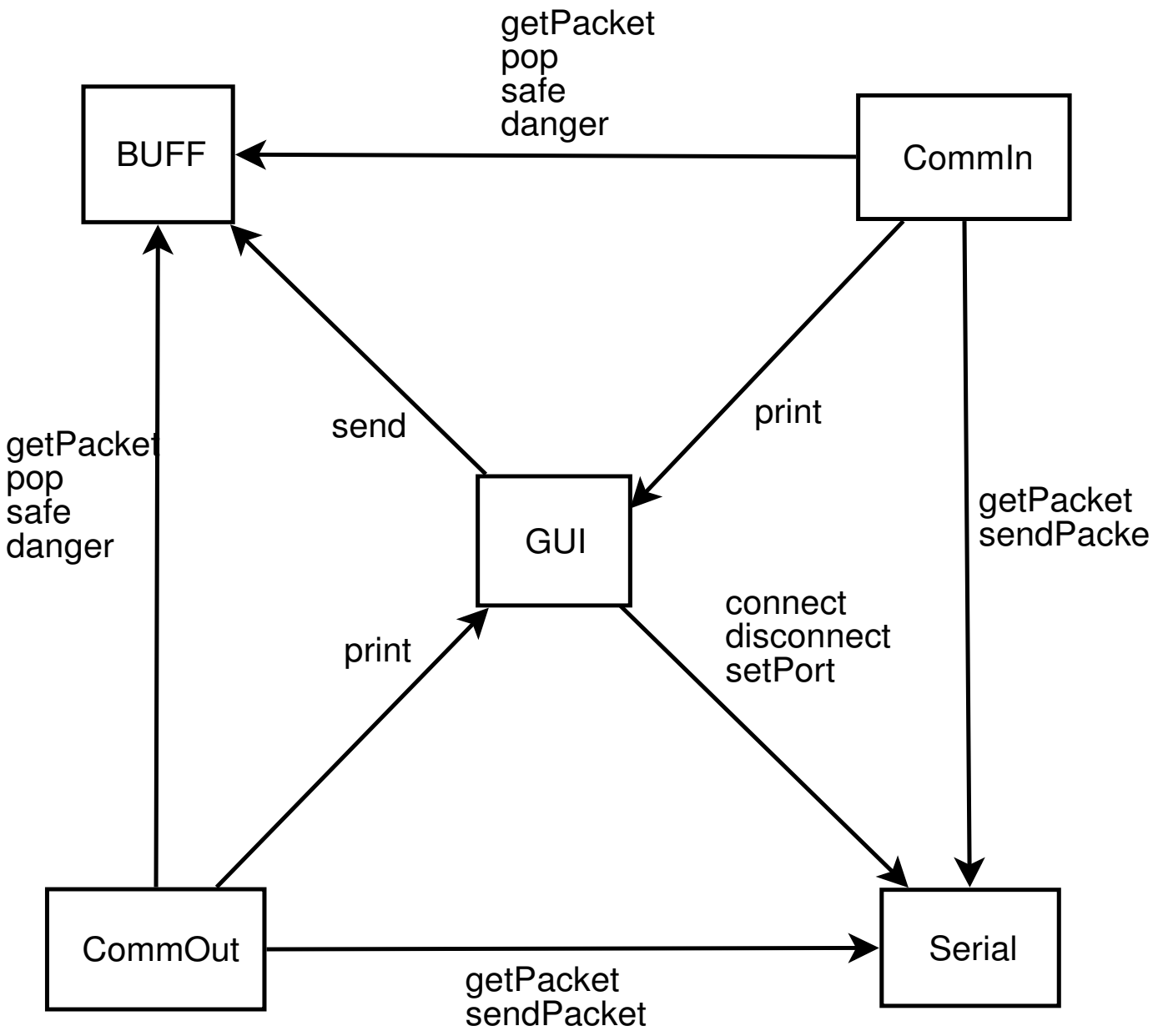
```
#include "Buffer.h"
#include "events.h"
#include "protocol.h"

Buffer::Buffer():count_(0)
{
    InitializeCriticalSection(&cs_);
    hEvent_ = CreateEvent(NULL, TRUE, FALSE, COMMOUT_START_EVENT);
}

void Buffer::send(Packet p) {
    EnterCriticalSection(&cs_);
    packetList_.push_front(p);

    if(count_ == 0) { // buffer is empty
        SetEvent(hEvent_);
    }

    count_++;
    LeaveCriticalSection(&cs_);
}

Packet Buffer::peek() {
    EnterCriticalSection(&cs_);
    Packet p = packetList_.back(); // Kyle is a goof
    LeaveCriticalSection(&cs_);
    return p;
}

void Buffer::pop() {
    EnterCriticalSection(&cs_);
    packetList_.pop_back();

    count_--;
    LeaveCriticalSection(&cs_);
}

bool Buffer::danger() {
    return count_ >= BUFFMAX;
}

bool Buffer::safe() {
    return count_ <= BUFFMIN;
}

bool Buffer::empty() {
    if(count_==0)
        return true;
```

```
        return false;
    }
```

```
#ifndef BUFFER_H
#define BUFFER_H

#include <windows.h>
#include <stdio.h>
#include <list>
#include "packet.h"

//when buffer reaches
#define BUFFMAX 30
// send RVI

#include "packet.h"
//when buffer reaches
#define BUFFMIN 5
// end RVI

class Buffer {
public:
    Buffer();
    ~Buffer(){ DeleteCriticalSection(&cs_); }
    void send(Packet p);    // add packet on the list
    Packet peek();          // returns next packet on list
    void pop();             // takes packet off list
    bool danger();          // send rvi
    bool safe();            // end rvi
    bool empty();
    std::list<Packet> packetList_;
private:
    CRITICAL_SECTION cs_;
    int count_;
    HANDLE hEvent_;
};

#endif BUFFER_H
```

Class Interaction Diagram



```
/*
*****************************************************************************
* CommOut.cpp                                                               *
* Purpose: Sending completed packets from the buffer to the serial port.    *
* Additionally, CommOut handles the RVI process of receiving data and       *
* sending that to the GUI.                                                  *
