Mar 02, 02 17:06	**AUTHORS**	Page 1/1

```
Flancrest Enterprises, Group 22
Chris Frost, Emilio Lahr-Vivaz, Scott Rein, and Thomas Whanger
```

```
TODO: Write this file.
FANG is the FlANcrest client Gui, which interfaces with
Decepticon Industries' robot server, which together allow robot control.
```

May 01, 02 8:45 **TODO** Page 1/1

```
== Maintenance ==
- *** If you add or remove a file from cvs, update the Makefile ***
- Create Makefile for cvs checkin/checkouts
   - Auto create ChangeLog
   - More?
- Write the README

== FANG General ==
= Robot
- Change threads that should exit upon disconnect() from CSingleLock
to CMultiLock and change disconnect() to only send a signal to some
disconnect event. Iff multiple people can receive events.

= Errors
- Change Errors.[h|cpp] to Utilities.[h|cpp]
- Make Errors::fangDebug() and Errors::fangError() log to a
vector<vector<string>> so that we can review them later, and so that the
debug console can be opened/closed w/o loss of data

= GUI
- Let user know if the robot is busy (robot, camera, [regular processing?])
- Check for robot == NULL
- Add const private data members for command strings
- Maybe add a function to make building commands easier
- Look into using a joystick

= Joystick
- move from using arrays to vectors

= Known System Bugs
- Sending a (setHeartBeat x) very soon after (init ...) casues problems
for the robot server
- Robot::disconnect() sometimes locks up
- The robot server may take a few seconds to get going?
```

```
lab8: code used in lab 8
lab9b: code used in lab 9
inspect3: phase three of inspection
lab12: code used in lab 12
games: code used in robot games
```

May 01, 02 8:46 **Makefile** Page 1/1

```
# $Id: Makefile,v 1.9 2002/05/01 12:46:18 ccf7f Exp $

A2PS=a2ps
PS2PDF=ps2pdf
SOURCETITLE="FANG Source, Flancrest Enterprises"
TABSIZE=3
CHARSPERLINE=85 # Take care of those occasional slightly long line
SOURCEFILES=AUTHORS COPYING README TODO TAGS Makefile \
fang/fang.h fang/fang.cpp \
fang/cmd.h fang/cmd.cpp \
fang/errors.h fang/errors.cpp \
fang/robot.h fang/robot.cpp \
fang/ws-util.h fang/ws-util.cpp \
fang/ConfigDlg.h fang/ConfigDlg.cpp \
fang/fangDlg.h fang/fangDlg.cpp \
fang/joystick.h fang/joystick.cpp \
fang/Resource.h fang/StdAfx.h fang/StdAfx.cpp fang/fang.rc fang/res/fang.rc2
A2PSOPTIONS=--header=$(SOURCETITLE) -r --tabsize=$(TABSIZE) \
--chars-per-line=$(CHARSPERLINE) --toc

fang.ps::
   $(A2PS) $(A2PSOPTIONS) \
   -o fang.ps \
   $(SOURCEFILES)

fang.pdf::
   $(A2PS) $(A2PSOPTIONS) \
   -o - \
   $(SOURCEFILES) | $(PS2PDF) - fang.pdf


clean::
   rm -f fang.ps fang.pdf
```

```
// $Id: fang.h,v 1.5 2002/04/27 16:29:01 ccf7f Exp $
// Flancrest Enterprises, Group 22
// About this file:
// main header file for the FANG application

#if !defined(AFX_FANG_H__7E6CD9AE_78D4_4402_A9D8_F9F6BEA79534__INCLUDED_)
#define AFX_FANG_H__7E6CD9AE_78D4_4402_A9D8_F9F6BEA79534__INCLUDED_

#if _MSC_VER > 1000
#pragma once
#endif                          // _MSC_VER > 1000

#ifndef __AFXWIN_H__
#error include 'stdafx.h' before including this file for PCH
#endif

#include "resource.h"           // main symbols

/////////////////////////////////////////////////////////////////////////////
// CFangApp:
// See fang.cpp for the implementation of this class
//

class Robot;

class CFangApp : public CWinApp
{
   public:
      CFangApp();

      // Overrides
      // ClassWizard generated virtual function overrides
      //{{AFX_VIRTUAL(CFangApp)
   public:
      virtual BOOL InitInstance();
      //}}AFX_VIRTUAL

      // Implementation

      //{{AFX_MSG(CFangApp)
      // NOTE - the ClassWizard will add and remove member functions here.
      //    DO NOT EDIT what you see in these blocks of generated code !
      //}}AFX_MSG
      DECLARE_MESSAGE_MAP()
};

/////////////////////////////////////////////////////////////////////////////

//{{AFX_INSERT_LOCATION}}
// Microsoft Visual C++ will insert additional declarations immediately before the pr
evious line.
#endif                          // !defined(AFX_FANG_H__7E6CD9AE_78D4_4402_A9D8_F9F6
BEA79534__INCLUDED_)
```

```
// $Id: fang.cpp,v 1.11 2002/04/27 16:29:01 ccf7f Exp $
// Flancrest Enterprises, Group 22
// About this file:
// Defines the class behaviors for the application.

#include "stdafx.h"
#include "fang.h"
#include "fangDlg.h"
#include "errors.h"
#include "robot.h"

#ifdef _DEBUG
#define new DEBUG_NEW
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
#endif

/////////////////////////////////////////////////////////////////////////////
// CFangApp

BEGIN_MESSAGE_MAP(CFangApp, CWinApp)
//{{AFX_MSG_MAP(CFangApp)
// NOTE - the ClassWizard will add and remove mapping macros here.
//    DO NOT EDIT what you see in these blocks of generated code!
//}}AFX_MSG
ON_COMMAND(ID_HELP, CWinApp::OnHelp)
END_MESSAGE_MAP()

/////////////////////////////////////////////////////////////////////////////
// CFangApp construction

CFangApp::CFangApp()
{
   // TODO: add construction code here,
   // Place all significant initialization in InitInstance
}


/////////////////////////////////////////////////////////////////////////////
// The one and only CFangApp object

const bool consoleWindow = true;
CFangApp theApp;

/////////////////////////////////////////////////////////////////////////////
// CFangApp initialization

BOOL CFangApp::InitInstance()
{
   const string fnName = "InitInstance";

   if (!AfxSocketInit())
   {
      AfxMessageBox(IDP_SOCKETS_INIT_FAILED);
      return FALSE;
   }

   // Standard initialization
   // If you are not using these features and wish to reduce the size
   //  of your final executable, you should remove from the following
   //  the specific initialization routines you do not need.

   #ifdef _AFXDLL
   Enable3dControls();           // Call this when using MFC in a shared DLL
   #else
   Enable3dControlsStatic();     // Call this when linking to MFC statically
   #endif

   Errors errors(consoleWindow);

   Robot robot;
   CFangDlg dlg;
   robot.setGui(&dlg);
```



```
   dlg.setRobot(&robot);

   m_pMainWnd = &dlg;
   int nResponse = dlg.DoModal();
   if (nResponse == IDOK)
   {
      // TODO: Place code here to handle when the dialog is
      //  dismissed with OK
   }
   else if (nResponse == IDCANCEL)
   {
      // TODO: Place code here to handle when the dialog is
      //  dismissed with Cancel
   }

   // Since the dialog has been closed, return FALSE so that we exit the
   //  application, rather than start the application's message pump.
   return FALSE;
}
```

```
// $Id: cmd.h,v 1.6 2002/04/21 19:33:29 trw7q Exp $
// Flancrest Enterprises, Group 22
// About this file:
// The Cmd class allows code to talk about information that will be
// sent to/received from the robot server without knowledge of the
// actual protocol and data representation being necessary.

#ifndef CMD_H
#define CMD_H

#include <string>
#include <vector>
using namespace std;

class Cmd
{
   // Associations
   // Attributes
   private:
      string command;
      vector<string> arguments;
      // Operations
   public:
      Cmd();
      // commandName is the name of the command, args is a vector of the
      // arguments
      Cmd(string commandName, vector<string> args);
      // return the command
      string getCommand() const;
      // return the arguments
      vector<string> getArgs() const;
};
#endif
```

```cpp
// $Id: cmd.cpp,v 1.5 2002/04/21 19:33:29 trw7q Exp $
// Flancrest Enterprises, Group 22

#include "stdafx.h"
#include "cmd.h"

//
// public
//

Cmd::Cmd()
{
}


Cmd::Cmd (string commandName, vector<string> args)
{
   command = commandName;
   arguments = args;
}


string Cmd::getCommand() const
{
   return command;
}


vector<string> Cmd::getArgs() const
{
   return arguments;
}
```

```
// $Id: errors.h,v 1.8 2002/04/21 19:33:29 trw7q Exp $
// Flancrest Enterprises, Group 22
// About this file:
// The Error class hides the method of informing the user of errors
// and debugging information.

// Use of the global functions, fangError and fangDebug, is prefered
// to using the Errors class instance member functions, but identical.

#ifndef ERRORS_H
#define ERRORS_H

#include <string>
using namespace std;

string int2string(const int input);

class Errors
{
   public:
      // Set consoleWindow to true if we should open a console window
      Errors(const bool consoleWindow);
      ~Errors();

      // See the argument decriptions below
      void ShowConsole(const bool show);
      void Error(const string Class, const string fn, const string errorMsg);
      void Debug(const string Class, const string fn, const string errorMsg);
   private:
      HANDLE printMutex;
      bool showConsole;
};

// Show (show==true) or don't show (show==false) console
void fangShowConsole(const bool show);

// fangError() and fangDebug() arguments:
// class is the name of the class, fn is the name of the function, and
// (error|debug)Msg is the actual message.
//
// User will be notified via gui
void fangError(const string Class, const string fn, const string errorMsg);

// Messages will be shown to the user only if in debugging mode
void fangDebug(const string Class, const string fn, const string debugMsg);
#endif
```

```cpp
// $Id: errors.cpp,v 1.11 2002/04/27 16:29:01 ccf7f Exp $
// Flancrest Enterprises, Group 22

#include "stdafx.h"

#include "errors.h"
#include "iostream"
using namespace std;

Errors *errors;

string int2string(const int input)
{
   char intAsString[20];
   _itoa(input, intAsString, 10);
   return string(intAsString);
}


Errors::Errors(const bool consoleWindow) : showConsole(false)
{
   const string fnName = "Errors";
   errors = NULL;

   printMutex = CreateMutex(NULL, FALSE,  "ErrorsPrintMutex");
   if(NULL == printMutex)
   {
      cerr << "Unable to create mutex printMutex" << endl;
      cerr << "GetLastError(): " << GetLastError() << endl;
      exit(-1);
   }

   ShowConsole(consoleWindow);
}


Errors::~Errors()
{
   ShowConsole(false);
}


void Errors::ShowConsole(const bool show)
{
   const string fnName = "ShowConsole";

   WaitForSingleObject(printMutex, INFINITE);
   if(show && !showConsole)
   {
      if(AllocConsole())
      {
         freopen("CONIN$", "rb", stdin);
         freopen("CONOUT$","wb", stdout);
         freopen("CONOUT$","wb", stderr);
         cout << "FANG - FlANcrest client Gui\nDebugging Console"
            << endl << endl;
         errors = this;
      }
      else
         Error("Errors", fnName,
            "Unable to open a console window, continuing without a console.");
   }
   else if(!show && showConsole)
   {
      FreeConsole();
   }
   showConsole = show;
   if(!ReleaseMutex(printMutex))
   {
      cerr << "ERROR| Errors::ShowConsole(): Couldn't release printMutex" << endl;
   }
}
```



```cpp
void Errors::Error(const string Class, const string fn, const string errorMsg)
{
   // TODO: this should popup a window for the user instead of displaying
   // to the console.
   // Maybe *also* log to the console, but at least tell the user graphically.
   WaitForSingleObject(printMutex, INFINITE);
   cout << "ERROR| " << Class << "::" << fn << "(): " << errorMsg << endl;
   if(!ReleaseMutex(printMutex))
   {
      cerr << "ERROR| Errors::Error(): Couldn't release printMutex" << endl;
   }
}


void Errors::Debug(const string Class, const string fn, const string debugMsg)
{
   WaitForSingleObject(printMutex, INFINITE);
   cout << "DEBUG| " << Class << "::" << fn << "(): " << debugMsg << endl;
   if(!ReleaseMutex(printMutex))
   {
      cerr << "ERROR| Errors::Debug(): Couldn't release printMutex" << endl;
   }
}


void fangShowConsole(const bool show)
{
   if(NULL != errors)
      errors->ShowConsole(show);
}


void fangError(const string Class, const string fn, const string errorMsg)
{
   if(NULL != errors)
      errors->Error(Class, fn, errorMsg);
}


void fangDebug(const string Class, const string fn, const string debugMsg)
{
   if(NULL != errors)
      errors->Debug(Class, fn, debugMsg);
}
```

```
// $Id: robot.h,v 1.17 2002/04/27 16:29:01 ccf7f Exp $
// Flancrest Enterprises, Group 22
// About this file:
// This class hides design decisions involving how information is sent to
// and received from the robot server (which hides how the actual hardware
// interfaces). In fact, this class completely abstracts the functional
// aspects of there existing a robot server.

#ifndef ROBOT_H
#define ROBOT_H

#include <string>
#include <afxmt.h>
#include "cmd.h"
#include "ws-util.h"
using namespace std;

class CFangDlg;

typedef DWORD timeSpan;           // milliseconds
typedef DWORD currentTime;        // milliseconds

class Robot
{
   // Associations
   private:
      CFangDlg *gui;

      // Attributes
   private:
      string hostname;
      WSAData wsaData;
      SOCKET sd;
      const int defaultPort;

     const double interalHBRate2Protocol;

      // Use data member access functions, touching these directly
      // is not thread safe.
      bool connected;
      currentTime lastHeartBeat;
      timeSpan heartBeatRate;
      bool haveHeartBeat;
      bool debugPrintsCmds;

      // Accessed only through respective mutexes
      // (used for inter-thread communication)
      string cmdToSend;
      string cmdRecvd;

      CEvent sendEvent, sendEventDoneEvent, recvEvent, recvEventDoneEvent,
         netRecvEventDoneEvent,
         heartBeatCheckEvent, heartBeatCheckEventDoneEvent,
         sendHeartBeatEvent, sendHeartBeatEventDoneEvent,
         ifaceEvent, ifaceEventDoneEvent, recvdDisconnectAckEvent,
         heartBeatCheckTimeSetEvent, sendHeartBeatTimeSetEvent;
      CMutex cmdToSendMutex, cmdRecvdMutex, lastHeartBeatMutex,
         heartBeatRateMutex, connectedMutex, haveHeartBeatMutex, guiMutex,
         connectDisconnectMutex, debugPrintsCmdsMutex;

      const string className;
      const string cmdBeginSymbol;
      const string cmdDelimSymbol;
      const string cmdEndSymbol;

      // Operations
   public:
      // Sets the private date member *gui to the argument. Returns true
      // if gui was successfully stored.
      bool setGui(CFangDlg *gui);
      Robot();
      ~Robot();
      // Connects to the given robot server host. Returns true if it was able
```



```
      // to connect.
      bool connect(const string hostname);
      // Disconnects from the robot server, returns true if it was able
      // to disconnect
      bool disconnect();
      // Tells the robot to do command, returns true if it was able to send
      // the message to the robot.
      bool tell(const Cmd command);
      // Log (log) or don't log (!log) robot communication i/o through
      // fangDebug()
      void logCommIO(const bool log);

      // These allow threads in member functions of this class to be started.
      // param is a pointer to this class instance's memory address.
      // Returns 0 on success.
      friend UINT startNetSend(LPVOID param);
      friend UINT startNetRecv(LPVOID param);
      friend UINT startIface(LPVOID param);
      friend UINT startHeartBeatCheck(LPVOID param);
      friend UINT startSendHeartBeat(LPVOID param);
   private:
      //
      // These are the threads of Robot.
      //
      // Sends data to the network
      UINT netSend();
      // Receives data from the network
      UINT netRecv();
      // Processes data received from the network
      UINT iface();
      // Checks that we are receiving a heartBeat from the robot server
      UINT heartBeatCheck();
      // Sends a heartBeat to the robot server
      UINT sendHeartBeat();

      //
      // Access functions for non-thread safe private data members
      //
      // get the heartBeatRate
      timeSpan getHeartBeatRate();
      // sets the heartbeat rate to newHeartBeatRate
      void  setHeartBeatRate(const timeSpan newHeartBeatRate);

      // gets the time of the last heartBeat received
      currentTime getLastHeartBeat();
      // sets the time of the last heartBeat received
      void  setLastHeartBeat(const currentTime newLastHeartBeat);

      // returns true if we have a heartBeat from the robot server
      bool getHaveHeartBeat();
      // sets whether or not we have a heartBeat from the server
      void setHaveHeartBeat(const bool haveIt);

      // returns true if we are currently connected to the robot server
      bool getConnected();
      // sets whether or not we are currently connected to the robot server
      void setConnected(const bool newConnected);

      // returns true if we are logging comm i/o to fangDebug()
      bool getDebugPrintsCmds();
      // sets whether or not we log comm i/o to fangDebug()
      void setDebugPrintsCmds(const bool show);

      // Parses command and turns it into a Cmd. Returns Cmd("", emptyVector)
      // if the command is invalid.
      Cmd string2Cmd(const string command) const;
      // Turns command into the format used to send the command over the network
      string Cmd2String(const Cmd command) const;

      // Passes cmd onto the gui
      void guiUpdate(const Cmd cmd);

      // Initializes variables
```

```
        void initVariables();
        // Sends the "init" command to the robot server
        void sendInitCmd();
        // Exits the thread if not connected. Can only be used in the netRecv() thread
        void checkConnected(const string &fnName);

        // if pcHost is a hostname, returns the ip address. if pcHost is an
        // ip address, returns pcHost
        u_long lookupAddress(const string pcHost) const;
        // brings up the connection to the remoteAddr (an ip addr) on the port
        // port. Returns the socket.
        SOCKET establishConnection(const u_long remoteAddr,const u_short port) const;
};
#endif
```

```
// $Id: robot.cpp,v 1.27 2002/04/30 10:14:21 ccf7f Exp $
// Flancrest Enterprises, Group 22

#include "stdafx.h"

#include <Windows.h>
#include <Mmsystem.h>

#include <winsock.h>
#include "fangDlg.h"
#include "errors.h"
#include "robot.h"

const bool showThreadEntrancesExits = false;
//
// public
//

Robot::Robot()
: defaultPort (6942),
className     ("Robot"),
cmdBeginSymbol("("),
cmdDelimSymbol(" "),
cmdEndSymbol  (")"),
haveHeartBeat (false),
heartBeatRate (0),
interalHBRate2Protocol(10),
lastHeartBeatMutex(FALSE, "lastHeartBeatMutex"),
heartBeatRateMutex(FALSE, "heartBeatRateMutex"),
haveHeartBeatMutex(FALSE, "haveHeartBeatMutex"),
connectedMutex(FALSE, "connectedMutex"),
guiMutex      (FALSE, "guiMutex"),
connectDisconnectMutex(FALSE, "connectDisconnectMutex"),
debugPrintsCmdsMutex(FALSE, "debugPrintsCmdsMutex"),
sendEvent     (FALSE, FALSE, "sendEvent"),
sendEventDoneEvent(FALSE, FALSE, "sendEventDoneEvent"),
recvEvent     (FALSE, FALSE, "recvEvent"),
recvEventDoneEvent(FALSE, FALSE, "recvEventDoneEvent"),
netRecvEventDoneEvent(FALSE, FALSE, "netRecvEventDoneEvent"),
heartBeatCheckEvent(FALSE,FALSE, "heartBeatCheckEvent"),
heartBeatCheckEventDoneEvent(FALSE, FALSE, "heartBeatCheckEventDoneEvent"),
sendHeartBeatEvent(FALSE, FALSE, "sendHeartBeatEvent"),
sendHeartBeatEventDoneEvent(FALSE, FALSE, "sendHeartBeatEventDoneEvent"),
ifaceEvent    (FALSE, FALSE, "ifaceEvent"),
ifaceEventDoneEvent(FALSE, FALSE, "ifaceEventDoneEvent"),
recvdDisconnectAckEvent(FALSE, FALSE, "recvdDisconnectAckEvent"),
heartBeatCheckTimeSetEvent(FALSE, FALSE, "heartBeatCheckTimeSetEvent"),
sendHeartBeatTimeSetEvent(FALSE, FALSE, "sendHeartBeatTimeSetEvent")

{
   initVariables();
   logCommIO(false);
}


Robot::~Robot()
{
   disconnect();
   WSACleanup();
}


bool Robot::connect(const string hostname)
{
   const string fnName = "connect";

   const string portNumDelim = ":";

   WaitForSingleObject(connectDisconnectMutex, INFINITE);

   if(getConnected())
   {
      fangError(className, fnName,
```



```
         "Already connected, can not connect multiple times.");
      return false;
   }

   const int portNumLocation = hostname.find(portNumDelim);
   string portNumParsedHostname = hostname;
   int actualPort = defaultPort;
   if (portNumLocation != string::npos)
   {
      actualPort = atoi(
         hostname.substr(portNumLocation+1,hostname.size()-1).c_str()
         );
      portNumParsedHostname = hostname.substr(0, portNumLocation);
   }

   const u_long remoteAddress = lookupAddress(portNumParsedHostname);
   if (INADDR_NONE == remoteAddress)
   {
      fangError(className, fnName, WSAGetLastErrorMessage("lookup address"));
      return false;
   }

   in_addr address;
   memcpy(&address, &remoteAddress, sizeof(u_long));

   sd = establishConnection(remoteAddress, htons(actualPort));
   if (INVALID_SOCKET == sd)
   {
      fangError(className, fnName, WSAGetLastErrorMessage("connect to server"));
      return false;
   }
   setConnected(true);

   const string connectAddress = inet_ntoa(address),
      connectPort    = int2string(actualPort);
   fangDebug(className, fnName, "Connected to "
      + connectAddress + ":" + connectPort);

   // We assume that now is the first heartBeat they send us.
