Hall switch to turn off
        failed = TRUE;          // if it did not there is may be a bad switch
      if((hallStat|m_hall_bit)!=m_hall_bit)
        done = TRUE;            // hall switch is off now turn around and refine pos
      if(tries > MAX_TRIES)
        failed=TRUE;
    } // end of this while loop
//    cMoveMotor(5000);           // move away from switch on side we approched before
    done = FALSE;
    tries=0;
    while(!done && !failed){    // do this until success or failure
      FHMoveMotor(-5);          // move in steps of 10 toward other limit
      tries++;
      if(limitStat[0])
        failed = TRUE;
      if((hallStat|m_hall_bit)==m_hall_bit)
        done = TRUE;
      if(tries > MAX_TRIES)
        failed=TRUE;
    } // end of while loop
  } // end of if(done) statement

  if(done) {                  // found home...must refine home again
    printf("hall effect tripped; refining position [dx=5, hall effect detrigger]\n");
    done = FALSE;
    tries=0;
    cSetSpeed(70);
    while(!done && !failed){    // do this until success or failure
      FHMoveMotor(5);           // move in steps of 10 toward other limit
      tries++;
      if(limitStat[0])          // stop and check for Hall switch to turn off
        failed = TRUE;          // if it did not there is may be a bad switch
      if((hallStat|m_hall_bit)!=m_hall_bit)
        done = TRUE;            // hall switch is off now turn around and refine pos
      if(tries > MAX_TRIES)
        failed=TRUE;
    } // end of this while loop
  } // end of if(done) statement

  cSetSpeed(stpsPerSec);                // Reset motor speed to normal operating value
  gPIO->drivePower(whichOpti,0);        // turn OptiStep Board OFF
  if(failed) printf("!!!!!! find home failed !!!!!!\n");
  else printf("successfully found home\n");
  return(failed);
}
// ***----END OF cGrating Class Methods------------------------------------***



// ***--------------------------------------------------------------------***
// ----->                cTipTilt class definitions                   <-----
// ***--------------------------------------------------------------------***

cTipTilt::cTipTilt(int axis, int optib, const char* n) {

  axisID = axis;
  whichOpti = optib;
  name = n;
  position = 0;
  box = 2;
  if(axis == 2)
    m_hall_bit = 0xF7;          // PIO bit PC3 is the Tip switch
  else
    m_hall_bit = 0xEF;          // PIO bit PC4 is the Tilt switch

  start = 400;
  end = 400;
  accel = 400;
  stpsPerSec = 300;
  cSetSpeed(stpsPerSec);

  gPIO->drivePower(whichOpti,1);  // turn the driver board on
  chkLimits();              // won't work unless power is on to Box #1
  gPIO->drivePower(whichOpti,0);  // turn the driver board off

  powrSetting = pwr0;                     // default power = 0%
//  cRampProfile(start,end,accel);          // set initial ramping parameters
  cReadStepCount();                       // overloaded method of CStepperMotor

}

void cTipTilt::cMoveMotor(long numofSteps) {

  cRampProfile(start,end,accel);        // set initial ramping parameters
  cSetToRamp();                         // set ramping on move
  cBasicMotor::cMoveMotor(numofSteps);
  position = stepCount;
  updateCount();
  return;
}

void cTipTilt::FHMoveMotor(long numofSteps) {

  cRampProfile(start,end,accel);        // set initial ramping parameters
  cSetToRamp();                         // set ramping on move
  CStepperMotor::cMoveMotor(numofSteps);
  chkLimits();
  position = stepCount;
  updateCount();
  return;
}
```

```
// CONTENTS : cVacuum.cpp -- This file contains the class definition
//                           for cVacuum class.

// ***-------------------------------***-----------------------------***
//    SPECS --
//            Operating System: DOS 6.2
//            Language        : C++ (Microsoft Specific)
//            Compiler        : Microsoft Visual C++ 1.5 (medium Lib)
//
//    HISTORY --
//            Creation Date   : Sept 13, 1994
//            Edit Dates      : Jan 22, 1995  - added VacOn & VacOff methods
//                            : Feb 09, 1995  - added VacOn to constructor
//                            : Feb 10, 1995  - modified VacOn() to turn both
//                                            - Hi Vac and Lo Vac power on @
//                                            - same time.
//                            : Apr 05, 1996  - added #include "stdafx.h" for
//                            :               - Windows compatability
//                            : May 14, 1996  - Corrected pressure problems.
//                            : Feb 03, 1997  - implement new read algorithm
//                            : Sep 09, 1997  - added is_vac_on flag
//                            : 10 Mar 98 - Removed sigma calculation
//                                        - Changed # of reads to 20
//                            : 30 Mar 98 - Added vacuum logging to file
//
//            Author          : TjM (University of Chicago, Yerkes Obs)
//
// ***------------------------------***-----------------------------***


#if defined(_WINDOWS)
#include "stdafx.h"
#endif
#if defined(_DOS)
#include <afx.h>
#endif

#include <conio.h>
#include <math.h>         // for port i/o functions inpw() and outp()
#include "cPIO.hpp"
#include "Vacuum.hpp"

extern cPIO        *gPIO;

// These are for vacuum logging
FILE *vacfp;
int RecordVac=0;

cVacuum::cVacuum() {

  pressure0 = 0.0;
  pressure1 = 0.0;
  dasBaseAddr = 0x2A0;

  outp(dasBaseAddr+0x09,0x00);    // initialize the DAS 1402 board

  outp(dasBaseAddr+0x02,0x10);    // set the ADC MUX to cycle between ch 0 & 1

  is_vac_on[0] = 0;
  is_vac_on[1] = 0;            // system is always started manually!!!

}

cVacuum::~cVacuum() {
  VacOff();                       // turn the sensors off.
  is_vac_on[0] = 0;
  is_vac_on[1] = 0;