   // This is wrong, but simplifies things, and isn't too far off.
   // TODO: fix this.
   setLastHeartBeat(timeGetTime());

   CWinThread *sendThread  = AfxBeginThread(startNetSend,        (LPVOID)this),
      *recvThread          = AfxBeginThread(startNetRecv,        (LPVOID)this),
      *interfaceThread     = AfxBeginThread(startIface,          (LPVOID)this),
      *hearBeatCheckThread = AfxBeginThread(startHeartBeatCheck, (LPVOID)this);

   sendInitCmd();

   CWinThread *sendHeartBeatThread = AfxBeginThread(startSendHeartBeat,
      (LPVOID)this);

   if(!ReleaseMutex(connectDisconnectMutex))
      fangError(className, fnName, "Couldn't release connectDisconnectMutex");

   return true;
}


bool Robot::disconnect()
{
   const string fnName = "disconnect";

   WaitForSingleObject(connectDisconnectMutex, INFINITE);

   if(!getConnected())
   {
      fangError(className, fnName, "Not connected, can't disconnect");
      return false;
   }

   CSingleLock cmdToSend_lock(&cmdToSendMutex, FALSE);
```

```
   CSingleLock cmdRecvd_lock (&cmdRecvdMutex,  FALSE);

   CSingleLock recvdDisconnectAck(&recvdDisconnectAckEvent, FALSE);

   CSingleLock sendEventDone    (&sendEventDoneEvent,          FALSE);
   CSingleLock netRecvEventDone (&netRecvEventDoneEvent,       FALSE);
   CSingleLock heartBeatDone    (&heartBeatCheckEventDoneEvent,FALSE);
   CSingleLock sendHeartBeatDone(&sendHeartBeatEventDoneEvent, FALSE);
   CSingleLock ifaceEventDone   (&ifaceEventDoneEvent,         FALSE);

   const string disconnectCmdName = "disconnect";
   const vector<string> disconnectCmdArgs;
   const Cmd disconnectCmd(disconnectCmdName, disconnectCmdArgs);
   if(!tell(disconnectCmd))
      fangError(className, fnName,
         "Unable to send disconnect cmd, continuing with disconnect routine.");

   // Wait for a "(ack disconnect)" to come back if we have a heartBeat
   // (if we don't have a heartBeat, they may not be able to send an ack)
   // FIXME 4/28 Chris: We never receive the signal for recvdDisconnectAck, why not?
   // Have the false below works around the problem.
   if(false && getHaveHeartBeat())
   {
      recvdDisconnectAck.Lock();
      recvdDisconnectAck.IsLocked();
   }

   // Prevent others from tryign to send/recv data
   cmdToSend_lock.Lock();
   cmdRecvd_lock.Lock();

   setConnected(false);

   sendEvent.SetEvent();
   heartBeatCheckEvent.SetEvent();
   sendHeartBeatEvent.SetEvent();
   sendHeartBeatTimeSetEvent.SetEvent();
   heartBeatCheckTimeSetEvent.SetEvent();
   recvEvent.SetEvent();         // tells iface, a bit misleading. TODO: fix

   sendEventDone.Lock();
   sendEventDone.IsLocked();
   heartBeatDone.Lock();
   heartBeatDone.IsLocked();
   sendHeartBeatDone.Lock();
   sendHeartBeatDone.IsLocked();
   netRecvEventDone.Lock();
   netRecvEventDone.IsLocked();
   ifaceEventDone.Lock();
   ifaceEventDone.IsLocked();

   if (ShutdownConnection(sd))
   {
      fangDebug(className, fnName, "Disconnected from server");
      cmdToSend_lock.Unlock();
      cmdRecvd_lock.Unlock();

      if(!ReleaseMutex(connectDisconnectMutex))
         fangError(className, fnName, "Couldn't release connectDisconnectMutex");

      return true;
   }
   else
   {
      fangDebug(className, fnName,
         WSAGetLastErrorMessage("Shutdown connection"));
      cmdToSend_lock.Unlock();
      cmdRecvd_lock.Unlock();

      if(!ReleaseMutex(connectDisconnectMutex))
         fangError(className, fnName, "Couldn't release connectDisconnectMutex");

      return false;
```



```
   }
}


bool Robot::tell(const Cmd command)
{
   const string fnName = "tell";

   if (!getConnected())
   {
      fangDebug(className, fnName,
         "Called but not connected, not sending cmd");
      return false;
   }

   CSingleLock cmdToSend_lock(&cmdToSendMutex, FALSE);
   CSingleLock sendEventDone(&sendEventDoneEvent, FALSE);

   cmdToSend_lock.Lock();
   cmdToSend = Cmd2String(command);
   sendEvent.SetEvent();
   sendEventDone.Lock();
   sendEventDone.IsLocked();
   cmdToSend_lock.Unlock();

   if (command.getCommand() == "setHeartBeat")
   {
      if(command.getArgs().size() == 0)
         fangError(className, fnName, string("Received ")
            + command.getCommand() + string(" with invalid args."));
      else
         setHeartBeatRate(atoi(command.getArgs()[0].c_str())*interalHBRate2Protocol);
   }

   return true;
}


void Robot::logCommIO(const bool log)
{
   const string fnName = "logCommIO";

   setDebugPrintsCmds(log);
}



//
// Private
//

//
// Begin: Threads
//

UINT Robot::netSend()
{
   const string fnName = "netSend";
   if(showThreadEntrancesExits)
      fangDebug(className, fnName, "Entered");

   while(1)
   {
      CSingleLock myEvent(&sendEvent, FALSE);

      myEvent.Lock();
      if (myEvent.IsLocked())
      {
         if(!getConnected())
         {
            if(showThreadEntrancesExits)
               fangDebug(className, fnName, "Exiting");
            sendEventDoneEvent.SetEvent();
            return 0;
```

```
         }
         const string cmdToSend_copy = cmdToSend;
         sendEventDoneEvent.SetEvent();

         if(send(sd, cmdToSend_copy.c_str(), cmdToSend_copy.size(), 0)
            == SOCKET_ERROR)
         {
            fangError(className, fnName,
               "Received SOCKET_ERROR on send():"
               + WSAGetLastErrorMessage("send echo packet"));
         }
         else
         {
            if (getDebugPrintsCmds())
               fangDebug(className, fnName, "<- " + cmdToSend_copy);
         }
      }
   }

   return 0;
}


void Robot::checkConnected(const string &fnName)
{
   if(!getConnected())
   {
      if(showThreadEntrancesExits)
         fangDebug(className, fnName, "Exiting");
      netRecvEventDoneEvent.SetEvent();
      AfxEndThread(0);
   }
}


UINT Robot::netRecv()
{
   const string fnName = "netRecv";
   if(showThreadEntrancesExits)
      fangDebug(className, fnName, "Entered");

   string buffer;
   const int bufferTmp_size = 1;
   char bufferTmp[bufferTmp_size];
   int recvResult, selectResult;

   timeval selectTimeOut;
   selectTimeOut.tv_sec  = 0;
   selectTimeOut.tv_usec = 25*10000;
   fd_set readDescs;

   while(1)
   {
      CSingleLock cmdRecvd_lock(&cmdRecvdMutex, FALSE);
      CSingleLock recvEventDone(&recvEventDoneEvent, FALSE);

      FD_ZERO(&readDescs);
      FD_SET(sd, &readDescs);
      selectResult = select(1, &readDescs, NULL, NULL, &selectTimeOut);
      if (0 == selectResult)
      {
         // socket time out
      }
      else if (SOCKET_ERROR == selectResult)
      {
         fangError(className, fnName, string("WSAGetLastError: ")
            + int2string(WSAGetLastError()));
      }

      checkConnected(fnName);

      if(FD_ISSET(sd, &readDescs))
      {
```



```
         recvResult = recv(sd, bufferTmp, bufferTmp_size, 0);

         if (SOCKET_ERROR == recvResult)
         {
            fangError(className, fnName, string("WSAGetLastError: ")
               + int2string(WSAGetLastError()));
         }
         else if (0 == recvResult)
         {
            fangDebug(className, fnName,
               "Connection closed by server!");
         }

         buffer += string(bufferTmp, bufferTmp_size);

         if (cmdEndSymbol[0] == bufferTmp[0])
         {
            if (getDebugPrintsCmds())
            {
               fangDebug(className, fnName, string("-> ") + buffer);
            }

            checkConnected(fnName);
            cmdRecvd_lock.Lock();
            cmdRecvd = buffer;
            recvEvent.SetEvent();

            checkConnected(fnName);
            // FIXME: if the iface thread exits between the above checkConnected()
            // and recvEventDone.IsLocked() line, we will be stuck.
            recvEventDone.Lock();
            recvEventDone.IsLocked();
            cmdRecvd_lock.Unlock();
            buffer = "";
         }
      }
   }

   return 0;
}


UINT Robot::iface()
{
   const string fnName = "iface";
   if(showThreadEntrancesExits)
      fangDebug(className, fnName, "Entered");

   while(1)
   {
      CSingleLock myEvent(&recvEvent, FALSE);
      myEvent.Lock();
      if(myEvent.IsLocked())
      {
         if(!getConnected())
         {
            if(showThreadEntrancesExits)
               fangDebug(className, fnName, "Exiting");
            ifaceEventDoneEvent.SetEvent();
            return 0;
         }

         const string recvdCmd = cmdRecvd;
         recvEventDoneEvent.SetEvent();
         const Cmd cmd = string2Cmd(recvdCmd);
         if ("heartBeat" == cmd.getCommand())
         {
            setLastHeartBeat(timeGetTime());
            if (!getHaveHeartBeat())
            {
               const string cmdName = "heartBeat";
               vector<string> cmdArgs;
               cmdArgs.push_back("Good");
```

```
               guiUpdate(Cmd(cmdName, cmdArgs));

               setHaveHeartBeat(true);
               fangDebug(className, fnName, "Regained heartBeat");
            }

         }
         else if ("unknownCmd" == cmd.getCommand())
         {
            vector<string> origCmdArgs;
            for(unsigned int i=1; i < cmd.getArgs().size(); i++)
               origCmdArgs.push_back(cmd.getArgs()[i]);

            fangError(className, fnName,
               string("unknownCmd report by server on: ")
               + Cmd2String(Cmd(cmd.getArgs()[0], origCmdArgs)));
         }
         else if ("invalidArgs" == cmd.getCommand())
         {
            vector<string> origCmdArgs;
            for(unsigned int i=1; i < cmd.getArgs().size(); i++)
               origCmdArgs.push_back(cmd.getArgs()[i]);

            fangError(className, fnName,
               string("invalidArgs report by server on: ")
               + Cmd2String(Cmd(cmd.getArgs()[0], origCmdArgs)));
         }
         else if ("ack" == cmd.getCommand()
              && cmd.getArgs().size() != 0
                 && "disconnect" == cmd.getArgs()[0])
         {
            recvdDisconnectAckEvent.SetEvent();
         fangDebug(className, fnName, "recvdDisconnectAckEvent.SetEvent()");
         }
         else
         {
            guiUpdate(cmd);
         }
      }
   }

   return 0;
}


UINT Robot::heartBeatCheck()
{
   const string fnName = "heartBeatCheck";
   if(showThreadEntrancesExits)
      fangDebug(className, fnName, "Entered");

   while(1)
   {
      CSingleLock heartBeatCheckKill(&heartBeatCheckEvent, FALSE);
      heartBeatCheckKill.Lock(0);
      if(heartBeatCheckKill.IsLocked())
      {
         if(!getConnected())
         {
            if(showThreadEntrancesExits)
               fangDebug(className, fnName, "Exiting");
            heartBeatCheckEventDoneEvent.SetEvent();
            return 0;
         }
         else
            fangDebug(className, fnName, "Why was I signaled?");
      }
      if(getHeartBeatRate() == 0)
      {
         CSingleLock heartBeatCheckTimeSet(&heartBeatCheckTimeSetEvent,
            FALSE);
         // Wait until heartBeatRate is no longer zero
         heartBeatCheckTimeSet.Lock();
```



```
         heartBeatCheckTimeSet.IsLocked();
      }

      currentTime timeNow   = timeGetTime();
      currentTime heartBeatN = getLastHeartBeat();

      timeSpan timeToSleep = getHeartBeatRate();
      if(timeToSleep < timeNow - heartBeatN)
      {
         setHaveHeartBeat(false);
         fangDebug(className, fnName, "No heartBeat");
      }
      else
         timeToSleep += heartBeatN - timeNow;

      Sleep(timeToSleep);
   }
   return 0;
}


// TODO: check that this updates when setHeartBeat sent
UINT Robot::sendHeartBeat()
{
   const string fnName = "sendHeartBeat";
   if(showThreadEntrancesExits)
      fangDebug(className, fnName, "Entered");

   const string cmdName = "heartBeat";
   const vector<string> cmdArgs;
   const Cmd heartBeatCmd(cmdName, cmdArgs);

   while(1)
   {
      CSingleLock sendHeartBeatKill(&sendHeartBeatEvent, FALSE);
      sendHeartBeatKill.Lock(0);
      if(sendHeartBeatKill.IsLocked())
      {
         if(!getConnected())
         {
            if(showThreadEntrancesExits)
               fangDebug(className, fnName, "Exiting");
            sendHeartBeatEventDoneEvent.SetEvent();
            return 0;
         }
         else
            fangDebug(className, fnName, "Why was I signaled?");
      }
      if(getHeartBeatRate() == 0)
      {
         CSingleLock sendHeartBeatTimeSet(&sendHeartBeatTimeSetEvent,
            FALSE);
         // Wait for heartBeatRate to change from zero
         sendHeartBeatTimeSet.Lock();
         sendHeartBeatTimeSet.IsLocked();
      }

      // Go a little faster than necessary to make sure our heartBeats
      // get there in time
      Sleep(getHeartBeatRate()/2);
      if(!tell(heartBeatCmd) && getConnected())
         fangError(className, fnName, "Unable to send heartBeat");
   }
   return 0;
}


//
// End: Threads
//

Cmd Robot::string2Cmd(const string command) const
{
```

```
   const string fnName = "string2Cmd";

   vector<string> cmdArgs;
   unsigned int posBegin = command.find(cmdBeginSymbol),
      posEnd   = command.find(cmdDelimSymbol);
   if(string::npos == posBegin)
   {
      fangError(className, fnName, "Received invalid command to parse: "
         + command);
      const vector<string> emptyArgs;
      const Cmd invalidCmd("", emptyArgs);
      return invalidCmd;
   }
   else if (string::npos == posEnd)
   {
      posEnd = command.find(cmdEndSymbol);
      if(string::npos == posEnd)
      {
         fangError(className, fnName, "Received invalid command to parse: "
            + command);
         const vector<string> emptyArgs;
         const Cmd invalidCmd("", emptyArgs);
         return invalidCmd;
      }
   }
   const string cmdName = command.substr(posBegin + cmdBeginSymbol.size(),
                                     posEnd   - cmdDelimSymbol.size());

   while(1)
   {
      posBegin = command.find(cmdDelimSymbol, posEnd);
      if(string::npos == posBegin)
         break;
      posEnd   = command.find(cmdDelimSymbol, posBegin+cmdDelimSymbol.size());
      if(string::npos == posEnd)
      {
         posEnd = command.find(cmdEndSymbol, posBegin+cmdDelimSymbol.size());
         if(string::npos == posEnd)
         {
            fangError(className, fnName,
               "Received invalid command to parse: " + command);
            const vector<string> emptyArgs;
            const Cmd invalidCmd("", emptyArgs);
            return invalidCmd;
         }
      }
     if (cmdDelimSymbol.size() != cmdEndSymbol.size())
     {
      // This code can't handle this case
      fangError(className, fnName,
         "Can't handle cmdEndSymbol.size() !+ cmdDelimSymbol.size()");
      const vector<string> emptyArgs;
      const Cmd invalidCmd("", emptyArgs);
        return invalidCmd;
     }

      cmdArgs.push_back(command.substr(posBegin + cmdDelimSymbol.size(),
                                  posEnd   - posBegin - 1));
   }

   return Cmd(cmdName, cmdArgs);
}


string Robot::Cmd2String(const Cmd command) const
{
   const string fnName = "Cmd2String";

   const vector<string> args = command.getArgs();
   string commandString = cmdBeginSymbol + command.getCommand();
   for(unsigned int i=0; i < args.size(); i++)
      commandString += cmdDelimSymbol + args[i];
   commandString += cmdEndSymbol;
```



```
   return commandString;
}


void Robot::initVariables()
{
   const string fnName = "initVariables";

   setConnected(false);
   setGui(NULL);

   const int nCode = WSAStartup(MAKEWORD(1, 1), &wsaData);
   if (0 != nCode)
   {
      const string errorMsg = string("WSAStartup() returned error: ")
         + int2string(nCode);
      fangError(className, fnName, errorMsg);
   }
}


void Robot::sendInitCmd()
{
   const string fnName = "sendInitCmd";

   const string cmdName = "init",
      rightWheelSpeed = "100", leftWheelSpeed = "100",
      panCameraSpeed  = "100", tiltCameraSpeed= "100",
      stopOnBump = "0";

   vector<string> cmdArgs;
   cmdArgs.push_back(rightWheelSpeed);
   cmdArgs.push_back(leftWheelSpeed);
   cmdArgs.push_back(panCameraSpeed);
   cmdArgs.push_back(tiltCameraSpeed);
   cmdArgs.push_back(stopOnBump);
   cmdArgs.push_back(int2string(getHeartBeatRate()/interalHBRate2Protocol));

   if(!tell(Cmd(cmdName, cmdArgs)))
      fangError(className, fnName, "Unable to send init cmd");
}


void Robot::guiUpdate(const Cmd cmd)
{
   const string fnName = "guiUpdate";

   CSingleLock gui_lock(&guiMutex, FALSE);
   gui_lock.Lock();

   if(NULL != gui)
      gui->update(cmd);
   else
      fangError(className, fnName, "gui==NULL");

   gui_lock.Unlock();
}


// Given an address string, determine if it's a dotted-quad IP address
// or a domain address.  If the latter, ask DNS to resolve it.  In
// either case, return resolved IP address.  If we fail, we return
// INADDR_NONE.
u_long Robot::lookupAddress(const string pcHost) const
{
   u_long remoteAddr = inet_addr(pcHost.c_str());
   if (INADDR_NONE == remoteAddr)
   {
      // pcHost isn't a dotted IP, so resolve it through DNS
      const hostent * const pHE = gethostbyname(pcHost.c_str());
      if (0 == pHE)
      {
```

```
         return INADDR_NONE;
      }
      remoteAddr = *((u_long*)pHE->h_addr_list[0]);
   }

   return remoteAddr;
}


// Connects to a given address, on a given port, both of which must be
// in network byte order.  Returns newly-connected socket if we succeed,
// or INVALID_SOCKET if we fail.
SOCKET Robot::establishConnection(const u_long remoteAddr,const u_short port)const
{
   SOCKET sd = socket(AF_INET, SOCK_STREAM, 0);
   if (sd != INVALID_SOCKET)
   {
      sockaddr_in sinRemote;
      sinRemote.sin_family = AF_INET;
      sinRemote.sin_addr.s_addr = remoteAddr;
      sinRemote.sin_port = port;
      if (::connect(sd, (sockaddr*)&sinRemote, sizeof(sockaddr_in)) ==
         SOCKET_ERROR)
      {
         sd = INVALID_SOCKET;
      }
   }

   return sd;
}


bool Robot::setGui(CFangDlg *gui)
{
   const string fnName = "setGui";

   CSingleLock gui_lock(&guiMutex, FALSE);
   gui_lock.Lock();
   this->gui = gui;
   gui_lock.Unlock();

   return true;
}

//
// private member access functions for non-thread safe data members
//

timeSpan Robot::getHeartBeatRate()
{
   CSingleLock heartBeatRate_lock(&heartBeatRateMutex, FALSE);
   heartBeatRate_lock.Lock();
   const timeSpan heartBeatRate_copy = heartBeatRate;
   heartBeatRate_lock.Unlock();

   return heartBeatRate_copy;
}


void Robot::setHeartBeatRate(const timeSpan newHeartBeatRate)
{
   CSingleLock heartBeatRate_lock(&heartBeatRateMutex, FALSE);
   bool startThreads = false,
      stopThreads  = false;
   const timeSpan oldHeartBeatRate = getHeartBeatRate();

   heartBeatRate_lock.Lock();
   heartBeatRate = newHeartBeatRate;
   heartBeatRate_lock.Unlock();

   if(newHeartBeatRate != 0 && oldHeartBeatRate == 0)
   {
      // Tell the two heartBeat threads getHeartBeatRate() != 0 anymore
```



```
      sendHeartBeatTimeSetEvent.SetEvent();
      heartBeatCheckTimeSetEvent.SetEvent();
   }
}


currentTime Robot::getLastHeartBeat()
{
   CSingleLock lastHeartBeat_lock(&lastHeartBeatMutex, FALSE);
   lastHeartBeat_lock.Lock();
   const currentTime lastHeartBeat_copy = lastHeartBeat;
   lastHeartBeat_lock.Unlock();

   return lastHeartBeat_copy;
}


void Robot::setLastHeartBeat(const currentTime newLastHeartBeat)
{
   CSingleLock lastHeartBeat_lock(&lastHeartBeatMutex, FALSE);
   lastHeartBeat_lock.Lock();
   lastHeartBeat = newLastHeartBeat;
   lastHeartBeat_lock.Unlock();
}


bool Robot::getHaveHeartBeat()
{
   CSingleLock haveHeartBeat_lock(&haveHeartBeatMutex, FALSE);
   haveHeartBeat_lock.Lock();
   const bool haveHeartBeat_copy = haveHeartBeat;
   haveHeartBeat_lock.Unlock();

   return haveHeartBeat_copy;
}


void Robot::setHaveHeartBeat(const bool haveIt)
{
   const string fnName = "setHaveHeartBeat";

   if(!haveIt && getHaveHeartBeat())
   {
      const string cmdName = "heartBeat";
      vector<string> cmdArgs;
      cmdArgs.push_back("Lost");
      guiUpdate(Cmd(cmdName, cmdArgs));
   }

   CSingleLock haveHeartBeat_lock(&haveHeartBeatMutex, FALSE);
   haveHeartBeat_lock.Lock();
   haveHeartBeat = haveIt;
   haveHeartBeat_lock.Unlock();
}


bool Robot::getConnected()
{
   CSingleLock connected_lock(&connectedMutex, FALSE);
   connected_lock.Lock();
   const bool connected_copy = connected;
   connected_lock.Unlock();

   return connected_copy;
}


void Robot::setConnected(const bool newConnected)
{
   CSingleLock connected_lock(&connectedMutex, FALSE);
   connected_lock.Lock();
   connected = newConnected;
   connected_lock.Unlock();
```

```
}


bool Robot::getDebugPrintsCmds()
{
   CSingleLock debugPrintsCmds_lock(&debugPrintsCmdsMutex, FALSE);
   debugPrintsCmds_lock.Lock();
   const bool debugPrintsCmds_copy = debugPrintsCmds;
   debugPrintsCmds_lock.Unlock();

   return debugPrintsCmds_copy;
}


void Robot::setDebugPrintsCmds(const bool newDebugPrintsCmds)
{
   CSingleLock debugPrintsCmds_lock(&debugPrintsCmdsMutex, FALSE);
   debugPrintsCmds_lock.Lock();
   debugPrintsCmds = newDebugPrintsCmds;
   debugPrintsCmds_lock.Unlock();
}


//
// start[NetSend|NetRecv|iface] are functions that call their respective
// member fns
//

UINT startNetSend(LPVOID param)
{
   const string fnName = "startNetSend";

   if(NULL != param)
      return ((Robot*)param)->netSend();
   else
   {
      fangError("", fnName,
         "Received NULL pointer, not starting Robot::netSend thread");
      return -1;
   }
}


UINT startNetRecv(LPVOID param)
{
   const string fnName = "startNetRecv";

   if(NULL != param)
      return ((Robot*)param)->netRecv();
   else
   {
      fangError("", fnName,
         "Received NULL pointer, not starting Robot::netRecv thread");
      return -1;
   }
}


UINT startIface(LPVOID param)
{
   const string fnName = "startIface";

   if(NULL != param)
      return ((Robot*)param)->iface();
   else
   {
      fangError("", fnName,
         "Received NULL pointer, not starting Robot::iface thread");
      return -1;
   }
}
```



```
UINT startHeartBeatCheck(LPVOID param)
{
   const string fnName = "startNetRecv";

   if(NULL != param)
      return ((Robot*)param)->heartBeatCheck();
   else
   {
      fangError("", fnName,
         "Received NULL pointer, not starting Robot::heartBeatCheck thread");
      return -1;
   }
}


UINT startSendHeartBeat(LPVOID param)
{
   const string fnName = "startSendHeartBeat";

   if(NULL != param)
      return ((Robot*)param)->sendHeartBeat();
   else
   {
      fangError("", fnName,
         "Received NULL pointer, not starting Robot::sendHeartBeat thread");
      return -1;
   }
}
```

```
// $Id: ws-util.h,v 1.2 2002/04/21 19:52:54 sjr4j Exp $
// Flancrest Enterprises, Group 22
// About this file:
// TODO

#if !defined(WS_UTIL_H)
#define WS_UTIL_H

// Uncomment one.
#include <winsock.h>
//#include <winsock2.h>

#include <iostream>
#include <string>

using namespace std;

extern string WSAGetLastErrorMessage(const char* pcMessagePrefix,
int nErrorID = 0);
extern bool ShutdownConnection(SOCKET sd);
#endif                           // !defined (WS_UTIL_H)
```

```
// $Id: ws-util.cpp,v 1.3 2002/04/21 19:52:54 sjr4j Exp $
// Flancrest Enterprises, Group 22

#include "stdafx.h"
#include "errors.h"
#include "ws-util.h"

#include <iostream>
#include <algorithm>
#include <strstream>

using namespace std;

#if !defined(_WINSOCK2API_)
// Winsock 2 header defines this, but Winsock 1.1 header doesn't.  In
// the interest of not requiring the Winsock 2 SDK which we don't really
// need, we'll just define this one constant ourselves.
#define SD_SEND 1
#endif

//// Constants /////////////////////////////////////////////////////////

const int kBufferSize = 1024;

//// Statics ///////////////////////////////////////////////////////////

// List of Winsock error constants mapped to an interpretation string.
// Note that this list must remain sorted by the error constants'
// values, because we do a binary search on the list when looking up
// items.
static struct ErrorEntry
{
   int nID;
   const char* pcMessage;

   ErrorEntry(int id, const char* pc = 0) :
   nID(id),
      pcMessage(pc)
   {
   }

   bool operator<(const ErrorEntry& rhs)
   {
      return nID < rhs.nID;
   }
}


gaErrorList[] =
{
   ErrorEntry(0,                  "No error"),
   ErrorEntry(WSAEINTR,           "Interrupted system call"),
   ErrorEntry(WSAEBADF,           "Bad file number"),
   ErrorEntry(WSAEACCES,          "Permission denied"),
   ErrorEntry(WSAEFAULT,          "Bad address"),
   ErrorEntry(WSAEINVAL,          "Invalid argument"),
   ErrorEntry(WSAEMFILE,          "Too many open sockets"),
   ErrorEntry(WSAEWOULDBLOCK,     "Operation would block"),
   ErrorEntry(WSAEINPROGRESS,     "Operation now in progress"),
   ErrorEntry(WSAEALREADY,        "Operation already in progress"),
   ErrorEntry(WSAENOTSOCK,        "Socket operation on non-socket"),
   ErrorEntry(WSAEDESTADDRREQ,    "Destination address required"),
   ErrorEntry(WSAEMSGSIZE,        "Message too long"),
   ErrorEntry(WSAEPROTOTYPE,      "Protocol wrong type for socket"),
   ErrorEntry(WSAENOPROTOOPT,     "Bad protocol option"),
   ErrorEntry(WSAEPROTONOSUPPORT, "Protocol not supported"),
   ErrorEntry(WSAESOCKTNOSUPPORT, "Socket type not supported"),
   ErrorEntry(WSAEOPNOTSUPP,      "Operation not supported on socket"),
   ErrorEntry(WSAEPFNOSUPPORT,    "Protocol family not supported"),
   ErrorEntry(WSAEAFNOSUPPORT,    "Address family not supported"),
   ErrorEntry(WSAEADDRINUSE,      "Address already in use"),
   ErrorEntry(WSAEADDRNOTAVAIL,   "Can't assign requested address"),
   ErrorEntry(WSAENETDOWN,        "Network is down"),
```



```
   ErrorEntry(WSAENETUNREACH,     "Network is unreachable"),
   ErrorEntry(WSAENETRESET,       "Net connection reset"),
   ErrorEntry(WSAECONNABORTED,    "Software caused connection abort"),
   ErrorEntry(WSAECONNRESET,      "Connection reset by peer"),
   ErrorEntry(WSAENOBUFS,         "No buffer space available"),
   ErrorEntry(WSAEISCONN,         "Socket is already connected"),
   ErrorEntry(WSAENOTCONN,        "Socket is not connected"),
   ErrorEntry(WSAESHUTDOWN,       "Can't send after socket shutdown"),
   ErrorEntry(WSAETOOMANYREFS,    "Too many references, can't splice"),
   ErrorEntry(WSAETIMEDOUT,       "Connection timed out"),
   ErrorEntry(WSAECONNREFUSED,    "Connection refused"),
   ErrorEntry(WSAELOOP,           "Too many levels of symbolic links"),
   ErrorEntry(WSAENAMETOOLONG,    "File name too long"),
   ErrorEntry(WSAEHOSTDOWN,       "Host is down"),
   ErrorEntry(WSAEHOSTUNREACH,    "No route to host"),
   ErrorEntry(WSAENOTEMPTY,       "Directory not empty"),
   ErrorEntry(WSAEPROCLIM,        "Too many processes"),
   ErrorEntry(WSAEUSERS,          "Too many users"),
   ErrorEntry(WSAEDQUOT,          "Disc quota exceeded"),
   ErrorEntry(WSAESTALE,          "Stale NFS file handle"),
   ErrorEntry(WSAEREMOTE,         "Too many levels of remote in path"),
   ErrorEntry(WSASYSNOTREADY,     "Network system is unavailable"),
   ErrorEntry(WSAVERNOTSUPPORTED, "Winsock version out of range"),
   ErrorEntry(WSANOTINITIALISED,  "WSAStartup not yet called"),
   ErrorEntry(WSAEDISCON,         "Graceful shutdown in progress"),
   ErrorEntry(WSAHOST_NOT_FOUND,  "Host not found"),
   ErrorEntry(WSANO_DATA,         "No host data of that type was found")
};
const int kNumMessages = sizeof(gaErrorList) / sizeof(ErrorEntry);

//// WSAGetLastErrorMessage ////////////////////////////////////////////
// A function similar in spirit to Unix's perror() that tacks a canned
// interpretation of the value of WSAGetLastError() onto the end of a
// passed string, separated by a ": ".  Generally, you should implement
// smarter error handling than this, but for default cases and simple
// programs, this function is sufficient.
//
// This function returns a pointer to an internal static buffer, so you
// must copy the data from this function before you call it again.  It
// follows that this function is also not thread-safe.

string WSAGetLastErrorMessage(const char* pcMessagePrefix,
int nErrorID )                    /* = 0 */
{
   // Build basic error string
   static char acErrorBuffer[256];
   ostrstream outs(acErrorBuffer, sizeof(acErrorBuffer));
   outs << pcMessagePrefix << ": ";

   // Tack appropriate canned message onto end of supplied message
   // prefix. Note that we do a binary search here: gaErrorList must be
   // sorted by the error constant's value.
   ErrorEntry* pEnd = gaErrorList + kNumMessages;
   ErrorEntry Target(nErrorID ? nErrorID : WSAGetLastError());
   ErrorEntry* it = lower_bound(gaErrorList, pEnd, Target);
   if ((it != pEnd) && (it->nID == Target.nID))
   {
      outs << it->pcMessage;
   }
   else
   {
      // Didn't find error in list, so make up a generic one
      outs << "unknown error";
   }
   outs << " (" << Target.nID << ")";

   // Finish error message off and return it.
   outs << ends;
   acErrorBuffer[sizeof(acErrorBuffer) - 1] = '\0';
   const string acErrorBuffer_asString(acErrorBuffer);
   return acErrorBuffer_asString;
}
```

```
//// ShutdownConnection ////////////////////////////////////////////////
// Gracefully shuts the connection sd down.  Returns true if we're
// successful, false otherwise.

bool ShutdownConnection(SOCKET sd)
{
   const string fnName = "ShutdownConnection";

   // Disallow any further data sends.  This will tell the other side
   // that we want to go away now.  If we skip this step, we don't
   // shut the connection down nicely.
   if (shutdown(sd, SD_SEND) == SOCKET_ERROR)
   {
      return false;
   }

   // Receive any extra data still sitting on the socket.  After all
   // data is received, this call will block until the remote host
   // acknowledges the TCP control packet sent by the shutdown above.
   // Then we'll get a 0 back from recv, signalling that the remote
   // host has closed its side of the connection.
   char acReadBuffer[kBufferSize];
   while (1)
   {
      int nNewBytes = recv(sd, acReadBuffer, kBufferSize, 0);
      if (nNewBytes == SOCKET_ERROR)
      {
         return false;
      }
      else if (nNewBytes != 0)
      {
         fangDebug("", fnName, string("FYI, received ")
            + int2string(nNewBytes)
            + string(" unexpected bytes during shutdown."));
      }
      else
      {
         // Okay, we're done!
         break;
      }
   }

   // Close the socket.
   if (closesocket(sd) == SOCKET_ERROR)
   {
      return false;
   }

   return true;
}
```

```
#if !defined(AFX_CONFIGDLG_H__0FCAF11C_8B34_4BBC_8A4E_1A0E3C5F4249__INCLUDED_)
#define AFX_CONFIGDLG_H__0FCAF11C_8B34_4BBC_8A4E_1A0E3C5F4249__INCLUDED_

#include "resource.h"            //main symbols

#if _MSC_VER > 1000
#pragma once
#endif                              // _MSC_VER > 1000
// ConfigDlg.h : header file
//
class CFangDlg;
/////////////////////////////////////////////////////////////////////////////
// CConfigDlg dialog

class CConfigDlg : public CDialog
{
   // Construction
   public:
      void setfangDlg(CFangDlg* fangDlg);
                                    // standard constructor
      CConfigDlg(CWnd* pParent = NULL);

      // Dialog Data
      //{{AFX_DATA(CConfigDlg)
      enum { IDD = IDD_CONFIGBOX };
      // NOTE: the ClassWizard will add data members here
      //}}AFX_DATA

      // Overrides
      // ClassWizard generated virtual function overrides
      //{{AFX_VIRTUAL(CConfigDlg)
   protected:
                                    // DDX/DDV support
      virtual void DoDataExchange(CDataExchange* pDX);
      //}}AFX_VIRTUAL

      // Implementation
   protected:

      // Generated message map functions
      //{{AFX_MSG(CConfigDlg)
      afx_msg void OnButtonApply();
      afx_msg void OnCheckDebug();
      afx_msg void OnCheckBump();
      afx_msg void OnCheckLogio();
      //}}AFX_MSG
      DECLARE_MESSAGE_MAP()
      private:
      CFangDlg* fangDlg;
      BOOL OnInitDialog();
};

//{{AFX_INSERT_LOCATION}}
// Microsoft Visual C++ will insert additional declarations immediately before the pr
evious line.
#endif                              // !defined(AFX_CONFIGDLG_H__0FCAF11C_8B34_4BBC_8A4E
_1A0E3C5F4249__INCLUDED_)
```

```
// ConfigDlg.cpp : implementation file
//

#include "stdafx.h"
#include "fang.h"
#include "fangDlg.h"
#include "ConfigDlg.h"
#include "errors.h"

#ifdef _DEBUG
#define new DEBUG_NEW
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
#endif

/////////////////////////////////////////////////////////////////////////////
// CConfigDlg dialog

                                 /*=NULL*/
CConfigDlg::CConfigDlg(CWnd* pParent )
: CDialog(CConfigDlg::IDD, pParent)
{
   //{{AFX_DATA_INIT(CConfigDlg)
   // NOTE: the ClassWizard will add member initialization here
   //}}AFX_DATA_INIT
}


void CConfigDlg::DoDataExchange(CDataExchange* pDX)
{
   CDialog::DoDataExchange(pDX);
   //{{AFX_DATA_MAP(CConfigDlg)
   // NOTE: the ClassWizard will add DDX and DDV calls here
   //}}AFX_DATA_MAP
}


BEGIN_MESSAGE_MAP(CConfigDlg, CDialog)
//{{AFX_MSG_MAP(CConfigDlg)
ON_BN_CLICKED(IDC_BUTTON_APPLY, OnButtonApply)
ON_BN_CLICKED(IDC_CHECK_DEBUG, OnCheckDebug)
ON_BN_CLICKED(IDC_CHECK_BUMP, OnCheckBump)
ON_BN_CLICKED(IDC_CHECK_LOGIO, OnCheckLogio)
//}}AFX_MSG_MAP
END_MESSAGE_MAP()

/////////////////////////////////////////////////////////////////////////////
// CConfigDlg message handlers

void CConfigDlg::setfangDlg(CFangDlg *fangDlg)
{
   this->fangDlg = fangDlg;
}


void CConfigDlg::OnButtonApply()
{
   CString text;
   string temp;

   GetDlgItemText(IDC_EDIT_BUMP_ANGLE, text);
   temp = text;
   if(temp != "")
      fangDlg->setbumpAngle(temp);

   GetDlgItemText(IDC_EDIT_CAM_BUMP_ANGLE, text);
   temp = text;
   if(temp != "")
      fangDlg->setbumpCamAngle(temp);

   GetDlgItemText(IDC_EDIT_BUMP_DISTANCE, text);
   temp = text;
   if(temp != "")
```



```
      fangDlg->setbumpDistance(temp);

   GetDlgItemText(IDC_EDIT_BUMP_RATE, text);
   temp = text;
   if(temp != "")
      fangDlg->setbumperRate(temp);

   GetDlgItemText(IDC_EDIT_CONFIG_RATE, text);
   temp = text;
   if(temp != "")
      fangDlg->setconfigRate(temp);

   GetDlgItemText(IDC_EDIT_HEARTBEAT_RATE, text);
   temp = text;
   if(temp != "")
      fangDlg->setheartbeatRate(temp);

   GetDlgItemText(IDC_EDIT_SONAR_RATE, text);
   temp = text;
   if(temp != "")
      fangDlg->setsonarRate(temp);

   GetDlgItemText(IDC_EDIT_SPEED_RATE, text);
   temp = text;
   if(temp != "")
      fangDlg->setspeedRate(temp);
}


void CConfigDlg::OnCheckDebug()
{
   int n = IsDlgButtonChecked(IDC_CHECK_DEBUG);
   if(n != 0)
      fangDlg->ShowDebugWindow(true);
   else
      fangDlg->ShowDebugWindow(false);
}


void CConfigDlg::OnCheckBump()
{
   int n = IsDlgButtonChecked(IDC_CHECK_BUMP);

   if(n != 0)
      fangDlg->StopOnBump(true);
   else
      fangDlg->StopOnBump(false);
}


BOOL CConfigDlg::OnInitDialog()
{
   CDialog::OnInitDialog();

   int text;

   text = fangDlg->getbumpAngle();
   SetDlgItemText(IDC_EDIT_BUMP_ANGLE, int2string(text).c_str());

   text = fangDlg->getbumpCamAngle();
   SetDlgItemText(IDC_EDIT_CAM_BUMP_ANGLE, int2string(text).c_str());

   text = fangDlg->getbumpDistance();
   SetDlgItemText(IDC_EDIT_BUMP_DISTANCE, int2string(text).c_str());

   text = fangDlg->getbumperRate();
   SetDlgItemText(IDC_EDIT_BUMP_RATE, int2string(text).c_str());

   text = fangDlg->getconfigRate();
   SetDlgItemText(IDC_EDIT_CONFIG_RATE, int2string(text).c_str());

   text = fangDlg->getheartbeatRate();
   SetDlgItemText(IDC_EDIT_HEARTBEAT_RATE, int2string(text).c_str());
```

```
   text = fangDlg->getsonarRate();
   SetDlgItemText(IDC_EDIT_SONAR_RATE, int2string(text).c_str());

   text = fangDlg->getspeedRate();
   SetDlgItemText(IDC_EDIT_SPEED_RATE, int2string(text).c_str());

   if(fangDlg->getShowDebugWindow())
      CDialog::CheckDlgButton(IDC_CHECK_DEBUG, 1);

   if(fangDlg->getStopOnBump())
      CDialog::CheckDlgButton(IDC_CHECK_BUMP, 1);

   if(fangDlg->getLogIO())
      CDialog::CheckDlgButton(IDC_CHECK_LOGIO, 1);

   return TRUE;                  // return TRUE  unless you set the focus to a contro
l
}


void CConfigDlg::OnCheckLogio()
{
   int n = IsDlgButtonChecked(IDC_CHECK_LOGIO);

   if(n != 0)
      fangDlg->LogCommIO(true);
   else
      fangDlg->LogCommIO(false);
}
```

```
// $Id: fangDlg.h,v 1.21 2002/04/30 19:47:19 el9d Exp $
// Flancrest Enterprises, Group 22
// About this file:
// This class hides the particular way that Cmds are handled, as well
// as the visual user interface's implementation.