}


void cVacuum::status() {

  dastatus = inp(dasBaseAddr+0x08);   // read the status reg of the DAS 1402
  return;
}

void cVacuum::readSensors() {

  int y, count0=0, count1=0, eN0 = 0, eN1 = 0;
  unsigned temp;
  double volt,
//  double volt0[100],
//         volt1[100],
         xi1 = 0,
         xi0 = 0;

  status();                          // read the status

  for(y=0;y<50;y++) {             // read the sensor 100 times
//    volt0[y] = 0;
//    volt1[y] = 0;
    volt=0.0;
    outp(dasBaseAddr,0x00);       // trigger the A/D conversion
    temp = inpw(dasBaseAddr);     // read channel of the DAS 1402 board
    if(temp & 0x0001) {           // channel 1
//      volt0[y] = (temp>>4)*2.44e-3; // right shift the ADC data so LSB first
////    volt0[y] = 8.5 - (y*.1);
//      if(volt0[y]>=1.50 && volt0[y]<=8.5) {
//        xi0 = xi0 + volt0[y];
      volt = (temp>>4)*2.44e-3; // right shift the ADC data so LSB first
//    volt0[y] = 8.5 - (y*.1);
      if(volt>=1.50 && volt<=8.5) {
        xi0 = xi0 + volt;
        eN0++;
        }
      }
    else {
//      volt1[y] = (temp>>4)*2.44e-3;  // 2.44mV per bit is the board conversion
////        volt1[y] = 8.5 - (y*.1);
//      if(volt1[y]>=1.50 && volt1[y]<=8.5) {
//        xi1 = xi1 + volt1[y];
      volt = (temp>>4)*2.44e-3;  // 2.44mV per bit is the board conversion
//        volt1[y] = 8.5 - (y*.1);
      if(volt>=1.50 && volt<=8.5) {
        xi1 = xi1 + volt;
        eN1++;
        }
      }
    }

  if(eN0)
    xi0 = xi0/eN0;
  if(eN1)
    xi1 = xi1/eN1;

//printf("v0: %e n0: %d    v1: %e n1: %d\n", xi0, eN0, xi1, eN1);
  sigma0 = 0;
  sigma1 = 0;
```

```
//  for(y=0;y<50;y++) {
//    if(volt0[y]>=1.50 && volt0[y]<=8.5) {
//      sigma0 = sigma0 + pow((volt0[y] - xi0),2);
//    }
//    if(volt1[y]>=1.50 && volt1[y]<=8.5) {
//      sigma1 = sigma1 + pow((volt1[y] - xi1),2);
//    }
//  }

  if(eN0) {
    pressure0 = xi0;
//    sigma0 = sqrt(sigma0/(eN0-1));
  }
  else  {
    pressure0 = 0;
//    sigma0 = 0;
  }

  if(eN1) {
    pressure1 = xi1;
//    sigma1 = sqrt(sigma1/(eN1-1));
  }
  else {
    pressure1 = 0;
//    sigma1 = 0;
  }

//&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&
// Diagnostic lines
//  pressure0 = 2.5;
//  pressure1 = 3.5;
//%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

  convert();                        // do the conversion to get decent units

// now we log to file and screen,if recording turned on
  if(RecordVac) {
    if((vacfp = fopen("vacuum.log", "at")) == NULL) {
      printf("cannot open vacuum.log, stopping logging\n");
      RecordVac=0;
      }
    else {
      printf("low: %g\thigh: %g\n", pressure0, pressure1);
      fprintf(vacfp, "%g\t %g\n", pressure0, pressure1);
      fclose(vacfp);
      }
    }

  return;
}

void cVacuum::convert() {

//-------------------------****---------------------------------------
// ==> The vacuum sensor voltage is read. The valid voltage range is:
//
//      1.5V (7.5e-10 Torr or 7.5e-5 Torr)
//
//      to
//
//      8.5V (7.5e-3 Torr or 750 Torr)
//
// ==> The conversion formula defined in BALZERS IKR 250 User’s Guide, pg 22-23
// ==> and the BALZERS TRP 250 Pirani gauge User’s Guide pg 21-22.
//
//-------------------------****---------------------------------------


  if(pressure0)
    pressure0 = pow(10.0, (pressure0-5.625));
  if(sigma0)
    sigma0 = pow(10.0, (sigma0-5.625));

  if(pressure1)
    pressure1 = pow(10.0, (pressure1-10.625));
  if(sigma1)
    sigma1 = pow(10.0, (sigma1-10.625));

}


void cVacuum::VacOn() {

  int port;

  port=gPIO->readPort(’A’);           // read the present port value

  gPIO->writePort(’A’, port&0x3F);    // turn ON all Vacuum Bus Power

  is_vac_on[0] = 1;
  is_vac_on[1] = 1;
  return;
}


void cVacuum::VacOff() {

  int port;

  port=gPIO->readPort(’A’);           // read the present port value

  gPIO->writePort(’A’,port|0xC0);     // turn OFF both vac sensors

  is_vac_on[0] = 0;
  is_vac_on[1] = 0;
  return;
}

void cVacuum::HiVacOn() {

  int port;

  port=gPIO->readPort(’A’);           // read the present port value

  gPIO->writePort(’A’, port&0x3F);    // turn ON Hi Vacuum Bus Power

  is_vac_on[1] = 1;
  return;
}


void cVacuum::LoVacOn() {

  int port;

  port=gPIO->readPort(’A’);           // read the present port value

  gPIO->writePort(’A’, port&0xBF);    // turn ON Vacuum Bus Power (Lo Vac)
```

```
  is_vac_on[0] = 1;
  return;
}


void cVacuum::HiVacOff() {

  int port;

  port=gPIO->readPort('A');           // read the present port value

  gPIO->writePort('A',port|0x80);     // turn OFF Hi vac sensor only

  is_vac_on[1] = 0;
  return;
}
```

```
// CONTENTS : ion.cpp -- This file contains the method definitions
//                           for class cIon.  cIon controls the ion
//                            pump.
// ***-------------------------------***------------------------------***
//    SPECS --
//            Operating System: DOS 6.2
//            Language        : C++ (Microsoft Specific)
//            Compiler        : Microsoft Visual C++ 1.5 (medium Lib)
//
//    HISTORY --
//            Creation Date   : 12 May 98
//            Edit Dates      :
//
//            Author          : Paul Gettings (U of Chicago)
//
// ***------------------------------***------------------------------***

#if defined(_WINDOWS)
#include "stdafx.h"
#endif
#if defined(_DOS)
#include <afx.h>
#endif

#include "cPIO.hpp"
#include "ion.hpp"

extern cPIO*  gPIO;
extern int errCode;

// ***------------------------------***------------------------------***
//
//    The ion pump is controlled via a couple of pins on a back DB9
//  connector.  Pins 3 & 7 must be shorted together to get the HV on
//  the ion pump to turn on.  We do this with PIO bit B0 (pin 10).
//  We can sense the state of the HV via pins 5 & 9 on the DB9.  This
//  is wired to pin 27 of the PIO (C2).
// ***------------------------------***------------------------------***


cIon::cIon() {
  IonOn();  // Turn the pump on, no matter what!
  }

void cIon::IonOn(void) {

  int port;

  port = gPIO->readPort('B');   // read the present state of port

  gPIO->writePort('B',port|0x01); // Set B0 high

  return;
}

void cIon::IonOff(void) {

  int port;

  port = gPIO->readPort('B');   // read the present state of port

  gPIO->writePort('B',port&0xFE); // Set B0 low

  return;
}

int cIon::IonStatus(void)
{
  int port;

  port = gPIO->readPort('C');

  if(port & 0x04)
    return(1);
  else
    return(0);
}
```

```
// CONTENTS : cVacuum.hpp -- This file contains the class definition
//                                for cFilter.