#if !defined(AFX_FANGDLG_H__BE51AAE4_63A8_4C63_B4D5_77E48D74057D__INCLUDED_)
#define AFX_FANGDLG_H__BE51AAE4_63A8_4C63_B4D5_77E48D74057D__INCLUDED_

#include "cmd.h"
#include "ConfigDlg.h"

#include "resource.h"            // main symbols

#if _MSC_VER > 1000
#pragma once
#endif                          // _MSC_VER > 1000

/////////////////////////////////////////////////////////////////////////////
// CFangDlg dialog

class Robot;
class CConfigDlg;

class CFangDlg : public CDialog
{
   // Construction
   public:
      //functions to get and set private data members
      bool getLogIO();
      void LogCommIO(bool log);
      void setspeedRate(string rate);
      void setsonarRate(string rate);
      void setheartbeatRate(string rate);
      void setconfigRate(string rate);
      void setbumperRate(string rate);
      void setbumpDistance(string distance);
      void setbumpCamAngle(string angle);
      void setbumpAngle(string angle);
      void setspeedRate(int rate);
      void setsonarRate(int rate);
      void setheartbeatRate(int rate);
      void setconfigRate(int rate);
      void setbumperRate(int rate);
      void setbumpDistance(int distance);
      void setbumpCamAngle(int angle);
      void setbumpAngle(int angle);
      int getbumpAngle();
      int getbumpCamAngle();
      int getbumpDistance();
      int getbumperRate();
      int getconfigRate();
      int getheartbeatRate();
      int getsonarRate();
      int getspeedRate();
      bool getShowDebugWindow();
      bool getStopOnBump();

      int getMode();
      void setMode(int mode);
      // standard constructor
      CFangDlg(CWnd* pParent = NULL);
      // associates an instance of the robot class with this class
      bool setRobot(Robot *robot);
      // updates the status information member variables and displays them
      // visually.  command is a Cmd containing the update information.
      void update(Cmd command);
      // sets whether to stop on bump (true) or not (false)
      void StopOnBump(bool stop);
      // sets whether to show debug window (true) or not (false)
      void ShowDebugWindow(bool show);

      //function to start joystick thread
```



```
      friend UINT startJoystick(LPVOID param);

      // Dialog Data
      //{{AFX_DATA(CFangDlg)
      enum { IDD = IDD_FANG_DIALOG };
      CSliderCtrl m_robotSpeed;
      CSliderCtrl m_camTilt;
      CSliderCtrl m_camPan;
      //}}AFX_DATA

      // ClassWizard generated virtual function overrides
      //{{AFX_VIRTUAL(CFangDlg)
   protected:
                                       // DDX/DDV support
      virtual void DoDataExchange(CDataExchange* pDX);
      //}}AFX_VIRTUAL

      // Implementation
   protected:
      HICON m_hIcon;
      BOOL PreTranslateMessage(MSG* pMsg);

   private:
      //stores current key pressed (used for keyboard input)
      UINT keyDown;
      //private data members to store user prefernces from Config box
      bool logIO;
      bool showDebugWindow;
      bool stopOnBump;
      int bumpAngle;
      int bumpCamAngle;
      int bumpDistance;
      int heartbeatRate;
      int speedRate;
      int configRate;
      int sonarRate;
      int bumperRate;
      //function to scale sonar data (may change to display better)
      double scale(double input);
      //accepts joystick input
      UINT pollJoystick();
     //invalidates client drawing area for redraw
      void PrePaint(vector<bool> sonars_changed, bool bumpers_changed);
      //internal variables to keep track of values sent to robot
      int currSpeed;
      int currCamTilt;
      int currCamPan;
      //consts for boundaries of variables not likely to change
      const int MIN_SPEED;
      const int MAX_SPEED;
      const int MAX_TILT_RANGE;
      const int MIN_TILT_RANGE;
      const int MAX_PAN_RANGE;
      const int MIN_PAN_RANGE;
      const int MIN_SONAR_DIST;
      const int MAX_SONAR_DIST;
      const int NUM_SONARS;
      const int NUM_BUMPERS;
      const int NUM_SPEEDS;
      //fastest and slowest rate to send data from robot (in Hz)
      const int MIN_RATE;
      const int MAX_RATE;
      //time between joystick pollings
      const int sleepTime;
      const int turnMult;
     //center of drawing area
     const int YCENTER;
     const int XCENTER;
      //indicates whether robot should move in steps (0 = steps, 1 = run)
      int mode;
      //takes a string and converts it to an int
      int string2int(string input);
      //for use in printing debug info
```

```
      const string className;

      // provides potential initialization not included in constructor
      void setup ();

      // Associations
      Robot *robot;

      // data structures to hold data from robot
      vector<int> sonar_data;
      vector<bool> bumper_data;
      vector<int> speed;
      bool robot_busy;
      bool camera_busy;
      bool connected;
      bool heartbeat;

      // Generated message map functions
      //{{AFX_MSG(CFangDlg)
      virtual BOOL OnInitDialog();
      afx_msg void OnSysCommand(UINT nID, LPARAM lParam);
      afx_msg void OnPaint();
      afx_msg HCURSOR OnQueryDragIcon();
      afx_msg void OnButtonLeft();
      afx_msg void OnButtonForward();
      afx_msg void OnButtonRight();
      afx_msg void OnButtonStop();
      afx_msg void OnButtonCamRight();
      afx_msg void OnButtonCamUp();
      afx_msg void OnButtonCamDown();
      afx_msg void OnButtonCamLeft();
      afx_msg void OnButtonConnect();
      afx_msg void OnButtonBack();
      afx_msg void OnButtonHome();
      afx_msg void OnButtonFwdright();
      afx_msg void OnButtonFwdleft();
      afx_msg void OnButtonBackleft();
      afx_msg void OnButtonBackright();
      afx_msg void OnButtonMode();
      afx_msg void OnButtonReset();
      afx_msg void OnReleasedcaptureSliderCampan(NMHDR* pNMHDR, LRESULT* pResult);
      afx_msg void OnReleasedcaptureSliderCamtilt(NMHDR* pNMHDR, LRESULT* pResult);
      afx_msg void OnReleasedcaptureSliderRobotspeed(NMHDR* pNMHDR, LRESULT* pResult)
;
      afx_msg void OnCheckSonar();
      afx_msg void OnCheckBumper();
      //}}AFX_MSG
      DECLARE_MESSAGE_MAP()
};

//{{AFX_INSERT_LOCATION}}
// Microsoft Visual C++ will insert additional declarations immediately before the pr
evious line.
#endif                              // !defined(AFX_FANGDLG_H__BE51AAE4_63A8_4C63_B4D5_7
7E48D74057D__INCLUDED_)
```

```
// $Id: fangDlg.cpp,v 1.50 2002/04/30 20:16:47 el9d Exp $
// Flancrest Enterprises, Group 22

#include "stdafx.h"
#include "fang.h"
#include "fangDlg.h"
#include "cmd.h"
#include "robot.h"
#include "errors.h"
#include "math.h"
#include "joystick.h"
#include "ConfigDlg.h"

#ifdef _DEBUG
#define new DEBUG_NEW
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
#endif

UINT startJoystick(LPVOID param);
/////////////////////////////////////////////////////////////////////////////
// CAboutDlg dialog used for App About

class CAboutDlg : public CDialog
{
   public:
      CAboutDlg();

      // Dialog Data
      //{{AFX_DATA(CAboutDlg)
      enum { IDD = IDD_ABOUTBOX };
      //}}AFX_DATA

      // ClassWizard generated virtual function overrides
      //{{AFX_VIRTUAL(CAboutDlg)
   protected:
                                    // DDX/DDV support
      virtual void DoDataExchange(CDataExchange* pDX);
      //}}AFX_VIRTUAL

      // Implementation
   protected:
      //{{AFX_MSG(CAboutDlg)
      //}}AFX_MSG
      DECLARE_MESSAGE_MAP()
};

CAboutDlg::CAboutDlg() : CDialog(CAboutDlg::IDD)
{
   //{{AFX_DATA_INIT(CAboutDlg)
   //}}AFX_DATA_INIT
}


void CAboutDlg::DoDataExchange(CDataExchange* pDX)
{
   CDialog::DoDataExchange(pDX);
   //{{AFX_DATA_MAP(CAboutDlg)
   //}}AFX_DATA_MAP
}


BEGIN_MESSAGE_MAP(CAboutDlg, CDialog)
//{{AFX_MSG_MAP(CAboutDlg)
// No message handlers
//}}AFX_MSG_MAP
END_MESSAGE_MAP()

/////////////////////////////////////////////////////////////////////////////
// CFangDlg dialog

                                 /*=NULL*/
CFangDlg::CFangDlg(CWnd* pParent )
```



```
: CDialog(CFangDlg::IDD, pParent), robot(NULL), NUM_BUMPERS(6), NUM_SONARS(16),
MAX_SONAR_DIST(25), MIN_SONAR_DIST(6), className("CFangDlg"),
NUM_SPEEDS(4),MIN_PAN_RANGE(-90), MAX_PAN_RANGE(90), MIN_TILT_RANGE(-90),
MAX_TILT_RANGE(90), MAX_SPEED(400), MIN_SPEED(0), turnMult(4), sleepTime(20),
MAX_RATE(100), MIN_RATE(0), XCENTER(132), YCENTER(126)
{
   //{{AFX_DATA_INIT(CFangDlg)
   //}}AFX_DATA_INIT
   // Note that LoadIcon does not require a subsequent DestroyIcon in Win32
   m_hIcon = AfxGetApp()->LoadIcon(IDR_MAINFRAME);
}


void CFangDlg::DoDataExchange(CDataExchange* pDX)
{
   CDialog::DoDataExchange(pDX);
   //{{AFX_DATA_MAP(CFangDlg)
   DDX_Control(pDX, IDC_SLIDER_ROBOTSPEED, m_robotSpeed);
   DDX_Control(pDX, IDC_SLIDER_CAMTILT, m_camTilt);
   DDX_Control(pDX, IDC_SLIDER_CAMPAN, m_camPan);
   //}}AFX_DATA_MAP
}


BEGIN_MESSAGE_MAP(CFangDlg, CDialog)
//{{AFX_MSG_MAP(CFangDlg)
ON_WM_SYSCOMMAND()
ON_WM_PAINT()
ON_WM_QUERYDRAGICON()
ON_BN_CLICKED(IDC_BUTTON_LEFT, OnButtonLeft)
ON_BN_CLICKED(IDC_BUTTON_FORWARD, OnButtonForward)
ON_BN_CLICKED(IDC_BUTTON_RIGHT, OnButtonRight)
ON_BN_CLICKED(IDC_BUTTON_STOP, OnButtonStop)
ON_BN_CLICKED(IDC_BUTTON_CAM_RIGHT, OnButtonCamRight)
ON_BN_CLICKED(IDC_BUTTON_CAM_UP, OnButtonCamUp)
ON_BN_CLICKED(IDC_BUTTON_CAM_DOWN, OnButtonCamDown)
ON_BN_CLICKED(IDC_BUTTON_CAM_LEFT, OnButtonCamLeft)
ON_BN_CLICKED(IDC_BUTTON_CONNECT, OnButtonConnect)
ON_BN_CLICKED(IDC_BUTTON_BACK, OnButtonBack)
ON_BN_CLICKED(IDC_BUTTON_HOME, OnButtonHome)
ON_BN_CLICKED(IDC_BUTTON_FWDRIGHT, OnButtonFwdright)
ON_BN_CLICKED(IDC_BUTTON_FWDLEFT, OnButtonFwdleft)
ON_BN_CLICKED(IDC_BUTTON_BACKLEFT, OnButtonBackleft)
ON_BN_CLICKED(IDC_BUTTON_BACKRIGHT, OnButtonBackright)
ON_BN_CLICKED(IDC_BUTTON_MODE, OnButtonMode)
ON_BN_CLICKED(IDC_BUTTON_RESET, OnButtonReset)
ON_NOTIFY(NM_RELEASEDCAPTURE, IDC_SLIDER_CAMPAN, OnReleasedcaptureSliderCampan)
ON_NOTIFY(NM_RELEASEDCAPTURE, IDC_SLIDER_CAMTILT, OnReleasedcaptureSliderCamtilt)
ON_NOTIFY(NM_RELEASEDCAPTURE, IDC_SLIDER_ROBOTSPEED, OnReleasedcaptureSliderRobotspee
d)
ON_BN_CLICKED(IDC_CHECK_SONAR, OnCheckSonar)
ON_BN_CLICKED(IDC_CHECK_BUMPER, OnCheckBumper)
ON_WM_GETDLGCODE()
//}}AFX_MSG_MAP
END_MESSAGE_MAP()

/////////////////////////////////////////////////////////////////////////////
// CFangDlg message handlers

BOOL CFangDlg::OnInitDialog()
{
   CDialog::OnInitDialog();

   // Add "About..." menu item to system menu.

   // IDM_ABOUTBOX must be in the system command range.
   ASSERT((IDM_ABOUTBOX & 0xFFF0) == IDM_ABOUTBOX);
   ASSERT(IDM_ABOUTBOX < 0xF000);
   ASSERT((IDM_JOYSTICK & 0xFFF0) == IDM_JOYSTICK);
   ASSERT(IDM_JOYSTICK < 0xF000);
   ASSERT((IDM_CONFIGBOX & 0xFFF0) == IDM_CONFIGBOX);
   ASSERT(IDM_CONFIGBOX < 0xF000);
```

```
   CMenu* pSysMenu = GetSystemMenu(FALSE);
   if (pSysMenu != NULL)
   {
      CString strAboutMenu;
      strAboutMenu.LoadString(IDS_CONFIGBOX);
      if (!strAboutMenu.IsEmpty())
      {
         pSysMenu->AppendMenu(MF_SEPARATOR);
         pSysMenu->AppendMenu(MF_STRING, IDM_CONFIGBOX, strAboutMenu);
      }
     strAboutMenu.LoadString(IDS_JOYSTICK);
      if (!strAboutMenu.IsEmpty())
      {
         pSysMenu->AppendMenu(MF_STRING, IDM_JOYSTICK, strAboutMenu);
      }
      strAboutMenu.LoadString(IDS_ABOUTBOX);
      if (!strAboutMenu.IsEmpty())
      {
         pSysMenu->AppendMenu(MF_SEPARATOR);
         pSysMenu->AppendMenu(MF_STRING, IDM_ABOUTBOX, strAboutMenu);
      }

   }

   // Set the icon for this dialog.  The framework does this automatically
   //  when the application's main window is not a dialog
   SetIcon(m_hIcon, TRUE);       // Set big icon
   SetIcon(m_hIcon, FALSE);      // Set small icon

   //Extra initialization here

   setup();

   // return TRUE  unless you set the focus to a control
   return FALSE;
}


void CFangDlg::OnSysCommand(UINT nID, LPARAM lParam)
{
   if ((nID & 0xFFF0) == IDM_ABOUTBOX)
   {
      CAboutDlg dlgAbout;
      dlgAbout.DoModal();
   }
   else if ((nID & 0xFFF0) == IDM_CONFIGBOX)
   {
      CConfigDlg dlgConfig;
      dlgConfig.setfangDlg(this);
      dlgConfig.DoModal();
   }
   else if ((nID & 0xFFF0) == IDM_JOYSTICK)
   {
      CWinThread *joystickThread = AfxBeginThread(startJoystick, (LPVOID)this);
   }
   else
   {
      CDialog::OnSysCommand(nID, lParam);
   }
}


// If you add a minimize button to your dialog, you will need the code below
//  to draw the icon.  For MFC applications using the document/view model,
//  this is automatically done for you by the framework.

void CFangDlg::OnPaint()
{
   //sets default button to "connect"
   SetDefID(IDC_BUTTON_CONNECT);

   if (IsIconic())
   {
```



```
      CPaintDC dc(this);         // device context for painting

      SendMessage(WM_ICONERASEBKGND, (WPARAM) dc.GetSafeHdc(), 0);

      // Center icon in client rectangle
      int cxIcon = GetSystemMetrics(SM_CXICON);
      int cyIcon = GetSystemMetrics(SM_CYICON);
      CRect rect;
      GetClientRect(&rect);
      int x = (rect.Width() - cxIcon + 1) / 2;
      int y = (rect.Height() - cyIcon + 1) / 2;

      // Draw the icon
      dc.DrawIcon(x, y, m_hIcon);
   }
   else
   {
      CDialog::OnPaint();
   }

   //bitmap drawing of robot, bump sensors, etc

   //data variables for drawing bitmaps
   CClientDC DC(this);
   CBitmap bit;
   CBitmap *pbit;
   CDC memDC;
   BITMAP *pbm = new BITMAP;
   int i;

   //initialization
   memDC.CreateCompatibleDC(&DC);

   bit.LoadBitmap(MAKEINTRESOURCE(IDB_BITMAP_ROBOT));
   bit.GetBitmap(pbm);
   memDC.SelectObject(&bit);
   DC.BitBlt(XCENTER - pbm->bmWidth/2, YCENTER - pbm->bmHeight/2,
      pbm->bmWidth, pbm->bmHeight, &memDC, 0, 0, SRCAND);
   pbit= memDC.SelectObject(&bit);
   bit.DeleteObject();
   memDC.SelectObject(pbit);

   for(i = 0; i < NUM_BUMPERS; i++)
   {
      if(bumper_data[i] == true)
      {
         switch(i)
         {
            case 0: bit.LoadBitmap(MAKEINTRESOURCE(IDB_BITMAP_BUMPON0));
            break;
            case 1: bit.LoadBitmap(MAKEINTRESOURCE(IDB_BITMAP_BUMPON1));
            break;
            case 2: bit.LoadBitmap(MAKEINTRESOURCE(IDB_BITMAP_BUMPON2));
            break;
            case 3: bit.LoadBitmap(MAKEINTRESOURCE(IDB_BITMAP_BUMPON3));
            break;
            case 4: bit.LoadBitmap(MAKEINTRESOURCE(IDB_BITMAP_BUMPON4));
            break;
            case 5: bit.LoadBitmap(MAKEINTRESOURCE(IDB_BITMAP_BUMPON5));
            break;
            default:
               break;
         }
         bit.GetBitmap(pbm);
         memDC.SelectObject(&bit);
         DC.BitBlt(XCENTER - pbm->bmWidth/2, YCENTER - pbm->bmHeight/2,
            pbm->bmWidth, pbm->bmHeight, &memDC, 0, 0, SRCAND);
         pbit= memDC.SelectObject(&bit);
         bit.DeleteObject();
         memDC.SelectObject(pbit);
      }
      else
      {
```

```
         switch(i)
         {
            case 0: bit.LoadBitmap(MAKEINTRESOURCE(IDB_BITMAP_BUMPOFF0));
            break;
            case 1: bit.LoadBitmap(MAKEINTRESOURCE(IDB_BITMAP_BUMPOFF1));
            break;
            case 2: bit.LoadBitmap(MAKEINTRESOURCE(IDB_BITMAP_BUMPOFF2));
            break;
            case 3: bit.LoadBitmap(MAKEINTRESOURCE(IDB_BITMAP_BUMPOFF3));
            break;
            case 4: bit.LoadBitmap(MAKEINTRESOURCE(IDB_BITMAP_BUMPOFF4));
            break;
            case 5: bit.LoadBitmap(MAKEINTRESOURCE(IDB_BITMAP_BUMPOFF5));
            break;
            default:
               break;
         }
         bit.GetBitmap(pbm);
         memDC.SelectObject(&bit);
         DC.BitBlt(XCENTER - pbm->bmWidth/2, YCENTER - pbm->bmHeight/2,
            pbm->bmWidth, pbm->bmHeight, &memDC, 0, 0, SRCAND);
         pbit= memDC.SelectObject(&bit);
         bit.DeleteObject();
         memDC.SelectObject(pbit);
      }
   }

   //draw sonar lines
   CPoint homePoint, startPoint, endPoint, offset;
   homePoint.x = XCENTER;
   homePoint.y = YCENTER;
   //5 is so that it doesn't draw right on top of bumper
   offset.x = pbm->bmWidth / 2 + 5;
   offset.y = pbm->bmHeight / 2 + 5;
   double currDegrees = 270;
   const double pi = 3.14159;
   const double d2r = pi / 180;

   CPoint points[5];
   CPoint currPoint, bot;
   bot.x = 5; bot.y = 5;

   for(i = 0; i < NUM_SONARS; i++)
   {
      currPoint = homePoint + CPoint(offset.x * cos(currDegrees * d2r),
         offset.y * sin(currDegrees * d2r));
      points[0] = currPoint;

      currPoint += CPoint(sin(currDegrees * d2r) *-1* bot.x, cos(currDegrees * d2r) *
 bot.y);
      points[1] = currPoint;

      currPoint += CPoint(scale(sonar_data[i]) * cos(currDegrees * d2r),
         scale(sonar_data[i]) * sin(currDegrees * d2r));
      points[2] = currPoint;

      currPoint -= CPoint(sin(currDegrees * d2r) *-2* bot.x, cos(currDegrees * d2r) *
2* bot.y);
      points[3] = currPoint;

      currPoint -= CPoint(scale(sonar_data[i]) * cos(currDegrees * d2r),
         scale(sonar_data[i]) * sin(currDegrees * d2r));
      points[4] = currPoint;

      DC.Polygon(points, 5);

      currDegrees -= 22.5;
   }

   delete pbm;
}
```



```
// The system calls this to obtain the cursor to display while the user drags
//  the minimized window.
HCURSOR CFangDlg::OnQueryDragIcon()
{
   return (HCURSOR) m_hIcon;
}


bool CFangDlg::setRobot(Robot *robot)
{
   this->robot = robot;
   return true;
}


void CFangDlg::update(Cmd command)
{
   const string fnName = "update";
   vector<string> args = command.getArgs();
   string cmd = command.getCommand();
   int i;
   vector<bool> sonars_changed;
   for(i = 0; i < NUM_SONARS; i++)
      sonars_changed.push_back(false);
   bool bumpers_changed;
   bool is_changed = false;

   //heartbeat cmd
   if(cmd == "heartBeat")
   {
      if(args[0] == "Good")
      {
         heartbeat = true;
         SetDlgItemText(IDC_STATIC_HEARTBEAT,"Receiving Heartbeat");
      }
      else if(args[0] == "Lost")
      {
         heartbeat = false;
         SetDlgItemText(IDC_STATIC_HEARTBEAT,"Heartbeat Lost");
         fangError(className, fnName, "Heartbeat lost");
      }
      else
         fangError(className, fnName, "heartbeat argument error");
   }

   //bpData cmd
   else if(cmd == "bpData")
   {
      if(args.size() != NUM_BUMPERS)
      {
         fangError(className, fnName, "bpData argument error");
         return;
      }
      for(i = 0; i < NUM_BUMPERS; i++)
      {
         if(int(bumper_data[i]) == string2int(args[i]))
         {
            //do nothing
         }
         else
         {
          if(string2int(args[i]) == 1)
               bumper_data[i] = true;
          else
               bumper_data[i] = false;
             is_changed = true;
          bumpers_changed = true;
         }
      }
   }

   //snData cmd
   else if(cmd == "snData")
```

```cpp
   {
      if(args.size() != NUM_SONARS)
      {
         fangError(className, fnName, "snData argument error");
         return;
      }
      for(i = 0; i < NUM_SONARS; i++)
      {
         if(string2int(args[i]) > MAX_SONAR_DIST)
         {
            args[i] = int2string(MAX_SONAR_DIST);
         }
         else if(string2int(args[i]) < MIN_SONAR_DIST)
         {
            args[i] = int2string(MIN_SONAR_DIST);
         }
      }
      for(i = 0; i < NUM_SONARS; i++)
      {
         if(sonar_data[i] == string2int(args[i]))
         {
            //do nothing
         }
         else
         {
            sonar_data[i] = string2int(args[i]);
         sonars_changed[i] = true;
            is_changed = true;
         }
      }
   }

   //spData cmd
   else if(cmd == "spData")
   {
      if(args.size() != speed.size())
      {
         fangError(className, fnName, "spData argument error");
         return;
      }
      for(i = 0; i < speed.size(); i++)
      {
         if(speed[i] == string2int(args[i]))
         {
            //do nothing
         }
         else
         {
            speed[i] = string2int(args[i]);
            is_changed = true;
         }
      }
   }

   //ack cmd
   else if(cmd == "ack")
   {
      if(args.size() != 1)
      {
         fangError(className, fnName, "ack argument error");
         return;
      }
      if(args[0] == "da" || args[0] == "pr" || args[0] == "rotateRobot"
         || args[0] == "sp" || args[0] == "st" || args[0] == "vm"
         || args[0] == "setStopOnBump"|| args[0] == "reset")
         robot_busy = false;
      else if(args[0] == "moveCamera" || args[0] == "bumpCamera")
         camera_busy = false;
      is_changed = true;
   }

   //bump cmd
   else if(cmd == "bump")
```



```cpp
   {
      //protocol spec changed - this command is no longer used
      //is_changed = true;
      //bumper_data[string2int(args[0])] = true;
   }

   //speedMax cmd
   else if(cmd == "speedMax")
   {

   }

   //tiltMax cmd
   else if(cmd == "tiltMax")
   {

   }

   //panMax cmd
   else if(cmd == "panMax")
   {

   }

   //modify display if robot status is changed
   if(is_changed == true)
      PrePaint(sonars_changed, bumpers_changed);
}


void CFangDlg::setup()
{
   //initializes data members
   int i;
   for(i = 0; i < NUM_BUMPERS; i++)
      bumper_data.push_back(false);
   for(i = 0; i < NUM_SONARS; i++)
      sonar_data.push_back(MAX_SONAR_DIST);
   for(i = 0; i < NUM_SPEEDS; i++)
      speed.push_back(0);
   mode = 1;
   connected = false;

   m_camPan.SetRange(MIN_PAN_RANGE, MAX_PAN_RANGE);
   //needed for some reason to get initial position drawn: 50 is arbitrary
   m_camPan.SetPos(50);
   m_camPan.SetPos((MIN_PAN_RANGE + MAX_PAN_RANGE) / 2);
   m_camPan.SetPageSize(15);
   currCamPan = m_camPan.GetPos();
   m_camTilt.SetRange(MIN_TILT_RANGE, MAX_TILT_RANGE);
   //needed for some reason to get initial position drawn: 50 is arbitrary
   m_camTilt.SetPos(50);
   m_camTilt.SetPos((MIN_TILT_RANGE + MAX_TILT_RANGE) / 2);
   m_camTilt.SetPageSize(15);
   currCamTilt = m_camTilt.GetPos();
   m_robotSpeed.SetRange(MIN_SPEED, MAX_SPEED);
   m_robotSpeed.SetPos((MAX_SPEED + MIN_SPEED) / 2);
   m_robotSpeed.SetPageSize(20);
   currSpeed = m_robotSpeed.GetPos();
   //checks sonar and bumper checks
   CheckDlgButton(IDC_CHECK_SONAR, 1);
   CheckDlgButton(IDC_CHECK_BUMPER, 1);

   //gives server edit box initial focus
   GotoDlgCtrl(GetDlgItem(IDC_EDIT_SERVER_NAME));

   //creates a thread to poll the joystick
   CWinThread *joystickThread = AfxBeginThread(startJoystick, (LPVOID)this);

   bumpAngle = 15;
   bumpCamAngle = 15;
   bumpDistance = 12;
   heartbeatRate = 2;
```

```
   speedRate = 0;
   configRate = 0;
   sonarRate = 5;
   bumperRate = 5;
   stopOnBump = false;
   showDebugWindow = true;
   logIO = false;

   keyDown = 0;
}


void CFangDlg::OnButtonConnect()
{
   if(robot == NULL)
      return;
   CString text;
   GetDlgItemText(IDC_BUTTON_CONNECT, text);
   if(text == "Connect")
   {
      GetDlgItemText(IDC_EDIT_SERVER_NAME, text);
      if(robot->connect(string(text)))
      {
         SetDlgItemText(IDC_BUTTON_CONNECT,"Disconnect");
         connected = true;

       string commandName = "sp";
         vector<string> args;
         string rightWheelSpeedArg = int2string(currSpeed);
         string leftWheelSpeedArg  = int2string(currSpeed);
         args.push_back(rightWheelSpeedArg);
         args.push_back(leftWheelSpeedArg);
         robot->tell(Cmd(commandName, args));
       args.clear();

       if(IsDlgButtonChecked(IDC_CHECK_BUMPER)){
         OnCheckBumper();
       }
       if(IsDlgButtonChecked(IDC_CHECK_SONAR)){
         OnCheckSonar();
       }
       setheartbeatRate(heartbeatRate);
       StopOnBump(stopOnBump);
      }
      else
         MessageBox("Failed to connect to Remote Host",  "Connection Error");
   }
   else
   {
      robot->disconnect();
      SetDlgItemText(IDC_BUTTON_CONNECT,"Connect");
      SetDlgItemText(IDC_STATIC_HEARTBEAT,"Not Connected");
      connected = false;
   }

   if(IsDlgButtonChecked(IDC_CHECK_BUMPER))
      OnCheckBumper();
   if(IsDlgButtonChecked(IDC_CHECK_SONAR))
      OnCheckSonar();
}


int CFangDlg::string2int(string input)
{
   return atoi(input.c_str());
}


void CFangDlg::setMode(int mode)
{
   this->mode = mode;
   if(mode == 0)
      SetDlgItemText(IDC_STATIC_MODE,"Mode: Bump");
```



```
   else
      SetDlgItemText(IDC_STATIC_MODE,"Mode: Run");
}


int CFangDlg::getMode()
{
   return mode;
}


void CFangDlg::OnButtonHome()
{
   if(robot == NULL)
      return;
   currCamTilt = 0;
   currCamPan = 0;
   m_camTilt.SetPos((MIN_TILT_RANGE + MAX_TILT_RANGE) / 2);
   m_camPan.SetPos((MIN_PAN_RANGE + MAX_PAN_RANGE) / 2);

   string commandName = "moveCamera";
   vector<string> args;
   string panDegrees = int2string(currCamPan * 10);
   string tiltDegrees  = int2string(currCamTilt * 10);
   args.push_back(panDegrees);
   args.push_back(tiltDegrees);
   robot->tell(Cmd(commandName, args));
}


void CFangDlg::OnButtonForward()
{
   if(robot == NULL)
      return;
   string commandName;
   vector<string> args;

   if(mode == 0)
   {
      //tells robot to move forward bumpDistance inches
      commandName = "pr";
      string rightWheelStepArg = int2string(10 * bumpDistance);;
      string leftWheelStepArg  = int2string(10 * bumpDistance);;
      args.push_back(rightWheelStepArg);
      args.push_back(leftWheelStepArg);
   }
   else
   {
      //moves robot forward at current speed
      commandName = "vm";
      string rightWheelSpeedArg = int2string(currSpeed);
      string leftWheelSpeedArg  = int2string(currSpeed);
      args.push_back(rightWheelSpeedArg);
      args.push_back(leftWheelSpeedArg);
   }

   robot->tell(Cmd(commandName, args));
}


void CFangDlg::OnButtonBack()
{
   if(robot == NULL)
      return;
   string commandName;
   vector<string> args;

   if(mode == 0)
   {
      //tells robot to move backward bumpDistance inches
      commandName = "pr";
      string rightWheelStepArg = int2string(-10 * bumpDistance);
      string leftWheelStepArg  = int2string(-10 * bumpDistance);
```

```
      args.push_back(rightWheelStepArg);
      args.push_back(leftWheelStepArg);
   }
   else
   {
      //moves robot backward at current speed
      commandName = "vm";
      string rightWheelSpeedArg = int2string(-1 * currSpeed);
      string leftWheelSpeedArg  = int2string(-1 * currSpeed);
      args.push_back(rightWheelSpeedArg);
      args.push_back(leftWheelSpeedArg);
   }

   robot->tell(Cmd(commandName, args));
}


void CFangDlg::OnButtonRight()
{
   if(robot == NULL)
      return;
   string commandName;
   vector<string> args;

   if(mode == 0)
   {
      //rotates robot by bumpAngle amount
      commandName = "rotateRobot";
      string turnDegreesArg = int2string(10 * bumpAngle);
      args.push_back(turnDegreesArg);
   }
   else
   {
      //rotates robot right at current speed
      commandName = "vm";
      string rightWheelSpeedArg = int2string(-1 * currSpeed);
      string leftWheelSpeedArg  = int2string(currSpeed);
      args.push_back(rightWheelSpeedArg);
      args.push_back(leftWheelSpeedArg);
   }

   robot->tell(Cmd(commandName, args));
}


void CFangDlg::OnButtonLeft()
{
   if(robot == NULL)
      return;
   string commandName;
   vector<string> args;

   if(mode == 0)
   {
      //rotates robot by bumpAngle degrees
      commandName = "rotateRobot";
      string turnDegreesArg = int2string(-10 * bumpAngle);
      args.push_back(turnDegreesArg);
   }
   else
   {
      //rotates robot left at current speed
      commandName = "vm";
      string rightWheelSpeedArg = int2string(currSpeed);
      string leftWheelSpeedArg  = int2string(-1 * currSpeed);
      args.push_back(rightWheelSpeedArg);
      args.push_back(leftWheelSpeedArg);
   }

   robot->tell(Cmd(commandName, args));
}
```



```
void CFangDlg::OnButtonFwdright()
{
   if(robot == NULL)
      return;
   string commandName;
   vector<string> args;

   if(mode == 0)
   {
      //sets wheel speeds
      commandName = "sp";
      string rightWheelSpeedArg = int2string(currSpeed / turnMult);
      string leftWheelSpeedArg  = int2string(currSpeed);
      args.push_back(rightWheelSpeedArg);
      args.push_back(leftWheelSpeedArg);
      robot->tell(Cmd(commandName, args));
      args.clear();
      //tells robot to move forward and right ~bumpDistance inches
      commandName = "pr";
      string rightWheelStepArg = int2string(10 * bumpDistance / turnMult);
      string leftWheelStepArg  = int2string(10 * bumpDistance);
      args.push_back(rightWheelStepArg);
      args.push_back(leftWheelStepArg);
      robot->tell(Cmd(commandName, args));
   }
   else
   {
      //moves robot forward and right at current speed
      commandName = "vm";
      string rightWheelSpeedArg = int2string(currSpeed / turnMult);
      string leftWheelSpeedArg  = int2string(currSpeed);
      args.push_back(rightWheelSpeedArg);
      args.push_back(leftWheelSpeedArg);
      robot->tell(Cmd(commandName, args));
   }
}


void CFangDlg::OnButtonFwdleft()
{
   if(robot == NULL)
      return;
   string commandName;
   vector<string> args;

   if(mode == 0)
   {
      //sets wheel speeds
      commandName = "sp";
      string rightWheelSpeedArg = int2string(currSpeed);
      string leftWheelSpeedArg  = int2string(currSpeed / turnMult);
      args.push_back(rightWheelSpeedArg);
      args.push_back(leftWheelSpeedArg);
      robot->tell(Cmd(commandName, args));
      args.clear();
      //tells robot to move forward and left ~bumpDistance inches
      commandName = "pr";
      string rightWheelStepArg = int2string(10 * bumpDistance);
      string leftWheelStepArg  = int2string(10 * bumpDistance / turnMult);
      args.push_back(rightWheelStepArg);
      args.push_back(leftWheelStepArg);
      robot->tell(Cmd(commandName, args));
   }
   else
   {
      //moves robot forward and left at current speed
      commandName = "vm";
      string rightWheelSpeedArg = int2string(currSpeed);
      string leftWheelSpeedArg  = int2string(currSpeed / turnMult);
      args.push_back(rightWheelSpeedArg);
      args.push_back(leftWheelSpeedArg);
      robot->tell(Cmd(commandName, args));
   }
```

```
}


void CFangDlg::OnButtonBackleft()
{
   if(robot == NULL)
      return;
   string commandName;
   vector<string> args;

   if(mode == 0)
   {
      //sets wheel speeds
      commandName = "sp";
      string rightWheelSpeedArg = int2string(currSpeed);
      string leftWheelSpeedArg  = int2string(currSpeed / turnMult);
      args.push_back(rightWheelSpeedArg);
      args.push_back(leftWheelSpeedArg);
      robot->tell(Cmd(commandName, args));
      args.clear();
      //tells robot to move back and left ~bumpDistance inches
      commandName = "pr";
      string rightWheelStepArg = int2string(-10 * bumpDistance);
      string leftWheelStepArg  = int2string(-10 * bumpDistance / turnMult);
      args.push_back(rightWheelStepArg);
      args.push_back(leftWheelStepArg);
      robot->tell(Cmd(commandName, args));
   }
   else
   {
      //moves robot back and left at current speed
      commandName = "vm";
      string rightWheelSpeedArg = int2string(-1 * currSpeed);
      string leftWheelSpeedArg  = int2string(-1 * currSpeed / turnMult);
      args.push_back(rightWheelSpeedArg);
      args.push_back(leftWheelSpeedArg);
      robot->tell(Cmd(commandName, args));
   }
}


void CFangDlg::OnButtonBackright()
{
   if(robot == NULL)
      return;
   string commandName;
   vector<string> args;

   if(mode == 0)
   {
      //sets wheel speeds
      commandName = "sp";
      string rightWheelSpeedArg = int2string(currSpeed / turnMult);
      string leftWheelSpeedArg  = int2string(currSpeed);
      args.push_back(rightWheelSpeedArg);
      args.push_back(leftWheelSpeedArg);
      robot->tell(Cmd(commandName, args));
      args.clear();
      //tells robot to move back and right ~bumpDistance inches
      commandName = "pr";
      string rightWheelStepArg = int2string(-10 * bumpDistance / turnMult);
      string leftWheelStepArg  = int2string(-10 * bumpDistance);
      args.push_back(rightWheelStepArg);
      args.push_back(leftWheelStepArg);
      robot->tell(Cmd(commandName, args));
   }
   else
   {
      //moves robot back and right at current speed
      commandName = "vm";
      string rightWheelSpeedArg = int2string(-1 * currSpeed / turnMult);
      string leftWheelSpeedArg  = int2string(-1 * currSpeed);
      args.push_back(rightWheelSpeedArg);
```



```
      args.push_back(leftWheelSpeedArg);
      robot->tell(Cmd(commandName, args));
   }
}


void CFangDlg::OnButtonStop()
{
   if(robot == NULL)
      return;
   string commandName = "st";
   vector<string> args;
   string robotArg = "1";
   string cameraArg = "0";
   args.push_back(robotArg);
   args.push_back(cameraArg);
   robot->tell(Cmd(commandName, args));
}


//bumps camera bumpCamAngle degrees up
void CFangDlg::OnButtonCamUp()
{
   if(robot == NULL)
      return;
   if(currCamTilt >= MAX_TILT_RANGE - bumpCamAngle)
      return;
   string commandName = "bumpCamera";
   vector<string> args;
   string panDistanceArg = "0";
   string tiltDistanceArg = int2string(10 * bumpCamAngle);
   args.push_back(panDistanceArg);
   args.push_back(tiltDistanceArg);
   robot->tell(Cmd(commandName, args));
   //updates internal data and display
   currCamTilt += bumpCamAngle;
   m_camTilt.SetPos(-1 * currCamTilt);
}


//bumps camera bumpCamAngle degrees down
void CFangDlg::OnButtonCamDown()
{
   if(robot == NULL)
      return;
   if(currCamTilt <= MIN_TILT_RANGE + bumpCamAngle)
      return;
   string commandName = "bumpCamera";
   vector<string> args;
   string panDistanceArg = "0";
   string tiltDistanceArg = int2string(-10 * bumpCamAngle);
   args.push_back(panDistanceArg);
   args.push_back(tiltDistanceArg);
   robot->tell(Cmd(commandName, args));
   //updates internal data and display
   currCamTilt -= bumpCamAngle;
   m_camTilt.SetPos(-1 * currCamTilt);
}


//bumps camera bumpCamAngle degrees right
void CFangDlg::OnButtonCamRight()
{
   if(robot == NULL)
      return;
   if(currCamPan >= MAX_PAN_RANGE - bumpCamAngle)
      return;
   string commandName = "bumpCamera";
   vector<string> args;
   string panDistanceArg = int2string(-10 * bumpCamAngle);
   string tiltDistanceArg = "0";
   args.push_back(panDistanceArg);
   args.push_back(tiltDistanceArg);
```

```
   robot->tell(Cmd(commandName, args));
   //updates internal data and display
   currCamPan += bumpCamAngle;
   m_camPan.SetPos(currCamPan);
}


//bumps camera bumpCamAngle degrees left
void CFangDlg::OnButtonCamLeft()
{
   if(robot == NULL)
      return;
   if(currCamPan <= MIN_PAN_RANGE + bumpCamAngle)
      return;
   string commandName = "bumpCamera";
   vector<string> args;
   string panDistanceArg = int2string(10 * bumpCamAngle);
   string tiltDistanceArg = "0";
   args.push_back(panDistanceArg);
   args.push_back(tiltDistanceArg);
   robot->tell(Cmd(commandName, args));
   //updates internal data and display
   currCamPan -= bumpCamAngle;
   m_camPan.SetPos(currCamPan);
}


//switches between bump (0) and run (1)
void CFangDlg::OnButtonMode()
{
   if(mode == 0)
      setMode(1);
   else
      setMode(0);
}


void CFangDlg::OnButtonReset()
{
   if(robot == NULL)
      return;
   string commandName = "reset";
   vector<string> args;
   robot->tell(Cmd(commandName, args));
}


void CFangDlg::OnReleasedcaptureSliderCampan(NMHDR* pNMHDR, LRESULT* pResult)
{
   if(robot == NULL)
      return;
   int nPos = m_camPan.GetPos();

   currCamPan = nPos;

   string commandName = "moveCamera";
   vector<string> args;
   string panDegrees = int2string(-1 * currCamPan * 10);
   string tiltDegrees  = int2string(-1 * currCamTilt * 10);
   args.push_back(panDegrees);
   args.push_back(tiltDegrees);
   robot->tell(Cmd(commandName, args));

   *pResult = 0;
}


void CFangDlg::OnReleasedcaptureSliderCamtilt(NMHDR* pNMHDR, LRESULT* pResult)
{
   if(robot == NULL)
      return;
   int nPos = m_camTilt.GetPos();
```



```
   currCamTilt = nPos;

   string commandName = "moveCamera";
   vector<string> args;
   string panDegrees = int2string(-1 * currCamPan * 10);
   string tiltDegrees  = int2string(-1 * currCamTilt * 10);
   args.push_back(panDegrees);
   args.push_back(tiltDegrees);
   robot->tell(Cmd(commandName, args));

   *pResult = 0;
}


void CFangDlg::OnReleasedcaptureSliderRobotspeed(NMHDR* pNMHDR, LRESULT* pResult)
{
   if(robot == NULL)
      return;
   int nPos = m_robotSpeed.GetPos();

   currSpeed = nPos;

   string commandName = "sp";
   vector<string> args;
   string rightWheelSpeedArg = int2string(currSpeed);
   string leftWheelSpeedArg  = int2string(currSpeed);
   args.push_back(rightWheelSpeedArg);
   args.push_back(leftWheelSpeedArg);
   robot->tell(Cmd(commandName, args));

   *pResult = 0;
}


UINT startJoystick(LPVOID param)
{
   const string fnName = "startJoystick";