// ***-------------------------------***------------------------------***
//     SPECS --
//             Operating System: DOS 6.2
//             Language        : C++ (Microsoft Specific)
//             Compiler        : Microsoft Visual C++ 1.5 (medium Lib)
//
//     HISTORY --
//             Creation Date   : Sep 13, 1994
//             Edit Dates      : Jan 22, 1995  - added VacOn & VacOff methods
//                             : Apr 11, 1996  - added HiVacOn/Off and LoVacOn()
//                             : Feb 04, 1997  - added sigma variables
//                             : Sep 09, 1997  - added is_vac_on member
//
//             Author          : TjM (University of Chicago, Yerkes Obs)
//
// ***-------------------------------***------------------------------***

#if defined(_WINDOWS)
#include <stdafx.h>
#else
#include <afx.h>
#endif

// These are for logging vac to file
extern FILE *vacfp;
extern int RecordVac;

class cVacuum : public CObject {

  private:

  public:

    cVacuum();
    ~cVacuum();
    void readSensors();
    void convert();
    void status();
    void VacOn();
    void VacOff();
    void HiVacOn();
    void HiVacOff();
    void LoVacOn();

    unsigned dastatus;
    double pressure0, sigma0, pressure1, sigma1;
    int dasBaseAddr;
    int is_vac_on[2];    // informs if systems is active

};
```

```
// CONTENTS : async.hpp -- Header file for the asynchronous Class cSerial

// ***-------------------------------***-----------------------------***
//    SPECS --
//           Operating System: DOS 6.2
//           Language        : C++ (Microsoft Specific)
//           Compiler        : Microsoft Visual C++ 1.5 (medium Lib)
//
//    HISTORY --
//           Creation Date   : June 21, 1994
//           Edit Dates      : July 13, 1994 - added serial interupts vectors
//
//           Author          : TjM (University of Chicago, Yerkes Obs)
//
// ***------------------------------***-----------------------------***

extern pCom;

#define COM1  0         //  define the serial ports
#define COM2  1
#define COM3  2
#define COM4  3

#define PRIME   0x3f8   // Primary I/O port address
#define SECOND  0x2f8   // Secondary I/O port address

#define LPT1  1         // define the parallel
#define LPT2  2
#define LPT3  3

#define RS232_IO  0x14  // software interrupt for ROM-BIOS

#define RS_INIT  0x0000 // service codes for ROM BIOS calls
#define RS_XMT   0x0100
#define RS_RCV   0x0200
#define RS_STAT  0x0300

#define BAUD_96  0xE0   // baud rate definitions
#define BAUD_48  0xC0
#define BAUD_24  0xA0
#define BAUD_12  0x80

#define PAR_NONE 0x00   // parity definitions
#define PAR_ODD  0x08
#define PAR_EVEN 0x24

#define STOP_1   0x00   // define stop bits
#define STOP_2   0x04

#define DATA_7   0x02
#define DATA_8   0x03   // define data size

#define XON   17
#define XOFF  19

#define RCV_DATA    0   // I/O port offsets for async board
#define XMT_DATA    0
#define INT_ENABLE  1
#define INT_ID      2
#define LINE_CTRL   3
#define MODEM_CTRL  4
#define LINE_STAT   5
#define MODEM_STAT  6

// Line Status Registor bits
#define XMT_RDY     0x20    // 1 = ok to transmit
#define BRK_DET     0x10    // 1 = break recv'd
#define FRM_ERR     0x08    // 1 = frameing error
#define PAR_ERR     0x04    // 1 = parity error
#define OVR_RUN     0x02    // 1 = recv'd overrun
#define DTA_RDY     0x01    // 1 = recv'd data ready
#define RCV_ERRS    0x1E    // 1 = recv'd error bits above

#define CAR_DET     0x80    // Modem register bits
#define RNG_IND     0x40
#define DSR         0x20
#define CTS         0x10

// --- Modem control register values ----
#define DTR   1
#define RTS   2
#define OUT2  8

// --- interrupt enable register signals ---
#define   DATAREADY   1
```

```
// CONTENTS : cError.hpp -- This file contains the function and class
//                          prototypes for the echelle motor project

// ***-------------------------------***------------------------------***
//    SPECS --
//            Operating System: DOS 6.2
//            Language        : C++ (Microsoft Specific)
//            Compiler        : Microsoft Visual C++ 1.5 (medium Lib)
//
//    HISTORY --
//            Creation Date   : June 20, 1994
//            Edit Dates      : Apr  29, 1996 - made Windows compatable
//                            :
//
//            Author          : TjM (University of Chicago, Yerkes Obs)
//
// ***------------------------------***------------------------------***

#if defined(_WINDOWS)
#include "stdafx.h"
#else
#include <afx.h>
#endif

// *********************************************
// ******* Prototype for cError Class  *********
// *********************************************

class cError : public CObject {

  public:
    cError();                   // constructor
    void cErrHndl();           // parses the errors

    char temp;

};
```

```
// CONTENTS : cFilterWhl.hpp -- This file contains the method definition
//                               for class cFilter.

// ***-------------------------------***-----------------------------***
//    SPECS --
//            Operating System: DOS 6.2
//            Language        : C++ (Microsoft Specific)
//            Compiler        : Microsoft Visual C++ 1.5 (medium Lib)
//
//    HISTORY --
//            Creation Date   : Aug 19, 1994
//            Edit Dates      : Sept 9, 1994  - added more class definitions:
//                            :               - cSlit-slit focus motor class
//                            :               - cTipTilt-tip/tilt motor class
//                            :               - cCalibration-calibration mirror
//                            :               - cCCDfocus-CCD focus motor class
//                            :               - cGrating-Grating motor class
//                            : Sept 26, 1994 - added cCalibration::calibIn(),
//                                              calibOut(), lampOn, lampOff
//                            :               - added cShutter class definition
//                            : Sept 27       - added cstring name to FilterWhl
//                                              class
//                            : Oct 3, 1994   - added chkLimit() method to
//                                              cFilterWhl class
//                            : Oct 31        - added cMoveMotor overloads to
//                                              CCDFocus, Grating, TipTilp
//                            : Dec 4, 1994   - findHome() method added to
//                                              cCCDfocus and TipTilt classes
//                            : Dec 10, 1994  - FindHome() method added to
//                                              cGrating class
//                            : Jan 4, 1995   - added cFilterWhl::updateCount()
//                            : Jan 19        - eliminated cShutter class
//                            : Jan 28, 1995  - added power method
//                            : Feb 10, 1995  - added hallStat int member var
//                            : Apr 04, 1995  - overloaded cReadStepCount() of
//                                              cCCDfocus class
//                            : Apr 04, 1995  - removed cShutter definitions
//                            : Aug 02, 1995  - added findHome method to cSlit
//                            : Feb 14, 1997  - added timer method to cFilter
//                            : Feb 27, 1997  - cGrating::findHome now rtns int
//                            : Feb 28, 1997  - created new Base motor class
//                                              cBasicMotor from which all other
//                                              motor classes will be daughters
//                            : Aug 19, 1997  - added new method cBasicMotor::FHMoveMo
tor()
//                            : Aug 29, 1997  - overloaded cBasicMotor::findHome in cG
rating
//
//            Author          : TjM (University of Chicago, Yerkes Obs)
//
// ***------------------------------***-----------------------------***

#include "ctimer.hpp"

#define MAX_TRIES   10000               // findHome() timeout - if we haven't hit
                                        // something after 10,000 tries, we have
                                        // a real problem

class cBasicMotor : public CStepperMotor, public cTimer {

  public:
    cBasicMotor();
    ~cBasicMotor();

    BOOL findHome();
    void cReadStepCount();                 // overlaod of CStepperMotor method
    void cMoveMotor(long numofSteps);     // overlaod of CStepperMotor method
    void chkLimits();
    void updateCount();                    // brings dependant objs up-to-date
    void power(int);                       // powers up or down the controller
    void FHMoveMotor(long numofSteps);    // added because blowing fuses
                                           // turning on/off pio power too fast
    BOOL timer(void);

    int m_hall_bit;                        // PIO bit of Hall Effect switch
    int statAux;                           // status of filter wheel switch
    int hallStat;                          // status of Hall Effect switches
    int limitStat[2];                      // shows the status of limit switches
    long position;
    CString name;                          // name of the device

};

class cFilterWhl : public cBasicMotor {

  public:

    int trigaddr;   // detent switch trigger address

    cFilterWhl();
    cFilterWhl(int, int, const char*);
    void findHome();
    void cMoveFilter(long pos);           // moves only filterWhl objects
};



class cCCDfocus : public cBasicMotor {

  public:

    cCCDfocus(int axis, int optib);
    void cMoveMotor(long numofSteps);     // overlaod of cFilterWhl method
    void FHMoveMotor(long numofSteps);    // overload of cBasicMotor
    void cReadStepCount();
    BOOL findHome(void);                   // overload of cBasicMotor::findHome()

};

class cSlit : public cBasicMotor {

  public:

    cSlit(int axis, int optib);
    void cMoveMotor(long numofSteps);     // overlaod of cFilterWhl method
    BOOL findHome(void);                   // overload of cBasicMotor::findHome()

};

class cCalibration : public cBasicMotor {

  public:

    int whichlamp;

    cCalibration(int axis, int optib);
    void calibIn();
    void calibOut();
    void lampOn(int);
    void lampOff();
```

```
    void cMoveMotor(long numofSteps);     // overlaod of cFilterWhl method

};

class cTipTilt : public cBasicMotor {

  public:

    cTipTilt(int axis, int optib, const char *n);
    void cMoveMotor(long numofSteps);     // overlaod of cFilterWhl method
    void FHMoveMotor(long numofSteps);    // overload of cBasicMotor method
};

class cGrating : public cBasicMotor {

  public:

    cGrating(int axis, int optib);
    void cMoveMotor(long numofSteps);     // overlaod of cFilterWhl method
    BOOL findHome(void);                  // overload of cBasicMotor::findHome()

};
```

```
// CONTENTS : cLamp.hpp -- This file contains the method definition
//                         for class cLamp. cLamp class will control
//                         the white light, Fe/Ne and Th/Ar lamps.
// ***------------------------------***------------------------------***
//    SPECS --
//            Operating System: DOS 6.2
//            Language        : C++ (Microsoft Specific)
//            Compiler        : Microsoft Visual C++ 1.5 (medium Lib)
//
//    HISTORY --
//            Creation Date   : Jan 23, 1995
//            Edit Dates      : Apr 29, 1996 - made Windows compatable
//
//            Author          : TjM (University of Chicago, Yerkes Obs)
//
// ***------------------------------***------------------------------***

#if defined(_WINDOWS)
#include "stdafx.h"
#else
#include <afx.h>
#endif

class cLamp : public CObject {

  public:

    CString lampName;     // 'White', 'Fe/Ne' or 'Th/Ar'
    CString lampW, lampT, lampF;    // individual lamp flags

    cLamp();
    void lampOn(char);
    void lampOff(char);

};
```

```
// CONTENTS : cMtrObL.hpp -- A list with the motor object pointers. This will
//                           be used in conjunction with cSystem class object.

// ***-------------------------------***------------------------------***
//    SPECS --
//            Operating System: DOS 6.2
//            Language        : C++ (Microsoft Specific)
//            Compiler        : Microsoft Visual C++ 1.5 (medium Lib)
//
//    HISTORY --
//            Creation Date   : Sept 1, 1994
//            Edit Dates      : Sept 9, 1994 - increased number of motors to 12 (up fr
om 6)
//                            : Apr 29, 1996 - made Windows compatable
//                            : Feb 28, 1997 - compatable with new cBasicMotor class
//
//            Author          : TjM (University of Chicago, Yerkes Obs)
//
// ***-------------------------------***------------------------------***

#if defined(_WINDOWS)
#include "stdafx.h"
#else
#include <afx.h>
#endif

class cMtrObL : public CObject {

  public:

    cBasicMotor* sList[12];

    cMtrObL();
    cBasicMotor* getPtr(int idx);
    void addPtrs();

};
```

```
// CONTENTS : cPIO.hpp -- This file is the include for the class definition
//                         for cPIO class. cPIO controls the power
//                         distribution for the motor enclosures.