   if(NULL != param)
      return ((CFangDlg*)param)->pollJoystick();
   else
   {
      fangError("", fnName,
         "Received NULL pointer, not starting joystick thread");
      return -1;
   }
}


UINT CFangDlg::pollJoystick()
{
   if(robot == NULL)
      return 0;
   const string fnName = "pollJoystick";
   Joystick js;
   bool PollJoystick = true;
   bool acceptDigitalAxis;

   const unsigned int num_axes    = js.GetNumAxes();
   double *axes  = new double[num_axes];
   unsigned int returned_axes    = 0;
   const unsigned int num_buttons = js.GetNumButtons();
   bool *buttons = new bool[num_buttons];
   unsigned int returned_buttons = 0;

   enum AXISSTATE { UP, DOWN, NONE };

   AXISSTATE xState = NONE, yState = NONE;
   AXISSTATE prevxState = NONE, prevyState = NONE;
   AXISSTATE DigitalxState = NONE, DigitalyState = NONE;
   AXISSTATE prevDigitalxState = NONE, prevDigitalyState = NONE;

   vector<bool> currButtons;
```

```
   for(int i = 0; i < num_buttons; i++)
      currButtons.push_back(false);

   int xPos = 0, yPos = 0;
   int prevxPos = 0, prevyPos = 0;
   int xCamPos = 0, yCamPos = 0;
   int magnitude = 0;
   int turnMagnitude = 0;
   int xCamHome = 0, yCamHome = 0;
   double analogTurnMult = 1;
   double stickAngle = 0;
   const double pi = 3.14159;
   string commandName;
   vector<string> args;

   //disable retrying of joystick, enable if there is an error
   CMenu* pSysMenu = GetSystemMenu(FALSE);
   if(pSysMenu != NULL)
      pSysMenu->EnableMenuItem(IDM_JOYSTICK, MF_GRAYED);

   while (PollJoystick)
   {
      if (1 != js.Query(axes, num_axes, returned_axes, buttons, num_buttons,
         returned_buttons))
      {
         fangError(className, fnName, "Joystick query error");
       if(pSysMenu != NULL)
         pSysMenu->EnableMenuItem(IDM_JOYSTICK, MF_ENABLED);
       MessageBox("Joystick query error", "Joystick Error");
         return 1;
      }
      if(returned_axes < 3 || returned_buttons < 12)
      {
         fangError(className, fnName, "Joystick query error: use the AxisPad!");
       if(pSysMenu != NULL)
         pSysMenu->EnableMenuItem(IDM_JOYSTICK, MF_ENABLED);
       MessageBox("Joystick not compatible", "Joystick Error");
         return 1;
      }

      //sets x axis state according to joystick data
      if(int(axes[0]) > 5)
         xState = UP;
      else if(int(axes[0]) < -5)
         xState = DOWN;
      else
         xState = NONE;

      //sets y axis state according to joystick data (y is reversed)
      if(int(axes[1]) < -5)
         yState = UP;
      else if(int(axes[1]) > 5)
         yState = DOWN;
      else
         yState = NONE;

      //only accept digital axis input if analog is not being used
      if(xState == NONE && yState == NONE && prevxState == NONE && prevyState == NONE
)
         acceptDigitalAxis = true;
      else
         acceptDigitalAxis = false;

//movement functions from left analog stick
      //sends movement command if state of first control stick has changed
      if(xPos == int(axes[0]) && yPos == int(axes[1]))
      {
         //do nothing
      }
      else
      {
         xPos = abs(int(axes[0]));
         yPos = abs(int(axes[1]));
```



```

         magnitude = sqrt(xPos * xPos + yPos * yPos);
         if(magnitude > 100)
            magnitude = 100;
         //converts magnitude from axis coords (-100, 100) to speed coords (-400, 400)
         magnitude *= 4;
         //scales magnitude according to current speed selected
         magnitude *= (currSpeed / double(MAX_SPEED));

         if(xPos != 0)
            stickAngle = atan(yPos / xPos);
         else
            stickAngle = pi / 2;
         analogTurnMult = 2 * stickAngle / pi;

         //for use only when there is no y motion
         turnMagnitude = xPos * 4 * currSpeed / double(MAX_SPEED);

           commandName = "vm";

           if(xState == NONE && yState == NONE)
           {
              commandName = "st";
              string robotArg = "1";
              string cameraArg = "0";
              args.push_back(robotArg);
              args.push_back(cameraArg);
           }
           else if(xState == UP && yState == NONE)
           {
              string rightWheelSpeedArg = int2string(-1 * turnMagnitude);
              string leftWheelSpeedArg = int2string(turnMagnitude);
              args.push_back(rightWheelSpeedArg);
              args.push_back(leftWheelSpeedArg);
           }
           else if(xState == DOWN && yState == NONE)
           {
              string rightWheelSpeedArg = int2string(turnMagnitude);
              string leftWheelSpeedArg = int2string(-1 * turnMagnitude);
              args.push_back(rightWheelSpeedArg);
              args.push_back(leftWheelSpeedArg);
           }
           else if(xState == NONE && yState == UP)
           {
              string rightWheelSpeedArg = int2string(magnitude);
              string leftWheelSpeedArg = int2string(magnitude);
              args.push_back(rightWheelSpeedArg);
              args.push_back(leftWheelSpeedArg);
           }
           else if(xState == NONE && yState == DOWN)
           {
              string rightWheelSpeedArg = int2string(-1 * magnitude);
              string leftWheelSpeedArg = int2string(-1 * magnitude);
              args.push_back(rightWheelSpeedArg);
              args.push_back(leftWheelSpeedArg);
           }
           else if(xState == UP && yState == UP)
           {
              string rightWheelSpeedArg = int2string(int(magnitude * analogTurnMult));
              string leftWheelSpeedArg = int2string(magnitude);
              args.push_back(rightWheelSpeedArg);
              args.push_back(leftWheelSpeedArg);
           }
           else if(xState == UP && yState == DOWN)
           {
              string rightWheelSpeedArg = int2string(int(-1 * magnitude * analogTurnMul
t));
              string leftWheelSpeedArg = int2string(-1 * magnitude);
              args.push_back(rightWheelSpeedArg);
              args.push_back(leftWheelSpeedArg);
           }
           else if(xState == DOWN && yState == UP)
           {
```

```
            string rightWheelSpeedArg = int2string(magnitude);
            string leftWheelSpeedArg = int2string(int(magnitude * analogTurnMult));
            args.push_back(rightWheelSpeedArg);
            args.push_back(leftWheelSpeedArg);
         }
         else if(xState == DOWN && yState == DOWN)
         {
            string rightWheelSpeedArg = int2string(-1 * magnitude);
            string leftWheelSpeedArg = int2string(int(-1 * magnitude * analogTurnMult
));
            args.push_back(rightWheelSpeedArg);
            args.push_back(leftWheelSpeedArg);
         }
         //only send command if command has changed since last command sent
         if(xPos != prevxPos || yPos != prevyPos)
         {
            prevxPos = xPos;
         prevyPos = yPos;
            robot->tell(Cmd(commandName, args));
         }
         args.clear();
      }

//camera functions from right analog stick
      //sends camera command if state of second control stick has changed
      if(xCamPos == int(axes[2]) && yCamPos == int(axes[3]))
      {
         //do nothing
      }
      else
      {
         xCamPos = int(axes[2]);
         yCamPos = int(axes[3]);

         //variables to control actual camera pan and tilt (in 1/10 degrees)
         int xCamMove = xCamPos * MAX_PAN_RANGE / 10.0 + xCamHome * 10;
         int yCamMove = yCamPos * MAX_TILT_RANGE / 10.0 - yCamHome * 10;

         //check for out-of-bounds argument
         if(xCamMove > MAX_PAN_RANGE * 10)
            xCamMove = MAX_PAN_RANGE * 10;
         else if(xCamMove < MIN_PAN_RANGE * 10)
            xCamMove = MIN_PAN_RANGE * 10;
         if(yCamMove > MAX_TILT_RANGE * 10)
            yCamMove = MAX_TILT_RANGE * 10;
         else if(yCamMove < MIN_TILT_RANGE * 10)
            yCamMove = MIN_TILT_RANGE * 10;

         currCamPan = xCamMove / 10;
         currCamTilt = -1 * yCamMove / 10;
         m_camPan.SetPos(currCamPan);
         m_camTilt.SetPos(-1 * currCamTilt);

         commandName = "moveCamera";
         string panDegrees = int2string(-1 * xCamMove);
         string tiltDegrees = int2string(-1 * yCamMove);
         args.push_back(panDegrees);
         args.push_back(tiltDegrees);

         robot->tell(Cmd(commandName, args));
         args.clear();
      }

//basic movement functions from digital pad
      //only accepts digital input if analog is not being used
      if(acceptDigitalAxis)
      {
         //sets x axis state according to joystick data
         if(int(axes[4]) > 5)
            DigitalxState = UP;
         else if(int(axes[4]) < -5)
            DigitalxState = DOWN;
         else
```



```
            DigitalxState = NONE;

         //sets y axis state according to joystick data (y is reversed)
         if(int(axes[5]) < -5)
            DigitalyState = UP;
         else if(int(axes[5]) > 5)
            DigitalyState = DOWN;
         else
            DigitalyState = NONE;

         //only accepts input from one axis at a time
         if(DigitalxState != prevDigitalxState && prevDigitalyState == NONE)
         {
            if(DigitalxState == UP)
               OnButtonRight();
            else if(DigitalxState == DOWN)
               OnButtonLeft();
            else
               OnButtonStop();
            prevDigitalxState = DigitalxState;
         }
         else if(DigitalyState != prevDigitalyState && prevDigitalxState == NONE)
         {
            if(DigitalyState == UP)
               OnButtonForward();
            else if(DigitalyState == DOWN)
               OnButtonBack();
            else
               OnButtonStop();
            prevDigitalyState = DigitalyState;
         }
      }

//button functions
      //button 1 = bump cam left
      if(buttons[0] && xCamHome >= MIN_PAN_RANGE + bumpCamAngle && !currButtons[0])
      {
         xCamHome -= bumpCamAngle;
         commandName = "bumpCamera";
         string panDistanceArg = int2string(10 * bumpCamAngle);
         string tiltDistanceArg = "0";
         args.push_back(panDistanceArg);
         args.push_back(tiltDistanceArg);
         robot->tell(Cmd(commandName, args));
         args.clear();
         currCamPan -= bumpCamAngle;
         m_camPan.SetPos(currCamPan);
         currButtons[0] = true;
      }
      else if (!buttons[0] && currButtons[0])
         currButtons[0] = false;
     //button 2 = bump cam up
      if(buttons[1] && yCamHome <= MAX_TILT_RANGE - bumpCamAngle && !currButtons[1])
      {
         yCamHome += bumpCamAngle;
         commandName = "bumpCamera";
         string panDistanceArg = "0";
         string tiltDistanceArg = int2string(10 * bumpCamAngle);
         args.push_back(panDistanceArg);
         args.push_back(tiltDistanceArg);
         robot->tell(Cmd(commandName, args));
         args.clear();
         currCamTilt += bumpCamAngle;
         m_camTilt.SetPos(-1 * currCamTilt);
         currButtons[1] = true;
      }
      else if (!buttons[1] && currButtons[1])
         currButtons[1] = false;
      //button 3 = bump cam down
      if(buttons[2] && yCamHome >= MIN_TILT_RANGE + bumpCamAngle && !currButtons[2])
      {
         yCamHome -= bumpCamAngle;
         commandName = "bumpCamera";
```

```
        string panDistanceArg = "0";
        string tiltDistanceArg = int2string(-10 * bumpCamAngle);
        args.push_back(panDistanceArg);
        args.push_back(tiltDistanceArg);
        robot->tell(Cmd(commandName, args));
        args.clear();
        currCamTilt -= bumpCamAngle;
        m_camTilt.SetPos(-1 * currCamTilt);
        currButtons[2] = true;
     }
     else if (!buttons[2] && currButtons[2])
        currButtons[2] = false;
     //button 4 = bump cam right
     if(buttons[3] && xCamHome <= MAX_PAN_RANGE - bumpCamAngle && !currButtons[3])
     {
        xCamHome += bumpCamAngle;
        commandName = "bumpCamera";
        string panDistanceArg = int2string(-10 * bumpCamAngle);
        string tiltDistanceArg = "0";
        args.push_back(panDistanceArg);
        args.push_back(tiltDistanceArg);
        robot->tell(Cmd(commandName, args));
        args.clear();
        currCamPan += bumpCamAngle;
        m_camPan.SetPos(currCamPan);
        currButtons[3] = true;
     }
     else if (!buttons[3] && currButtons[3])
        currButtons[3] = false;
     //button 5 = set speed max
     if(buttons[4] && !currButtons[4])
     {
      currSpeed = MAX_SPEED;
        commandName = "sp";
        string rightWheelSpeedArg = int2string(currSpeed);
        string leftWheelSpeedArg  = int2string(currSpeed);
        args.push_back(rightWheelSpeedArg);
        args.push_back(leftWheelSpeedArg);
        robot->tell(Cmd(commandName, args));

        args.clear();
        m_robotSpeed.SetPos(currSpeed);
        currButtons[4] = true;
     }
     else if (!buttons[4] && currButtons[4])
        currButtons[4] = false;
    //button 6 = set speed 3/4
     if(buttons[5] && !currButtons[5])
     {
      currSpeed = 3 * (MAX_SPEED + MIN_SPEED) / 4;
        commandName = "sp";
        string rightWheelSpeedArg = int2string(currSpeed);
        string leftWheelSpeedArg  = int2string(currSpeed);
        args.push_back(rightWheelSpeedArg);
        args.push_back(leftWheelSpeedArg);
        robot->tell(Cmd(commandName, args));

        args.clear();
        m_robotSpeed.SetPos(currSpeed);
        currButtons[5] = true;
     }
     else if (!buttons[5] && currButtons[5])
        currButtons[5] = false;
     //button 7 = set speed 1/2
     if(buttons[6] && !currButtons[6])
     {
      currSpeed = (MAX_SPEED + MIN_SPEED) / 2;
        commandName = "sp";
        string rightWheelSpeedArg = int2string(currSpeed);
        string leftWheelSpeedArg  = int2string(currSpeed);
        args.push_back(rightWheelSpeedArg);
        args.push_back(leftWheelSpeedArg);
        robot->tell(Cmd(commandName, args));
```



```
        args.clear();
        m_robotSpeed.SetPos(currSpeed);
        currButtons[6] = true;
     }
     else if (!buttons[6] && currButtons[6])
        currButtons[6] = false;
    //button 8 = set speed 1/4
     if(buttons[7] && !currButtons[7])
     {
      currSpeed = (MAX_SPEED + MIN_SPEED) / 4;
        commandName = "sp";
        string rightWheelSpeedArg = int2string(currSpeed);
        string leftWheelSpeedArg  = int2string(currSpeed);
        args.push_back(rightWheelSpeedArg);
        args.push_back(leftWheelSpeedArg);
        robot->tell(Cmd(commandName, args));

        args.clear();
        m_robotSpeed.SetPos(currSpeed);
        currButtons[7] = true;
     }
     else if (!buttons[7] && currButtons[7])
        currButtons[7] = false;
     //button 11 = esc = stop robot, reset camera and speed
     if(buttons[10] && !currButtons[10])
     {
        commandName = "st";
        string robotArg = "1";
        string cameraArg = "1";
        args.push_back(robotArg);
        args.push_back(cameraArg);
        robot->tell(Cmd(commandName, args));
        args.clear();

        OnButtonHome();

        m_robotSpeed.SetPos((MAX_SPEED + MIN_SPEED) / 2);
        currSpeed = m_robotSpeed.GetPos();
        xCamHome = 0;
        yCamHome = 0;

        currButtons[10] = true;
     }
     else if (!buttons[10] && currButtons[10])
        currButtons[10] = false;
     //button 12 = enter = reset from bump
     if(buttons[11] && !currButtons[11])
     {
        commandName = "reset";
        robot->tell(Cmd(commandName, args));

        currButtons[11] = true;
     }
     else if(!buttons[11] && currButtons[11])
        currButtons[11] = false;

     Sleep(sleepTime);
  }

  delete [] axes;
  delete [] buttons;

  return 0;
}


void CFangDlg::OnCheckSonar()
{
  if(robot == NULL)
     return;
  int n = IsDlgButtonChecked(IDC_CHECK_SONAR);
```

```
   vector<string> args;
   string commandName = "getSn";
   string periodArg;
   double tempRate = 1.0 / sonarRate;
   tempRate *= 100;

   if(n != 0)
   {
      periodArg = int2string(int(tempRate));
      args.push_back(periodArg);
   }
   else
   {
      periodArg = int2string(0);
      args.push_back(periodArg);
   }

   robot->tell(Cmd(commandName, args));
}


void CFangDlg::OnCheckBumper()
{
   if(robot == NULL)
      return;
   int n = IsDlgButtonChecked(IDC_CHECK_BUMPER);

   vector<string> args;
   string commandName = "getBp";
   string periodArg;
   double tempRate = 1.0 / bumperRate;
   tempRate *= 100;

   if(n != 0)
   {
      periodArg = int2string(int(tempRate));
      args.push_back(periodArg);
   }
   else
   {
      periodArg = int2string(0);
      args.push_back(periodArg);
   }

   robot->tell(Cmd(commandName, args));
}


double CFangDlg::scale(double input)
{
   return (input * 2);
}


BOOL CFangDlg::PreTranslateMessage(MSG* pMsg)
{
   if(robot == NULL)
      return CDialog::PreTranslateMessage(pMsg);
   string commandName;
   vector<string> args;
   string param1, param2;

   //handle keys that map to buttons
   if(pMsg->message == WM_KEYDOWN && connected)
   {
      //only ignore repeated commands if mode is run
      if(keyDown == pMsg->wParam && mode == 1)
         return TRUE;

      switch(pMsg->wParam)
      {
         //movement keys
         case VK_NUMPAD1: OnButtonBackleft();
```



```
         keyDown = pMsg->wParam;
         return TRUE;
         break;
         case VK_NUMPAD2: OnButtonBack();
         keyDown = pMsg->wParam;
         return TRUE;
         break;
         case VK_NUMPAD3: OnButtonBackright();
         keyDown = pMsg->wParam;
         return TRUE;
         break;
         case VK_NUMPAD4: OnButtonLeft();
         keyDown = pMsg->wParam;
         return TRUE;
         break;
         case VK_NUMPAD5: OnButtonStop();
         return TRUE;
         break;
         case VK_NUMPAD6: OnButtonRight();
         keyDown = pMsg->wParam;
         return TRUE;
         break;
         case VK_NUMPAD7: OnButtonFwdleft();
         keyDown = pMsg->wParam;
         return TRUE;
         break;
         case VK_NUMPAD8: OnButtonForward();
         keyDown = pMsg->wParam;
         return TRUE;
         break;
         case VK_NUMPAD9: OnButtonFwdright();
         keyDown = pMsg->wParam;
         return TRUE;
         break;
         //camera keys
                                 /*"A"*/
         case 0x41: OnButtonCamLeft();
         return TRUE;
         break;
                                 /*"D"*/
         case 0x44: OnButtonCamRight();
         return TRUE;
         break;
                                 /*"S"*/
         case 0x53: OnButtonCamDown();
         return TRUE;
         break;
                                 /*"W"*/
         case 0x57: OnButtonCamUp();
         return TRUE;
         break;
         //speed keys
         case VK_PRIOR:          //page up
            currSpeed += 20;
            if(currSpeed > MAX_SPEED)
               currSpeed = MAX_SPEED;
            m_robotSpeed.SetPos(currSpeed);

            commandName = "sp";
            param1 = int2string(currSpeed);
            param2  = int2string(currSpeed);
            args.push_back(param1);
            args.push_back(param2);
            robot->tell(Cmd(commandName, args));

            return TRUE;
            break;
         case VK_NEXT:           //page down
            currSpeed -= 20;
            if(currSpeed < MIN_SPEED)
               currSpeed = MIN_SPEED;
            m_robotSpeed.SetPos(currSpeed);
```

```
            commandName = "sp";
            param1 = int2string(currSpeed);
            param2  = int2string(currSpeed);
            args.push_back(param1);
            args.push_back(param2);
            robot->tell(Cmd(commandName, args));

            return TRUE;
            break;
         default:
            break;
      }
   }
   else if(pMsg->message == WM_KEYUP && connected)
   {
      //only ignore repeated commands if mode is run
      if(keyDown == pMsg->wParam && mode == 1)
      {
         keyDown = 0;
         OnButtonStop();
         return TRUE;
      }
   }

   return CDialog::PreTranslateMessage(pMsg);
}


//////////////////////////set configuration data

void CFangDlg::ShowDebugWindow(bool show)
{
   showDebugWindow = show;
   fangShowConsole(show);
}


void CFangDlg::StopOnBump(bool stop)
{
   if(robot == NULL)
      return;
   stopOnBump = stop;

   string commandName = "setStopOnBump";
   vector<string> args;
   string param1;

   if(stop)
   {
      param1 = "1";
   }
   else
   {
      param1 = "0";
   }

   args.push_back(param1);
   robot->tell(Cmd(commandName, args));
}


bool CFangDlg::getStopOnBump()
{
   return stopOnBump;
}


bool CFangDlg::getShowDebugWindow()
{
   return showDebugWindow;
}
```



```
void CFangDlg::setbumpAngle(int angle)
{
   const string fnName = "setbumpAngle";
   const int MAX_BUMP = 180, MIN_BUMP = 0;

   if(angle < MAX_BUMP && angle > MIN_BUMP)
      bumpAngle = angle;
   else
      fangError(className, fnName, "bump angle out of bounds (" +
     int2string(MIN_BUMP) + "," + int2string(MAX_BUMP) + ")");
}


void CFangDlg::setbumpCamAngle(int angle)
{
   const string fnName = "setbumpCamAngle";
   int min = MAX_TILT_RANGE;
   if (min < MAX_PAN_RANGE)
      min = MAX_PAN_RANGE;
   const int MAX_BUMP = min, MIN_BUMP = 0;

   if(angle < MAX_BUMP && angle > MIN_BUMP)
      bumpCamAngle = angle;
   else
      fangError(className, fnName, "camera bump angle out of bounds (" +
     int2string(MIN_BUMP) + "," + int2string(MAX_BUMP) + ")");
}


void CFangDlg::setbumpDistance(int distance)
{
   const string fnName = "setbumpDistance";
   const int MAX_BUMP = 120, MIN_BUMP = 0;

   if(distance < MAX_BUMP && distance > MIN_BUMP)
      bumpDistance = distance;
   else
      fangError(className, fnName, "bump distance out of bounds (" +
     int2string(MIN_BUMP) + "," + int2string(MAX_BUMP) + ")");
}


void CFangDlg::setbumperRate(int rate)
{
   const string fnName = "setbumperRate";

   if(rate == bumperRate)
      return;
   if(rate >= MIN_RATE && rate <= MAX_RATE){
      bumperRate = rate;
      OnCheckBumper();
   }
   else
      fangError(className, fnName, "bumper refresh rate out of bounds (" +
         int2string(MIN_RATE) + "," + int2string(MAX_RATE) + ")");
}


void CFangDlg::setconfigRate(int rate)
{
   const string fnName = "setconfigRate";

   if(rate >= MIN_RATE && rate <= MAX_RATE)
      configRate = rate;
   else
      fangError(className, fnName, "config refresh rate out of bounds (" +
         int2string(MIN_RATE) + "," + int2string(MAX_RATE) + ")");
}


void CFangDlg::setheartbeatRate(int rate)
{
   if(robot == NULL)
```

```
        return;
    const string fnName = "setheartbeatRate";

    if(rate >= MIN_RATE && rate <= MAX_RATE){
        heartbeatRate = rate;
      string commandName = "setHeartBeat";
      vector<string> args;
      double tempRate = 1.0 / heartbeatRate;
      tempRate *= 100;
        string param1 = int2string(int(tempRate));
        args.push_back(param1);
      robot->tell(Cmd(commandName, args));
    }
    else
        fangError(className, fnName, "heartbeat rate out of bounds (" +
            int2string(MIN_RATE) + "," + int2string(MAX_RATE) + ")");
}


void CFangDlg::setsonarRate(int rate)
{
    const string fnName = "setsonarRate";

    if(rate >= MIN_RATE && rate <= MAX_RATE){
        sonarRate = rate;
      OnCheckSonar();
    }
    else
        fangError(className, fnName, "sonar refresh rate out of bounds (" +
            int2string(MIN_RATE) + "," + int2string(MAX_RATE) + ")");
}


void CFangDlg::setspeedRate(int rate)
{
    const string fnName = "setspeedRate";

    if(rate >= MIN_RATE && rate <= MAX_RATE)
        speedRate = rate;
    else
        fangError(className, fnName, "speed refresh rate out of bounds (" +
            int2string(MIN_RATE) + "," + int2string(MAX_RATE) + ")");
}


void CFangDlg::setbumpAngle(string angle)
{
    setbumpAngle(string2int(angle));
}


void CFangDlg::setbumpCamAngle(string angle)
{
    setbumpCamAngle(string2int(angle));
}


void CFangDlg::setbumpDistance(string distance)
{
    setbumpDistance(string2int(distance));
}


void CFangDlg::setbumperRate(string rate)
{
    setbumperRate(string2int(rate));
}


void CFangDlg::setconfigRate(string rate)
{
    setconfigRate(string2int(rate));
}
```



```

void CFangDlg::setheartbeatRate(string rate)
{
    setheartbeatRate(string2int(rate));
}


void CFangDlg::setsonarRate(string rate)
{
    setsonarRate(string2int(rate));
}


void CFangDlg::setspeedRate(string rate)
{
    setspeedRate(string2int(rate));
}


int CFangDlg::getbumpAngle()
{
    return bumpAngle;
}


int CFangDlg::getbumpCamAngle()
{
    return bumpCamAngle;
}


int CFangDlg::getbumpDistance()
{
    return bumpDistance;
}


int CFangDlg::getbumperRate()
{
    return bumperRate;
}


int CFangDlg::getconfigRate()
{
    return configRate;
}


int CFangDlg::getheartbeatRate()
{
    return heartbeatRate;
}


int CFangDlg::getsonarRate()
{
    return sonarRate;
}


int CFangDlg::getspeedRate()
{
    return speedRate;
}


void CFangDlg::LogCommIO(bool log)
{
    if(robot == NULL)
        return;
    logIO = log;
```

```
   robot->logCommIO(log);
}


bool CFangDlg::getLogIO()
{
   return logIO;
}

void CFangDlg::PrePaint(vector<bool> sonars_changed, bool bumpers_changed)
{
   int i;
   CBitmap bit;
   BITMAP *pbm = new BITMAP;

   bit.LoadBitmap(MAKEINTRESOURCE(IDB_BITMAP_BUMPOFF0));
   bit.GetBitmap(pbm);
   bit.DeleteObject();

   //invalidate sonar lines
   CPoint homePoint, startPoint, endPoint, offset;
   homePoint.x = XCENTER;
   homePoint.y = YCENTER;
   //5 is so that it invalidates around entire sonar rect
   offset.x = pbm->bmWidth / 2 + 5;
   offset.y = pbm->bmHeight / 2 + 5;
   double currDegrees = 270;
   const double pi = 3.14159;
   const double d2r = pi / 180;

   CPoint points[5];
   CPoint currPoint, bot;
   //bot is 3 greater than when drawing to catch edges
   bot.x = 8; bot.y = 8;

   for(i = 0; i < NUM_SONARS; i++)
   {
      currPoint = homePoint + CPoint(offset.x * cos(currDegrees * d2r),
         offset.y * sin(currDegrees * d2r));
      points[0] = currPoint;

      currPoint += CPoint(sin(currDegrees * d2r) *-1* bot.x, cos(currDegrees * d2r) *
 bot.y);
      points[1] = currPoint;

      currPoint += CPoint(scale(MAX_SONAR_DIST + 1) * cos(currDegrees * d2r),
         scale(MAX_SONAR_DIST + 1) * sin(currDegrees * d2r));
      points[2] = currPoint;

      currPoint -= CPoint(sin(currDegrees * d2r) *-2* bot.x, cos(currDegrees * d2r) *
2* bot.y);
      points[3] = currPoint;

      currPoint -= CPoint(scale(MAX_SONAR_DIST + 1) * cos(currDegrees * d2r),
         scale(MAX_SONAR_DIST + 1) * sin(currDegrees * d2r));
      points[4] = currPoint;

     if(sonars_changed[i]){
        CRgn invalidateArea;
        invalidateArea.CreatePolygonRgn(points, 5, WINDING);
        InvalidateRgn(&invalidateArea);
     }

      currDegrees -= 22.5;
   }

   //invalidate bumpers
   if(bumpers_changed){
      CRgn invalidateArea;
      CRect invalidateRect(CPoint(XCENTER - offset.x, YCENTER - offset.y),
         CPoint(XCENTER + offset.x, YCENTER + offset.y));
      invalidateArea.CreateEllipticRgnIndirect(invalidateRect);
      InvalidateRgn(&invalidateArea);
```



```
   }
}
```

```
// $Id: joystick.h,v 1.5 2002/04/27 16:29:01 ccf7f Exp $
// Flancrest Enterprises, Group 22
// About this file:
// TODO

// Notes on usage:
//  Init() is automatically called by the constructor, no further calls
//    of Init() are necessary unless you want to handle a change in the joystick
//    (eg a joystick is connected when this class is instantiated, and then
//    later the joystick is swapped out with a different joystick with different
//    capabilities (like number of buttons or axes).
//    See the Query(*axes, Num_Axes, *returned_axes) call to have Query()
//    call Init() everytime it is called if you wish to change this.
//    (Query() does not presently do this because Init() has some overhead,
//    which may be quite high if no joysticks are attached.)

#ifndef _Joystick_H_
#define _Joystick_H_

#include <windows.h>
#include <Mmsystem.h>

#pragma warning(disable:4800) // disable performance warnings for int->bool

class Joystick
{
   public:
      // Constructors/destructors
      Joystick();
      ~Joystick();

      int Init(); // Non-one if fail
      // errNoJoyDrivers if computer does not have joystick drivers,
     //   errNoJoysAttached if no joysticks are attached,
     //   errNumButtonsGrMaxNumButtons if found NumButtons > MaxNumButtons
     //   (which is ok, but only MaxNumButtons will be used).

      // Query() returns the same error messages as Init(), with the addition of
     //   errJoyQueryFailure which means the joystick query failed,
     //   and errNullPointerPassed if a null pointer was passed in.
      // axes and buttons are arrays of the returned elements (each axes element give
s
      //   the location in that axis from -100 to 100), and returns at max
      //   the number of elements which fill the provided array.
      // Num_[Axes|Buttons] tells Query() how many elements the axes/buttons
      //   array have memory for.
      // Returned_[Axes|Buttons] tells the caller how many axes/buttons were actually
      //   returned.
      // The axes are returned in order: X, Y, Z, R, U, V. If any axes is missing
      //   the others will move up (eg if only U is not present, X, Y, Z, R, V will b
e
      //   returned).
      int Query(double *axes,    const unsigned int Num_Axes,   unsigned int &Returne
d_Axes,
         bool *buttons, const unsigned int Num_Buttons,unsigned int &Returned_Buttons
);
      int Query(double *axes,    const unsigned int Num_Axes,   unsigned int &Returne
d_Axes);

      unsigned int GetNumAxes() const;
      unsigned int GetNumButtons() const;

     // Return values for Query()
     const int errNumButtonsGrMaxNumButtons, errNoJoysAttached, errNoJoyDrivers,
        errJoyQueryFailure, errNullPointerPassed;
   private:
      // Scales axis
      void ScaleAxis(double &axis, double scale, int range) const;
      // Correct axis for center-calculation roundoff error
      void DeadCheck(double &axis) const;

      JOYINFOEX joyinfo;
      unsigned int wDeviceID;
```



```
      // Until we know how to query for variable number of buttons,
     //   make this compile-time defined.
      const unsigned int MaxNumButtons;
      unsigned int NumButtons;
      // Scale is 0 if axis not present
      double POVScale, RScale, UScale, VScale,  XScale, YScale, ZScale;
      int    POVRange, RRange, URange, VRange,  XRange, YRange, ZRange;
};
#endif
```

```
// $Id: joystick.cpp,v 1.5 2002/04/27 16:29:01 ccf7f Exp $
// Flancrest Enterprises, Group 22

#include "stdafx.h"

#include "joystick.h"
#include <math.h>

Joystick::Joystick()
: MaxNumButtons(12),
errNumButtonsGrMaxNumButtons(-1),
errNoJoysAttached(-2),
errNoJoyDrivers(-3),
errJoyQueryFailure(-4),
errNullPointerPassed(-5)
{
   XRange = YRange = ZRange = RRange = URange = VRange = 200;
   POVRange = 2;
   POVScale = XScale = YScale = ZScale = RScale = UScale = VScale = 0;
   NumButtons = 0;

   Init();
}


Joystick::~Joystick()
{
}


int Joystick::Init()
{
   JOYCAPS joyCaps;
   bool JoyHasAxisR, JoyHasAxisU, JoyHasAxisV, JoyHasAxisZ, JoyHasPOV;
   JoyHasAxisR = JoyHasAxisU = JoyHasAxisV = JoyHasAxisZ = JoyHasPOV = false;

   int MinR, MaxR, MinU, MaxU, MinV, MaxV, MaxX, MinX, MaxY, MinY,
      MaxZ, MinZ, MaxPOV, MinPOV;
   MinR = MaxR = MinU = MaxU = MinV = MaxV = MaxX = MinX = MaxY = MinY
      = MaxZ = MinZ = MaxPOV = MinPOV = 0;

   UINT wNumDevs = 0;
   BOOL bDev1Attached = false, bDev2Attached = false;

   memset(&joyinfo, 0, sizeof(JOYINFOEX));
   joyinfo.dwSize = sizeof(JOYINFOEX);
   joyinfo.dwFlags = JOY_RETURNALL;

   // Does this computer support joysticks?
   if((wNumDevs = joyGetNumDevs()) == 0)
   {
      // No joystick drivers
      return errNoJoyDrivers;
   }

   // Are there any attached joysticks? (we only check the first two)
   //   Decide which to use if so.
   bDev1Attached = (joyGetPosEx(JOYSTICKID1, &joyinfo) != JOYERR_UNPLUGGED);
   bDev2Attached = wNumDevs == 2
      && joyGetPosEx(JOYSTICKID2, &joyinfo) != JOYERR_UNPLUGGED;
                                    // decide which joystick to use
   if(bDev1Attached || bDev2Attached)
      wDeviceID = bDev1Attached ? JOYSTICKID1 : JOYSTICKID2;
   else
   {
      // No attached joysticks
      return errNoJoysAttached;
   }

   // What are the capabilities of the joystick we've found?
   // Check to see which axes exist and get ranges for those that do.
   // Find how many buttons exist.
   joyGetDevCaps(wDeviceID, &joyCaps, sizeof(joyCaps));
```



```
   MinX = joyCaps.wXmin;
   MaxX = joyCaps.wXmax;
   XScale = XRange/(double)(MaxX - MinX);

   MinY = joyCaps.wYmin;
   MaxY = joyCaps.wYmax;
   YScale = YRange/(double)(MaxY - MinY);

   JoyHasAxisZ = joyCaps.wCaps & JOYCAPS_HASZ;
   if (JoyHasAxisZ)
   {
      MinZ = joyCaps.wZmin;
      MaxZ = joyCaps.wZmax;
      ZScale = ZRange/(double)(MaxZ - MinZ);
   }

   JoyHasAxisR = joyCaps.wCaps & JOYCAPS_HASR;
   if (JoyHasAxisR)
   {
      MinR = joyCaps.wRmin;
      MaxR = joyCaps.wRmax;
      RScale = RRange/(double)(MaxR - MinR);
   }

   JoyHasAxisU = joyCaps.wCaps & JOYCAPS_HASU;
   if (JoyHasAxisU)
   {
      MinU = joyCaps.wUmin;
      MaxU = joyCaps.wUmax;
      UScale = URange/(double)(MaxU - MinU);
   }

   JoyHasAxisV = joyCaps.wCaps & JOYCAPS_HASV;
   if (JoyHasAxisV)
   {
      MinV = joyCaps.wVmin;
      MaxV = joyCaps.wVmax;
      VScale = VRange/(double)(MaxV - MinV);
   }

   JoyHasPOV = (joyCaps.wCaps & JOYCAPS_POV4DIR)
      || (joyCaps.wCaps & JOYCAPS_POVCTS);
   if (JoyHasPOV)
   {
     MinPOV = -1;
     MaxPOV = 1;
      POVScale = POVRange/(double)(MaxPOV - MinPOV);
   }

   NumButtons = joyCaps.wNumButtons;
   if(NumButtons > MaxNumButtons)
   {
      NumButtons = MaxNumButtons;
      return errNumButtonsGrMaxNumButtons;
   }

   return 1;
}


int Joystick::Query(double *axes,  const unsigned int Num_Axes,
unsigned int &Returned_Axes, bool *buttons,
const unsigned int Num_Buttons,
unsigned int &Returned_Buttons)
{
   const int QueryReturnValue = Query(axes, Num_Axes, Returned_Axes);
   if(1 != QueryReturnValue)
      return QueryReturnValue;

   //
   // Read buttons
   //
```

```
    if (NULL == buttons)
    {
        return errNullPointerPassed;
    }

    unsigned int NumButtonsToCheck = Num_Buttons;
    if(Num_Buttons > NumButtons)
    {
        NumButtonsToCheck = NumButtons;
    }
    Returned_Buttons = NumButtonsToCheck;

    // TODO: The below code is compile time dependent. How can the
    //    JOY_BUTTON enumeration be done dynamically?
    if (NumButtonsToCheck >= 1)
        buttons[0] =  (BOOL)(joyinfo.dwButtons & JOY_BUTTON1);
    if (NumButtonsToCheck >= 2)
        buttons[1] =  (BOOL)(joyinfo.dwButtons & JOY_BUTTON2);
    if (NumButtonsToCheck >= 3)
        buttons[2] =  (BOOL)(joyinfo.dwButtons & JOY_BUTTON3);
    if (NumButtonsToCheck >= 4)
        buttons[3] =  (BOOL)(joyinfo.dwButtons & JOY_BUTTON4);
    if (NumButtonsToCheck >= 5)
        buttons[4] =  (BOOL)(joyinfo.dwButtons & JOY_BUTTON5);
    if (NumButtonsToCheck >= 6)
        buttons[5] =  (BOOL)(joyinfo.dwButtons & JOY_BUTTON6);
    if (NumButtonsToCheck >= 7)
        buttons[6] =  (BOOL)(joyinfo.dwButtons & JOY_BUTTON7);
    if (NumButtonsToCheck >= 8)
        buttons[7] =  (BOOL)(joyinfo.dwButtons & JOY_BUTTON8);
    if (NumButtonsToCheck >= 9)
        buttons[8] =  (BOOL)(joyinfo.dwButtons & JOY_BUTTON9);
    if (NumButtonsToCheck >= 10)
        buttons[9] =  (BOOL)(joyinfo.dwButtons & JOY_BUTTON10);
    if (NumButtonsToCheck >= 11)
        buttons[10] = (BOOL)(joyinfo.dwButtons & JOY_BUTTON11);
    if (NumButtonsToCheck >= 12)
        buttons[11] = (BOOL)(joyinfo.dwButtons & JOY_BUTTON12);

    return 1;
}


int Joystick::Query(double *axes, const unsigned int Num_Axes,
unsigned int &Returned_Axes)
{
    const unsigned int NumAxes = GetNumAxes();
    unsigned int AxesNum = 0;

    if(NULL == axes)
    {
        return errNullPointerPassed;
    }

    /*  // Were you to want Init() to be called with every Query() call,
        //   uncomment these four lines.
        const int InitReturn = Init();
        if(1 != InitReturn) {
            return InitReturn;
        }
    */
    if(joyGetPosEx(wDeviceID, &joyinfo) != JOYERR_NOERROR)
    {
        return errJoyQueryFailure;
    }

    unsigned int NumAxesToCheck = Num_Axes;
    if(Num_Axes > NumAxes)
    {
        NumAxesToCheck = NumAxes;
    }
    Returned_Axes = NumAxesToCheck;
```



```
    if ((0 != XScale) && (NumAxesToCheck > AxesNum))
    {
        axes[AxesNum] = joyinfo.dwXpos;
        ScaleAxis(axes[AxesNum], XScale, XRange);
        AxesNum++;
    }

    if ((0 != YScale) && (NumAxesToCheck > AxesNum))
    {
        axes[AxesNum] = joyinfo.dwYpos;
        ScaleAxis(axes[AxesNum], YScale, YRange);
        AxesNum++;
    }

    if ((0 != ZScale) && (NumAxesToCheck > AxesNum))
    {
        axes[AxesNum] = joyinfo.dwZpos;
        ScaleAxis(axes[AxesNum], ZScale, ZRange);
        AxesNum++;
    }

    if ((0 != RScale) && (NumAxesToCheck > AxesNum))
    {
        axes[AxesNum] = joyinfo.dwRpos;
        ScaleAxis(axes[AxesNum], RScale, RRange);
        AxesNum++;
    }

    if ((0 != UScale) && (NumAxesToCheck > AxesNum))
    {
        axes[AxesNum] = joyinfo.dwUpos;
        ScaleAxis(axes[AxesNum], UScale, URange);
        AxesNum++;
    }

    if ((0 != VScale) && (NumAxesToCheck > AxesNum))
    {
        axes[AxesNum] = joyinfo.dwVpos;
        ScaleAxis(axes[AxesNum], VScale, VRange);
        AxesNum++;
    }

    if ((0 != POVScale) && (NumAxesToCheck > AxesNum+1))
    {
        if (65535 == joyinfo.dwPOV)
        {
            axes[AxesNum] = 0;
            AxesNum++;
            axes[AxesNum] = 0;
            AxesNum++;
        }
        else
        {
            const double degreesToRadians = 3.141592653589793 / 180;
            const int scaleFactor = 100;
            axes[AxesNum] = scaleFactor*sin((joyinfo.dwPOV / 100)*degreesToRadians);
            DeadCheck(axes[AxesNum]);
            AxesNum++;
            axes[AxesNum] = -1*scaleFactor*cos((joyinfo.dwPOV / 100)*degreesToRadians);
            DeadCheck(axes[AxesNum]);
            AxesNum++;
        }
    }

    return 1;
}


unsigned int Joystick::GetNumAxes() const
{
    int NumAxes = 0;

    if(XScale)
```

```
      NumAxes++;
   if(YScale)
      NumAxes++;
   if(ZScale)
      NumAxes++;
   if(RScale)
      NumAxes++;
   if(UScale)
      NumAxes++;
   if(VScale)
      NumAxes++;
   if(POVScale)
      NumAxes += 2;

   return NumAxes;
}


unsigned int Joystick::GetNumButtons() const
{
   return NumButtons;
}


// Private

void Joystick::ScaleAxis(double &axis, double scale, int range) const
{
   axis *= scale;
   axis = axis - (range/2);
   DeadCheck(axis);
}


void Joystick::DeadCheck(double &axis) const
{
   if (0.002 > axis && -0.002 < axis)
      axis = 0;
}
```

```
//{{NO_DEPENDENCIES}}
// Microsoft Developer Studio generated include file.