// ***-------------------------------***-----------------------------***
//    SPECS --
//            Operating System: DOS 6.2
//            Language        : C++ (Microsoft Specific)
//            Compiler        : Microsoft Visual C++ 1.5 (medium Lib)
//
//    HISTORY --
//            Creation Date   : Jan 16, 1995
//            Edit Dates      : Jan 19, 1995 - added readPort &
//                            :                writePort methods
//                            : Jan 30, 1995 - added 2nd arg. to pwr24V method
//                            : Apr 29, 1996 - made Windows compatable
//
//            Author          : TjM (University of Chicago, Yerkes Obs)
//
// ***------------------------------***-----------------------------***

#if defined(_WINDOWS)
#include "stdafx.h"
#else
#include <afx.h>
#endif

class cPIO : public CObject {

  private:
    int BaseAddr;

  public:
    cPIO();
    ~cPIO();
    void pwr24V(int,int);
    void drivePower(int, int);
    int readPort(char);
    void writePort(char,int);

};
```

```
// CONTENTS : cserial.hpp -- This file contains the function and class
//                           prototypes for the echelle motor project

// ***-------------------------------***-----------------------------***
//    SPECS --
//            Operating System: DOS 6.2
//            Language        : C++ (Microsoft Specific)
//            Compiler        : Microsoft Visual C++ 1.5 (medium Lib)
//
//    HISTORY --
//            Creation Date   : June 20, 1994
//            Edit Dates      : July 13, 1994 - added interrupt vectors
//                            : July 20, 1994 - added cCmdParse class
//                            : Apr 29, 1996  - made Windows compatable
//
//            Author          : TjM (University of Chicago, Yerkes Obs)
//
// ***------------------------------***-----------------------------***

#if defined(_WINDOWS)
#include "stdafx.h"
#else
#include <afx.h>
#endif

// ********************************************
// ****** Prototype for cSerial Class  *********
// ********************************************

class cSerial : public CObject {

  private:

    unsigned pConfig;
    int pCom;         // 0 indicates address 0x3f8, 1 indicates addr 0x2f8
    int iobase;       // base I/O port address of async card

  public:

    cSerial (unsigned addr);
    void initSerial(int baud, int databits, int parity, int stopbits);
    unsigned chkPortStat();
    int readThePort();
    void writeToPort(char *str);
    void signOn();

    char io_str[1024];    // create buffer io_str

};
```

```
// CONTENTS: cSystem.cpp -- Object class that represents the complete system.
//
// ***-------------------------------***-----------------------------***
//    SPECS --
//            Operating System: DOS 6.2
//            Language        : C++ (Microsoft Specific)
//            Compiler        : Microsoft Visual C++ 1.5 (medium Lib)
//
//    HISTORY --
//            Creation Date   : Aug 18, 1994
//            Edit Dates      : Aug 23, 1994 - added systemReset method
//                            : Oct 5, 1994  - added interFaceStat() method
//                            : Apr 28, 1995 - added queryStatus() method
//                            : Apr 29, 1996 - made Windows compatable
//
//            Author          : TjM (University of Chicago, Yerkes Obs)
//
// ***-------------------------------***-----------------------------***

#if defined(_WINDOWS)
#include "stdafx.h"
#else
#include <afx.h>
#endif

class cSystem : public CObject {

  private:

  public:

    cSystem();
    void systemStatus();
    void interFaceStat();
    void systemReset();
    void queryStatus();
};
```

```
// CONTENTS : ctimer.hpp -- This file contains the function and class
//                           prototypes for the echelle motor project

// ***--------------------------------***------------------------------***
//     SPECS --
//             Operating System: DOS 6.2
//             Language        : C++ (Microsoft Specific)
//             Compiler        : Microsoft Visual C++ 1.5 (medium Lib)
//
//     HISTORY --
//             Creation Date   : June 20, 1994
//             Edit Dates      : Sept 9, 1994 - modified so cChkTimer accept float
//                             : Apr 29, 1996 - made Windows compatable
//
//             Author          : TjM (University of Chicago, Yerkes Obs)
//
// ***-------------------------------***------------------------------***

#if defined(_WINDOWS)
#include "stdafx.h"
#endif
#if defined(_DOS)
#include <afx.h>
#endif

// *********************************************
// ******* Prototype for cTimer Class  *********
// *********************************************

class cTimer : public CTimeSpan {

  private:

    time_t tNow;
    time_t tThen;


  public:
    cTimer();               // constructor
    void cStartTimer();     // start the timer
    int cChkTimer(float);     // wait a specified time (integer seconds)
    void waitNSeconds(float); // wait N seconds (float up to 32 seconds)
    time_t elapsed;

};
```

```
// CONTENTS : cError.hpp -- This file contains the function and class
//                           prototypes for the echelle motor project

// ***-------------------------------***------------------------------***
//    SPECS --
//            Operating System: DOS 6.2
//            Language        : C++ (Microsoft Specific)
//            Compiler        : Microsoft Visual C++ 1.5 (medium Lib)
//
//    HISTORY --
//            Creation Date   : June 20, 1994
//            Edit Dates      : Apr  29, 1996 - made Windows compatable
//                            :
//
//            Author          : TjM (University of Chicago, Yerkes Obs)
//
// ***------------------------------***------------------------------***

#if defined(_WINDOWS)
#include "stdafx.h"
#else
#include <afx.h>
#endif

// *********************************************
// ******* Prototype for cError Class  *********
// *********************************************

class cError : public CObject {

  public:
    cError();                   // constructor
    void cErrHndl();           // parses the errors

    char temp;

};
```

```
// CONTENTS : cserial.hpp -- This file contains the function and class
//                          prototypes for the echelle motor project

// ***-------------------------------***-----------------------------***
//    SPECS --
//            Operating System: DOS 6.2
//            Language        : C++ (Microsoft Specific)
//            Compiler        : Microsoft Visual C++ 1.5 (medium Lib)
//
//    HISTORY --
//            Creation Date   : June 20, 1994
//            Edit Dates      : July 13, 1994 - added interrupt vectors
//                            : July 20, 1994 - added cCmdParse class
//                            : Apr 29, 1996  - made Windows compatable
//
//            Author          : TjM (University of Chicago, Yerkes Obs)
//
// ***------------------------------***-----------------------------***

#if defined(_WINDOWS)
#include "stdafx.h"
#else
#include <afx.h>
#endif

// ********************************************
// ****** Prototype for cSerial Class  *********
// ********************************************

class cSerial : public CObject {

  private:

    unsigned pConfig;
    int pCom;         // 0 indicates address 0x3f8, 1 indicates addr 0x2f8
    int iobase;       // base I/O port address of async card

  public:

    cSerial (unsigned addr);
    void initSerial(int baud, int databits, int parity, int stopbits);
    unsigned chkPortStat();
    int readThePort();
    void writeToPort(char *str);
    void signOn();

    char io_str[1024];    // create buffer io_str

};
```

```
// CONTENTS : ctimer.hpp -- This file contains the function and class
//                          prototypes for the echelle motor project

// ***------------------------------***-----------------------------***
//    SPECS --
//            Operating System: DOS 6.2
//            Language        : C++ (Microsoft Specific)
//            Compiler        : Microsoft Visual C++ 1.5 (medium Lib)
//
//    HISTORY --
//            Creation Date   : June 20, 1994
//            Edit Dates      : Sept 9, 1994 - modified so cChkTimer accept float
//                            : Apr 29, 1996 - made Windows compatable
//
//            Author          : TjM (University of Chicago, Yerkes Obs)
//
// ***------------------------------***-----------------------------***

#if defined(_WINDOWS)
#include "stdafx.h"
#endif
#if defined(_DOS)
#include <afx.h>
#endif

// *********************************************
// ******* Prototype for cTimer Class  *********
// *********************************************

class cTimer : public CTimeSpan {

  private:

    time_t tNow;
    time_t tThen;


  public:
    cTimer();               // constructor
    void cStartTimer();     // start the timer
    int cChkTimer(float);     // wait a specified time (integer seconds)
    void waitNSeconds(float); // wait N seconds (float up to 32 seconds)
    time_t elapsed;

};
```

```
 // CONTENTS : Dispatch.hpp -- This file contains the class definition
//                                  cDispatcher.