// Used by fang.rc
//
#define IDM_ABOUTBOX                    0x0010
#define IDM_CONFIGBOX                   0x0020
#define IDM_JOYSTICK                    0x0030
#define IDD_ABOUTBOX                    100
#define IDS_ABOUTBOX                    101
#define IDD_FANG_DIALOG                 102
#define IDP_SOCKETS_INIT_FAILED         103
#define IDS_CONFIGBOX                   104
#define IDS_JOYSTICK                    105
#define IDR_MAINFRAME                   128
#define IDB_BITMAP_ROBOT                133
#define IDB_BITMAP_BUMP0                135
#define IDB_BITMAP_BUMP1                136
#define IDD_CONFIGBOX                   137
#define IDB_BITMAP_BUMPOFF1             137
#define IDB_BITMAP_BUMPOFF2             138
#define IDB_BITMAP_BUMPOFF3             139
#define IDB_BITMAP_BUMPOFF4             140
#define IDB_BITMAP_BUMPOFF5             141
#define IDB_BITMAP_BUMPOFF0             142
#define IDB_BITMAP_BUMPON0              143
#define IDB_BITMAP_BUMPON1              144
#define IDB_BITMAP_BUMPON2              145
#define IDB_BITMAP_BUMPON3              146
#define IDB_BITMAP_BUMPON4              147
#define IDB_BITMAP_BUMPON5              149
#define IDC_BUTTON_FORWARD              1000
#define IDC_BUTTON_STOP                 1001
#define IDC_BUTTON_RIGHT                1002
#define IDC_BUTTON_LEFT                 1004
#define IDC_BUTTON_CAM_LEFT             1007
#define IDC_BUTTON_CAM_UP               1008
#define IDC_BUTTON_CAM_RIGHT            1009
#define IDC_BUTTON_CAM_DOWN             1010
#define IDC_EDIT_SERVER_NAME            1011
#define IDC_BUTTON_CONNECT              1013
#define IDC_STATIC_CAMERA_CONTROLS      1014
#define IDC_STATIC_ROBOT_CONTROLS       1015
#define IDC_STATIC_STATUS_INFORMATION   1018
#define IDC_STATIC_REMOTE_HOST          1019
#define IDC_STATIC_CONNECTED            1021
#define IDC_STATIC_HEARTBEAT            1022
#define IDC_BUTTON1                     1024
#define IDC_BUTTON_HOME                 1024
#define IDC_BUTTON_APPLY                1024
#define IDC_STATIC_ROBOT_SENSORS        1025
#define IDC_SLIDER_CAMTILT              1028
#define IDC_SLIDER_CAMPAN               1029
#define IDC_BUTTON_BACK                 1030
#define IDC_BUTTON_FWDRIGHT             1031
#define IDC_BUTTON_FWDLEFT              1032
#define IDC_BUTTON_BACKRIGHT            1033
#define IDC_BUTTON_BACKLEFT             1034
#define IDC_SLIDER_ROBOTSPEED           1035
#define IDC_STATIC_RBTSPEED             1036
#define IDC_BUTTON_MODE                 1037
#define IDC_BUTTON_RESET                1038
#define IDC_STATIC_MODE                 1039
#define IDC_CHECK_SONAR                 1040
#define IDC_CHECK_BUMPER                1041
#define IDC_CHECK_DEBUG                 1042
#define IDC_CHECK_BUMP                  1043
#define IDC_CHECK_LOGIO                 1044
#define IDC_EDIT_BUMP_DISTANCE          1046
#define IDC_EDIT_BUMP_ANGLE             1047
#define IDC_EDIT_CAM_BUMP_ANGLE         1048
#define IDC_EDIT_BUMP_DISTANCE2         1049
#define IDC_EDIT_SONAR_RATE             1049
#define IDC_EDIT_BUMP_ANGLE2            1050
```



```
#define IDC_EDIT_BUMP_RATE              1050
#define IDC_EDIT_CAM_BUMP_ANGLE2        1051
#define IDC_EDIT_CONFIG_RATE            1051
#define IDC_EDIT_BUMP_DISTANCE3         1052
#define IDC_EDIT_SPEED_RATE             1052
#define IDC_EDIT_BUMP_ANGLE3            1053
#define IDC_EDIT_HEARTBEAT_RATE         1053

// Next default values for new objects
//
#ifdef APSTUDIO_INVOKED
#ifndef APSTUDIO_READONLY_SYMBOLS
#define _APS_NEXT_RESOURCE_VALUE        143
#define _APS_NEXT_COMMAND_VALUE         32771
#define _APS_NEXT_CONTROL_VALUE         1042
#define _APS_NEXT_SYMED_VALUE           103
#endif
#endif
```

```
// stdafx.h : include file for standard system include files,
//  or project specific include files that are used frequently, but
//      are changed infrequently
//

#if !defined(AFX_STDAFX_H__A9990B34_C3CC_4C59_82B6_6B5881941303__INCLUDED_)
#define AFX_STDAFX_H__A9990B34_C3CC_4C59_82B6_6B5881941303__INCLUDED_

#if _MSC_VER > 1000
#pragma once
#endif                          // _MSC_VER > 1000

#pragma warning(disable: 4786)

#define VC_EXTRALEAN            // Exclude rarely-used stuff from Windows headers

#include <afxwin.h>             // MFC core and standard components
#include <afxext.h>             // MFC extensions
#include <afxdtctl.h>           // MFC support for Internet Explorer 4 Common Contro
ls
#ifndef _AFX_NO_AFXCMN_SUPPORT
#include <afxcmn.h>             // MFC support for Windows Common Controls
#endif                          // _AFX_NO_AFXCMN_SUPPORT

#include <afxsock.h>            // MFC socket extensions

//{{AFX_INSERT_LOCATION}}
// Microsoft Visual C++ will insert additional declarations immediately before the pr
evious line.
#endif                          // !defined(AFX_STDAFX_H__A9990B34_C3CC_4C59_82B6_6B
5881941303__INCLUDED_)
```

Apr 21, 02 15:33 **StdAfx.cpp** Page 1/1

```
// stdafx.cpp : source file that includes just the standard includes
// fang.pch will be the pre-compiled header
// stdafx.obj will contain the pre-compiled type information

#include "stdafx.h"
```

```
//Microsoft Developer Studio generated resource script.
//
#include "resource.h"

#define APSTUDIO_READONLY_SYMBOLS
/////////////////////////////////////////////////////////////////////////////
//
// Generated from the TEXTINCLUDE 2 resource.
//
#include "afxres.h"

/////////////////////////////////////////////////////////////////////////////
#undef APSTUDIO_READONLY_SYMBOLS

/////////////////////////////////////////////////////////////////////////////
// English (U.S.) resources

#if !defined(AFX_RESOURCE_DLL) || defined(AFX_TARG_ENU)
#ifdef _WIN32
LANGUAGE LANG_ENGLISH, SUBLANG_ENGLISH_US
#pragma code_page(1252)
#endif //_WIN32

#ifdef APSTUDIO_INVOKED
/////////////////////////////////////////////////////////////////////////////
//
// TEXTINCLUDE
//

1 TEXTINCLUDE DISCARDABLE
BEGIN
    "resource.h\0"
END

2 TEXTINCLUDE DISCARDABLE
BEGIN
    "#include ""afxres.h""\r\n"
    "\0"
END

3 TEXTINCLUDE DISCARDABLE
BEGIN
    "#define _AFX_NO_SPLITTER_RESOURCES\r\n"
    "#define _AFX_NO_OLE_RESOURCES\r\n"
    "#define _AFX_NO_TRACKER_RESOURCES\r\n"
    "#define _AFX_NO_PROPERTY_RESOURCES\r\n"
    "\r\n"
    "#if !defined(AFX_RESOURCE_DLL) || defined(AFX_TARG_ENU)\r\n"
    "#ifdef _WIN32\r\n"
    "LANGUAGE 9, 1\r\n"
    "#pragma code_page(1252)\r\n"
    "#endif //_WIN32\r\n"
    "#include ""res\\fang.rc2""  // non-Microsoft Visual C++ edited resources\r\n"
    "#include ""afxres.rc""         // Standard components\r\n"
    "#endif\r\n"
    "\0"
END

#endif    // APSTUDIO_INVOKED


/////////////////////////////////////////////////////////////////////////////
//
// Icon
//

// Icon with lowest ID value placed first to ensure application icon
// remains consistent on all systems.
IDR_MAINFRAME           ICON    DISCARDABLE     "res\\fang.ico"

/////////////////////////////////////////////////////////////////////////////
//
// Dialog
```



```
//

IDD_ABOUTBOX DIALOG DISCARDABLE  0, 0, 235, 55
STYLE DS_MODALFRAME | WS_POPUP | WS_CAPTION | WS_SYSMENU
CAPTION "About fang"
FONT 8, "MS Sans Serif"
BEGIN
    ICON            IDR_MAINFRAME,IDC_STATIC,10,15,20,20
    LTEXT           "FANG Version 1.0",IDC_STATIC,39,12,119,8,SS_NOPREFIX
    LTEXT           "Copyright (C) 2002 Flancrest Enterprises",IDC_STATIC,39,
                    28,136,8
    DEFPUSHBUTTON   "OK",IDOK,178,7,50,14,WS_GROUP
END

IDD_FANG_DIALOG DIALOGEX 0, 0, 357, 286
STYLE DS_MODALFRAME | WS_POPUP | WS_VISIBLE | WS_CAPTION | WS_SYSMENU
EXSTYLE WS_EX_APPWINDOW
CAPTION "FANG"
FONT 8, "MS Sans Serif", 0, 0, 0x1
BEGIN
    PUSHBUTTON      "Forward",IDC_BUTTON_FORWARD,67,188,45,13
    PUSHBUTTON      "Stop",IDC_BUTTON_STOP,67,205,45,13
    PUSHBUTTON      "Right",IDC_BUTTON_RIGHT,115,204,45,13
    PUSHBUTTON      "Left",IDC_BUTTON_LEFT,19,204,45,13
    PUSHBUTTON      "Pan Left",IDC_BUTTON_CAM_LEFT,205,207,45,13
    PUSHBUTTON      "Pan Up",IDC_BUTTON_CAM_UP,233,186,45,13
    PUSHBUTTON      "Pan Right",IDC_BUTTON_CAM_RIGHT,261,207,45,13
    PUSHBUTTON      "Pan Down",IDC_BUTTON_CAM_DOWN,233,226,45,13
    EDITTEXT        IDC_EDIT_SERVER_NAME,189,16,93,13,ES_AUTOHSCROLL
    PUSHBUTTON      "Connect",IDC_BUTTON_CONNECT,291,16,45,13
    GROUPBOX        "Camera Controls",IDC_STATIC_CAMERA_CONTROLS,182,150,170,
                    130
    GROUPBOX        "Robot Controls",IDC_STATIC_ROBOT_CONTROLS,6,150,170,130
    GROUPBOX        "Status Information",IDC_STATIC_STATUS_INFORMATION,182,
                    40,170,108
    GROUPBOX        "Remote Host",IDC_STATIC_REMOTE_HOST,181,5,170,32
    LTEXT           "Not Connected",IDC_STATIC_HEARTBEAT,195,60,87,9
    GROUPBOX        "Robot Sensors",IDC_STATIC_ROBOT_SENSORS,5,5,170,143
    CONTROL         "Slider3",IDC_SLIDER_CAMTILT,"msctls_trackbar32",
                    TBS_VERT | TBS_BOTH | TBS_NOTICKS | WS_TABSTOP,317,170,
                    20,85
    CONTROL         "Slider4",IDC_SLIDER_CAMPAN,"msctls_trackbar32",TBS_BOTH |
                    TBS_NOTICKS | WS_TABSTOP,213,250,85,15
    PUSHBUTTON      "Back",IDC_BUTTON_BACK,67,222,45,13
    PUSHBUTTON      "Fwd Right",IDC_BUTTON_FWDRIGHT,115,188,45,13
    PUSHBUTTON      "Fwd Left",IDC_BUTTON_FWDLEFT,19,188,45,13
    PUSHBUTTON      "Back Right",IDC_BUTTON_BACKRIGHT,115,222,45,13
    PUSHBUTTON      "Back Left",IDC_BUTTON_BACKLEFT,19,222,45,13
    CONTROL         "Slider5",IDC_SLIDER_ROBOTSPEED,"msctls_trackbar32",
                    TBS_BOTH | TBS_NOTICKS | WS_TABSTOP,63,248,85,15
    LTEXT           "Speed:",IDC_STATIC_RBTSPEED,31,251,27,10
    PUSHBUTTON      "Home",IDC_BUTTON_HOME,302,257,45,13
    PUSHBUTTON      "Mode",IDC_BUTTON_MODE,40,165,45,13
    PUSHBUTTON      "Reset",IDC_BUTTON_RESET,96,165,45,13
    LTEXT           "Mode: Run",IDC_STATIC_MODE,195,76,62,12
    CONTROL         "Get Sonar Data",IDC_CHECK_SONAR,"Button",
                    BS_AUTOCHECKBOX | WS_TABSTOP,194,114,82,9
    CONTROL         "Get Bumper Data",IDC_CHECK_BUMPER,"Button",
                    BS_AUTOCHECKBOX | WS_TABSTOP,194,129,84,12
END

IDD_CONFIGBOX DIALOG DISCARDABLE  0, 0, 295, 127
STYLE DS_MODALFRAME | WS_POPUP | WS_CAPTION | WS_SYSMENU
CAPTION "Settings"
FONT 8, "MS Sans Serif"
BEGIN
    DEFPUSHBUTTON   "Close",IDOK,231,109,50,14
    PUSHBUTTON      "Apply",IDC_BUTTON_APPLY,175,109,50,14
    LTEXT           "Bump Distance:",IDC_STATIC,13,20,80,8
    LTEXT           "Rotation Bump Angle:",IDC_STATIC,13,38,80,8
    LTEXT           "Camera Bump Angle:",IDC_STATIC,13,55,80,8
    LTEXT           "Sonar Data:",IDC_STATIC,176,20,46,8
    LTEXT           "Bumper Data:",IDC_STATIC,176,38,48,8
```

```
    GROUPBOX        "Refresh Rates",IDC_STATIC,168,5,122,99
    LTEXT           "Configuration Data:",IDC_STATIC,176,55,61,8
    LTEXT           "Speed Data:",IDC_STATIC,176,70,46,8
    LTEXT           "Heartbeat:",IDC_STATIC,176,85,42,8
    CONTROL         "Show Debug Window",IDC_CHECK_DEBUG,"Button",
                    BS_AUTOCHECKBOX | WS_TABSTOP,4,86,86,8
    CONTROL         "Stop On Bump",IDC_CHECK_BUMP,"Button",BS_AUTOCHECKBOX |
                    WS_TABSTOP,99,86,61,8
    GROUPBOX        "Movement Parameters",IDC_STATIC,5,5,150,70
    EDITTEXT        IDC_EDIT_BUMP_DISTANCE,93,18,25,12,ES_AUTOHSCROLL
    EDITTEXT        IDC_EDIT_BUMP_ANGLE,93,35,25,12,ES_AUTOHSCROLL
    EDITTEXT        IDC_EDIT_CAM_BUMP_ANGLE,93,52,25,12,ES_AUTOHSCROLL
    LTEXT           "inches",IDC_STATIC,121,20,24,8
    LTEXT           "degrees",IDC_STATIC,121,37,30,8
    LTEXT           "degrees",IDC_STATIC,121,54,29,8
    EDITTEXT        IDC_EDIT_SONAR_RATE,241,18,25,12,ES_AUTOHSCROLL
    EDITTEXT        IDC_EDIT_BUMP_RATE,241,35,25,12,ES_AUTOHSCROLL
    EDITTEXT        IDC_EDIT_CONFIG_RATE,241,52,25,12,ES_AUTOHSCROLL
    LTEXT           "Hz",IDC_STATIC,269,20,14,8
    LTEXT           "Hz",IDC_STATIC,269,37,14,8
    LTEXT           "Hz",IDC_STATIC,269,54,14,8
    EDITTEXT        IDC_EDIT_SPEED_RATE,241,69,25,12,ES_AUTOHSCROLL
    EDITTEXT        IDC_EDIT_HEARTBEAT_RATE,241,85,25,12,ES_AUTOHSCROLL
    LTEXT           "Hz",IDC_STATIC,269,71,14,8
    LTEXT           "Hz",IDC_STATIC,269,87,14,8
    CONTROL         "Log Communication I/O",IDC_CHECK_LOGIO,"Button",
                    BS_AUTOCHECKBOX | WS_TABSTOP,4,99,112,8
END


#ifndef _MAC
/////////////////////////////////////////////////////////////////////////////
//
// Version
//

VS_VERSION_INFO VERSIONINFO
 FILEVERSION 1,0,0,1
 PRODUCTVERSION 1,0,0,1
 FILEFLAGSMASK 0x3fL
#ifdef _DEBUG
 FILEFLAGS 0x1L
#else
 FILEFLAGS 0x0L
#endif
 FILEOS 0x4L
 FILETYPE 0x1L
 FILESUBTYPE 0x0L
BEGIN
    BLOCK "StringFileInfo"
    BEGIN
        BLOCK "040904b0"
        BEGIN
            VALUE "Comments", "\0"
            VALUE "CompanyName", "\0"
            VALUE "FileDescription", "fang MFC Application\0"
            VALUE "FileVersion", "1, 0, 0, 1\0"
            VALUE "InternalName", "fang\0"
            VALUE "LegalCopyright", "Copyright (C) 2002\0"
            VALUE "LegalTrademarks", "\0"
            VALUE "OriginalFilename", "fang.EXE\0"
            VALUE "PrivateBuild", "\0"
            VALUE "ProductName", "fang Application\0"
            VALUE "ProductVersion", "1, 0, 0, 1\0"
            VALUE "SpecialBuild", "\0"
        END
    END
    BLOCK "VarFileInfo"
    BEGIN
        VALUE "Translation", 0x409, 1200
    END
END
```



```
#endif    // !_MAC


/////////////////////////////////////////////////////////////////////////////
//
// DESIGNINFO
//

#ifdef APSTUDIO_INVOKED
GUIDELINES DESIGNINFO DISCARDABLE
BEGIN
    IDD_ABOUTBOX, DIALOG
    BEGIN
        LEFTMARGIN, 7
        RIGHTMARGIN, 228
        TOPMARGIN, 7
        BOTTOMMARGIN, 48
    END

    IDD_FANG_DIALOG, DIALOG
    BEGIN
        LEFTMARGIN, 7
        RIGHTMARGIN, 350
        TOPMARGIN, 7
        BOTTOMMARGIN, 279
    END

    IDD_CONFIGBOX, DIALOG
    BEGIN
        LEFTMARGIN, 7
        RIGHTMARGIN, 288
        TOPMARGIN, 7
        BOTTOMMARGIN, 120
    END
END
#endif    // APSTUDIO_INVOKED


/////////////////////////////////////////////////////////////////////////////
//
// Bitmap
//

IDB_BITMAP_ROBOT        BITMAP  DISCARDABLE     "res\\robot.bmp"
IDB_BITMAP_BUMPOFF1     BITMAP  DISCARDABLE     "res\\bumpOff1.bmp"
IDB_BITMAP_BUMPOFF2     BITMAP  DISCARDABLE     "res\\bumpOff2.bmp"
IDB_BITMAP_BUMPOFF3     BITMAP  DISCARDABLE     "res\\bumpOff3.bmp"
IDB_BITMAP_BUMPOFF4     BITMAP  DISCARDABLE     "res\\bumpOff4.bmp"
IDB_BITMAP_BUMPOFF5     BITMAP  DISCARDABLE     "res\\bumpOff5.bmp"
IDB_BITMAP_BUMPOFF0     BITMAP  DISCARDABLE     "res\\bumpOff0.bmp"
IDB_BITMAP_BUMPON0      BITMAP  DISCARDABLE     "res\\bumpOn0.bmp"
IDB_BITMAP_BUMPON1      BITMAP  DISCARDABLE     "res\\bumpOn1.bmp"
IDB_BITMAP_BUMPON2      BITMAP  DISCARDABLE     "res\\bumpOn2.bmp"
IDB_BITMAP_BUMPON3      BITMAP  DISCARDABLE     "res\\bumpOn3.bmp"
IDB_BITMAP_BUMPON4      BITMAP  DISCARDABLE     "res\\bumpOn4.bmp"
IDB_BITMAP_BUMPON5      BITMAP  DISCARDABLE     "res\\bumpOn5.bmp"

/////////////////////////////////////////////////////////////////////////////
//
// String Table
//

STRINGTABLE DISCARDABLE
BEGIN
    IDS_ABOUTBOX            "&About fang..."
    IDP_SOCKETS_INIT_FAILED "Windows sockets initialization failed."
    IDS_CONFIGBOX           "&Configure"
    IDS_JOYSTICK            "&Retry Joystick"
END

#endif    // English (U.S.) resources
/////////////////////////////////////////////////////////////////////////////
```

```
#ifndef APSTUDIO_INVOKED
/////////////////////////////////////////////////////////////////////////////
//
// Generated from the TEXTINCLUDE 3 resource.
//
#define _AFX_NO_SPLITTER_RESOURCES
#define _AFX_NO_OLE_RESOURCES
#define _AFX_NO_TRACKER_RESOURCES
#define _AFX_NO_PROPERTY_RESOURCES

#if !defined(AFX_RESOURCE_DLL) || defined(AFX_TARG_ENU)
#ifdef _WIN32
LANGUAGE 9, 1
#pragma code_page(1252)
#endif //_WIN32
#include "res\fang.rc2"  // non-Microsoft Visual C++ edited resources
#include "afxres.rc"         // Standard components
#endif

/////////////////////////////////////////////////////////////////////////////
#endif    // not APSTUDIO_INVOKED
```

Mar 22, 02 10:01 **fang.rc2** Page 1/1

```
//
// FANG.RC2 - resources Microsoft Visual C++ does not edit directly
//

#ifdef APSTUDIO_INVOKED
   #error this file is not editable by Microsoft Visual C++
#endif //APSTUDIO_INVOKED


/////////////////////////////////////////////////////////////////////////////
// Add manually edited resources here...

/////////////////////////////////////////////////////////////////////////////
```

May 02, 02 0:13 **Table of Content** Page 1/1

Table of Contents