// ***-------------------------------***------------------------------***
//    SPECS --
//            Operating System: DOS 6.2
//            Language        : C++ (Microsoft Specific)
//            Compiler        : Microsoft Visual C++ 1.5 (medium Lib)
//
//    HISTORY --
//            Creation Date   : July 20, 1994
//            Edit Dates      : July 25, 1994 - major mods including name changes
//                            : Aug 11, 1994  - major mods including name changes
//                            : Sept 21, 1994 - added prs_ac_cmd
//                            : Sept 04, 1997 - changed nsteps[] to 2 dimensional arra
y
//                            : Sept 08, 1997 - added chkrange() method
//
//            Author          : TjM (University of Chicago, Yerkes Obs)
//
// ***------------------------------***------------------------------***


// *****************************************************
// ******* Prototype for cDispatcher Class  *********
// *****************************************************

// ***--------------------------------***-----------------------------------***
// *** This method parses incomming message strings. Passes parsed parameters
// *** on to router() method. Selects the appropriate object to whom command
// *** parameters are sent. All objects talk throughresponder method.
// *** Parses the move command strings grabs the start end and accel
// *** parameters for a ramp change.
// ***--------------------------------***-----------------------------------***

class cDispatcher : public CObject {

  private:

    int stat;
    int ShutDown;                     // shutdown flag for autostart
    int DeGas();                      // Move motors to degas tank
    void waitNSeconds(float secsToWait);            // Wait for N secs, N is real

  public:

    cDispatcher();                    // class constructor
    void parser(const char* );        //
    void router();                    //
    void responder(int respType);     //
    int prs_mv_cmd();                 //
    void prs_ac_cmd();                //
    BOOL chkrange(int, long);         // check the range of the incomming move
                                      //
    CString inStr, outStr;            //
                                      //
    int obj, cmd;
    long target_pos[10];                // target position for the move
    long nsteps[10][2];
    int lastOS;                       // the last optistep card that was used

};
```

```
// CONTENTS : cLamp.hpp -- This file contains the method definition
//                          for class cLamp. cLamp class will control
//                          the white light, Fe/Ne and Th/Ar lamps.
// ***------------------------------***------------------------------***
//    SPECS --
//            Operating System: DOS 6.2
//            Language        : C++ (Microsoft Specific)
//            Compiler        : Microsoft Visual C++ 1.5 (medium Lib)
//
//    HISTORY --
//            Creation Date   : Jan 23, 1995
//            Edit Dates      : Apr 29, 1996 - made Windows compatable
//
//            Author          : TjM (University of Chicago, Yerkes Obs)
//
// ***------------------------------***------------------------------***

#if defined(_WINDOWS)
#include "stdafx.h"
#else
#include <afx.h>
#endif

class cIon : public CObject {

  public:
    cIon();

    void IonOn(void);
    void IonOff(void);
    int IonStatus(void);
};
```

```
/* Borland C++ headers shown in this example:
extern "C" void far isarc(long int *,long int *,long int *,long int *, int *,int *);

Include a "C" in each of the lines below for Borland C++.
*/

extern void far isarc(long int *,long int *,long int *,long int *, int *,int *);
extern void far issetbacklash(int *,int *,int *,int *);
extern void far issetbacklashdir(int *,int *,int *,int *);
extern void far ischeckcard(int *,int *);
extern void far iscontrol(int *,int *);
extern void far isdriverinfo(int *);
extern void far isencodemode(int *, int *, int *);
extern void far isfeedhold(int *);
extern void far isjogger(long int *,long int *, long int *, long int *, int *, int *,
int *);
extern void far isjogkeys(int *,int *,int *, int *, int *, int *, int *, int *);
extern void far isjogtapsteps(int *);
extern void far isjoyport(int *);
extern void far isjoystick(long int *,long int *,long int *, long int *, int *, int *,
 int *);
extern void far issetkeyboardint (int *);
extern void far isloadcount(long int *,long int *,long int *, long int *);
extern void far ismover(long int *,long int *, long int *, long int *, int *);
extern void far ispowerinit(int *);
extern void far isprofile(int, int,long int);
extern void far isrampset();
extern void far isreadcount(long int *,long int *,long int *, long int *);
extern void far issetcplane(int *,int*);
extern void far issetcardtype(int *);
extern void far isjogtapsteps (int *);
extern void far issetmode(int *);
extern void far issetmultiplex(int *);
extern void far issetspeed(int *);
extern void far isstatus(int *, int *);
extern void far isthreader(long int *,long int *,long int *, long int *, long int *, i
nt *, int *);
```

```
// CONTENTS : Motor.hpp -- This file contains the global definitions for the
//                          echelle motor controllers.
//
// ***------------------------------***-----------------------------***
//     SPECS --
//             Operating System: DOS 6.2
//             Language        : C++ (Microsoft Specific)
//             Compiler        : Microsoft Visual C++ 1.5 (medium Lib)
//
//     HISTORY --
//             Creation Date   : Aug 11, 1994
//             Edit Dates      :
//
//             Author          : TjM (University of Chicago, Yerkes Obs)
//
// ***------------------------------***-----------------------------***

#include "classes.hpp"
#include "proto.hpp"
#include "cserial.hpp"
#include "cerror.hpp"
#include "opti1.hpp"
#include "cFilter.hpp"
#include "cMtrObL.hpp"
//#include "ctimer.hpp"
#include "optiparm.hpp"     // parameter definitions
#include "async.hpp"
#include "dispatch.hpp"
#include "cSystem.hpp"
#include "Vacuum.hpp"
#include "objsave.hpp"
#include "cPIO.hpp"
#include "cLamp.hpp"
#include "ion.hpp"
extern "C" {
#include "serial.h"
#include "timer.h"
}

#define TRUE 1
#define FALSE 0

extern int stopFlag;      // used for indicating a exit from the program is desired
extern int errCode;
extern int rtnCode;       // global var used to store the return code from object meth
ods

extern buffer_count;      // serial buffer count (how amny characters are in the buffe
r)
```

```
// CONTENTS : objSave.hpp --  This file contains the method definitions
//                            for cObjSave, a class used for file I/O.
//
// ***-------------------------------***-----------------------------***
//    SPECS --
//            Operating System: DOS 6.2
//            Language        : C++ (Microsoft Specific)
//            Compiler        : Microsoft Visual C++ 1.5 (medium Lib)
//
//    HISTORY --
//            Creation Date   : Oct 18, 1994
//            Edit Dates      : Apr 29, 1996 - made Windows compatable
//
//            Author          : TjM (University of Chicago, Yerkes Obs)
//
// ***------------------------------***-----------------------------***

#if defined(_WINDOWS)
#include "stdafx.h"
#else
#include <afx.h>
#endif

class cObjSave : public CObject {

  private:

  CString fileName;

  public:

    cObjSave();
    BOOL write();
    BOOL read();

};
```

```
// CONTENTS : cOptiS.hpp -- This file contains the function and class
//                          prototypes for the echelle motor project

// ***------------------------------***-----------------------------***
//    SPECS --
//            Operating System: DOS 6.2
//            Language        : C++ (Microsoft Specific)
//            Compiler        : Microsoft Visual C++ 1.5 (medium Lib)
//
//    HISTORY --
//            Creation Date   : Aug 11, 1994
//            Edit Dates      : Aug 29, 1994 - changed constructor argument to
//                            :                local var
//                            : Sept 5, 1994 - removed redundant cReadStepCount
//                            :                from CStepperMotor class
//                            : Sept 6, 1994 - moved cSetMovMode from
//                            :                CStepperMotor to COptiStep class
//                            : Sept 26      - CStepperMotor::cMoveMotor now
//                            :                takes long int
//                            : Jan 28, 1995 - added "box" to CStepperMotor
//
//            Author          : TjM (University of Chicago, Yerkes Obs)
//
// ***------------------------------***-----------------------------***

#define TRUE 1
#define FALSE 0
#include "cIOCard.hpp"
#include <time.h>

// *********************************************
// ***** Prototype for COptiStep Class  ********
// *********************************************

class COptiStep : public cIOCard {

  private:
    int whichPort,      // Board addressing parameters
        ospStat;

    void cSetCardType (int cardType);
    void cPortControl (int whichPort, int state);

  public:
    COptiStep();                  // default constructor
    COptiStep(int baseAddress);   // specific addressing constructor
    int cLimitStatus();
    void cCardInit ();            // need to call this each time a different -
    void cReadStepCount();        // OptiStep Board is used
    void cSetStepCount(int stpMtr, long newCount);
    void cResetCounters();        // sets board counters to correct value
    void cSetMovMode (int);

    double oscFreq;
    long  cnt1, cnt2, cnt3, cnt4; // the step count from all 4 motors
    int boardOK;                  // 1 = OK, 0 = Problems



};

// ***************************************************************
// ***** Prototype for CStepperMotor Class **********************
// ***************************************************************

class CStepperMotor : public COptiStep {

  private:

    int clkDelay;     // the delay between sync pulses out to motor

  public:

    int axisID,       // the number of the motor (1 to 3)
        whichOpti,    // which optistep card controls motor (1-4)
        powrSetting,  // 0%, 20%, 60% or 100%
        box;          // which enclosure controls motor (1-2)

    int movHow,       // make relative or absolute move
        stpsPerSec,   // max speed of the motor
        accel,        // acceleration of the ramp
        start,        // starting velocity (counts/sec)
        end;          // velocity at end of ramp (must be less than stpPerSec)

    long stepCount;

    CStepperMotor();                                // constructor
    CStepperMotor(int axis, int optib);             // constructor
    void cMoveMotor (long);
    void cRampProfile (int start, int end, int accel);
    void cSetSpeed (int speed);
    void cSetToRamp ();
    void cPowerCtrl (int powrSetting);

};


// ================================================================
```

```
/* optiparm.hpp --- Include file for the OptiStep Class   */

#define OptiSP 3    /* 3 is the OptiStep Plus card type  */

#define  pwr100  15   /* pwr0=0%, pwr20=20%, pwr60=60%, pwr100=100% */
#define  pwr60   5
#define  pwr20   10
#define  pwr0  0

#define errNone 0
#define errOpt1 -1
#define errOpt2 -2
#define errOpt3 -3
#define errOpt4 -4

#define errPort -5
#define errCount -6

/* Limit switch error definitions */

#define A1_BOT 1
#define A1_TOP 2
#define A2_BOT 3
#define A2_TOP 4
#define A3_BOT 5
#define A3_TOP 6

/* Motor Encoder error defintions */

#define errEncode1 32
#define errEncode2 33
#define errEncode3 34
#define errEncode4 35

/* Enclosure Error Codes */

#define errPwr1 51   // power (24V) box 1
#define errPwr2 52   // power (24V) box 2
#define errVacP 53   // Vacuum Sensor power
#define errVacL 54   // Vacuum Sensor Lo Limit
#define errVacH 55   // Vacuum Sensor Over Pressure
#define errLamp 56   // Lamp source power


/* Misc. motor error code definitions */

#define errShld 51
#define errESCkey 129
#define errPause 189

/* Syntax Errors */

#define errSyntax 100

#define  MovRel  0
#define  MovAbs  1

// --- command definitions -------------------------------

#define  KIL    999         // Total shutdown
#define  MVR    0           // move relative
#define  CI     1           // cal mirror in
#define  CO     2           // cal mirror out
#define  SC     3           // shutter close
#define  SO     4           // shutter open
#define  ST     6           // system status
#define  AC     7           // set acceleration
#define  IS     8           // interface status
#define  LA     9           // lamp on
#define  LO     10          // lamp off
#define  RS     11          // system reset
#define  ZC     12          // Zero Motor Count
#define  FH     13          // find home command
#define  VA     14          // Vacuum System On
#define  VO     15          // Vacuum system Off
#define  LX     16          // exclusive lamp on
#define  PA     17          // power activate
#define  PO     18          // power off
#define  IA     19          // Ion Pump Activate
#define  IO     20          // Ion Pump Off
#define  CC     199         // Console-only command - DeGas, Write Vacuum, etc.
#define  DG     200         // DeGas console-only command
#define  WV     201         // Write Vacuum console-only command
#define  WO     202         // Write Off (vacuum) console-only command
```

```
/* optiStep.hpp --- Include file for the OptiStep Class   */

#define OptiSP 3    /* 3 is the OptiStep Plus card type  */

#define  pwr100  15   /* pwr0=0%, pwr20=20%, pwr60=60%, pwr100=100% */
#define  pwr60   5
#define  pwr20   10
#define  pwr0  0

#define errNone 0
#define errOpti1  -1
#define errOpti2  -2
#define errOpti3  -3
#define errOpti4  -4
//#define errPort   -6
//#define errCount  -7

/* Limit switch error definitions */

#define errLSw1 1
#define errLSw2 2
#define errLSw3 3
#define errLSw4 4
#define errLSw5 5
#define errLSw6 6
#define errLSw7 7
#define errLSw8 8

/* Motor Encoder error defintions */

#define errEncode1 32
#define errEncode2 33
#define errEncode3 34
#define errEncode4 35

/* Misc. motor error code definitions */

#define errShld 51
#define errESCkey 129
#define errPause 189

#define  MovRel  0
#define  MovAbs  1
```

```
// CONTENTS : pointers.hpp -- A list with the object pointers.

// ***--------------------------------***------------------------------***
//    SPECS --
//            Operating System: DOS 6.2
//            Language        : C++ (Microsoft Specific)
//            Compiler        : Microsoft Visual C++ 1.5 (medium Lib)
//
//    HISTORY --
//            Creation Date   : Sept 1, 1994
//            Edit Dates      : Sept 9, 1994 - added new class pointers for
//                            :              - the motors and OptiStep boards
//                            : Apr 28, 1995 - eliminated cShutter pointer
//
//            Author          : TjM (University of Chicago, Yerkes Obs)
//
// ***-------------------------------***------------------------------***

extern COptiStep    *gOptiOne;       // pointer to OptiStep board #1
extern COptiStep    *gOptiTwo;       // pointer to OptiStep board #2
extern COptiStep    *gOptiThree;     // pointer to OptiStep board #3
extern COptiStep    *gOptiFour;      // pointer to OptiStep board #4

extern cFilterWhl   *gEFilter;       // echelle filter wheel object pointer
extern cFilterWhl   *gCFilter;       // guide camera filter wheel pointer
extern cSlit        *gSlit;          // slit focus pointer
extern cTipTilt     *gTip;           // CCD tip motor pointer
extern cTipTilt     *gTilt;          // CCD tilt motor pointer
extern cGrating     *gGrating;       // grating tilt motor pointer
extern cCCDfocus    *gCCD;           // CCD focus motors pointer
extern cCalibration *gCalMirror;     // calibration folding mirror motor pointer
// extern cShutter      *gShutter;       // shutter object pointer

extern cMtrObL      *gList;          // object list object pointer
extern cObjSave     *gSave;          // object pointer for saving motor states
extern cDispatcher  *gZephyr;        // pointer to cDispatcher object
```

```
// CONTENTS : proto.hpp -- This file contains the function and class
//                         prototypes for the echelle motor project

// ***-------------------------------***------------------------------***
//    SPECS --
//            Operating System: DOS 6.2
//            Language        : C++ (Microsoft Specific)
//            Compiler        : Microsoft Visual C++ 1.5 (medium Lib)
//
//    HISTORY --
//            Creation Date   : June 20, 1994
//            Edit Dates      : Nov 8, 1994   - added vois Ctrl_k()
//                            : May 3, 1995   - added menu() function
//                            : Apr 29, 1996  - made Windows compatable
//                            : Aug 29, 1996  - added signonBanner() prototype
//
//            Author          : TjM (University of Chicago, Yerkes Obs)
//
// ***-------------------------------***------------------------------***

#if defined(_WINDOWS)
#include "stdafx.h"
#else
#include <afx.h>
#endif

// ********************************************
// ***** Prototypes for keyboard.cpp files *****
// ********************************************

int keyhit();
int getkey ();
void keyflush();
void waitcr();
int chkforESC();
int Ctrl_k();
int menu();
void signonBanner();
```

```
// CONTENTS : serial.h -- Header file for the asynchronous Serial functions

// ***-------------------------------***-----------------------------***
//    SPECS --
//            Operating System: DOS 6.2
//            Language        : C (Microsoft specific)
//            Compiler        : Microsoft Visual C++ 1.5 (medium Lib)
//
//    HISTORY --
//            Creation Date   : July 15, 1994
//            Edit Dates      :
//
//            Author          : Microsoft QuickC Programming (M. Waite, Microsoft Pres
s 1988)
//
// ***------------------------------***-----------------------------***/

extern SERIALPORT;
extern char *nextserialin, *nextserialout;

/* --- serial prototypes --- */
void  initserialvect();
int readSerial();
int writeserial(int);
void clear_serial_Q();
void restore_serialint();
int howManyChars();
int chkForRTN();

/* --- serial macros --- */
#define serialstat() (inp(LINESTATUS))
#define input_char_ready() (nextserialin!=nextserialout)
#define XON   17
#define XOFF  19

/* -------------- serial port addresses ------------------ */
/* - 8250 UART base port address: COM1 = 3f8, COM2 = 2f8 - */
#define BASE        (0x3f8-((SERIALPORT-1)<<8))
#define TXDATA       BASE
#define RXDATA       BASE
#define DIVLSB       BASE
#define DIVMSB      (BASE+1)
#define INTENABLE   (BASE+1)
#define INTIDENT    (BASE+2)
#define LINECTL     (BASE+3)
#define MODEMCTL    (BASE+4)
#define LINESTATUS  (BASE+5)
#define MODEMSTATUS (BASE+6)

/* --- serial interrupt values --- */
#define IRQ           (4-(SERIALPORT-1))      /* 0-7 = IRQ0 - IRQ7 */
#define SERIALINT     (12-(SERIALPORT-1))     /* serial interrupt vector */
#define SERIALIRQ     (~(1 << IRQ))
#define PIC01         0x21              /* 8259 Programmable Interrupt Controller */
#define PIC00         0x20              /* Same */
#define EOI           0x20              /* End of Interrupt Command */
#define TIMER         0x1C              /* PC timer interrupt vector */


/* --- line status register values --- */
#define XMIT_DATA_READY   0x20

/* --- Modem control register values ---- */
#define DTR   1
#define RTS   2
#define OUT2  8

/* --- ASCII serial control characters ---- */
#define   SOH   1
#define   EOT   4
#define   ACK   6
#define   NAK   0x15
#define   CAN   0x18

/* --- interrupt enable register signals --- */
#define   DATAREADY   1

/* --- serial input interrupt buffer --- */
#define SERIALBUFSIZE   1024
#define SAFETYLVL       (SERIALBUFSIZE/4)
#define THRESHLD        (SAFETYLVL*3)
```

```
// Timer.h -- Header file that supports timer functions

extern unsigned ticker;

// --- timer prototypes ---
void intercept_timer(void (*) ());
void restore_timer();

// --- timer macros ---
#define timed_out() (ticker==0)
#define set_timer(secs) ticker=(secs)*182/10+1
```