* Author: Max Wardell                                                       *
* Version: 1.0                                                              *
*****************************************************************************
*/

#include "CommOut.h"

/*Purpose: Constructor for the CommOut object.
* Parameters: Buffer *buffer: A pointer to the Buffer object.
*             Serial *serial: A pointer to the Serial object.
*/
CommOut::CommOut(Buffer *buffer, Serial *serial, Controller *gui): buffer_(buffer),
                                                                   serial_(serial),
                                                                   gui_(gui) {
}

////Outbuff gets populated, this function is called.
///*Purpose: Connects to the receiver*/
void CommOut::ConnectClient() {
    //TODO: Try this up here, then delete it.
    serial_->sendPacket(Packet(ENQ));
    Sleep(100);
    SendPacket(); //Send the packet
}

//Outbuff gets populated, this function is called.
/*Purpose: Connects to the receiver*/
////void CommOut::ConnectClient() {
////    //TODO: Try this up here, then delete it.
////    serial_->sendPacket(Packet(ENQ));
////
////    while(true) {
////        //serial_->sendPacket(Packet(ENQ));
////        try {
////            //packet_ = serial_->getPacket(TIMEOUT_TIME);//Grab the packet from the
serial port.
////            packet_ = serial_->getPacket(100);
////        }
////        catch (const int i) {         //Never received a packet, timeout.
////            //SetEvent(gotoIdle);
////            ENSURE_EXCEPTION(i, TIMEOUT_EXCEPTION);
////            throw GOTO_IDLE_EXCEPTION;
////        }
```

```
////        if(!packet_.valid())
////            continue; //Packet was invalid, wait for another
////        if(packet_.flags() == ACK0)    //Packet is an ACK0,
////            break; //Break out of loop and send packet.
////        if(packet_.flags() == ENQ) {
////            Sleep(1000);
////            throw GOTO_IDLE_EXCEPTION;
////        }
////    }
////    SendPacket(); //Send the packet
////}

/*Purpose: Sends the packet*/
void CommOut::SendPacket() {
    while(!buffer_->empty()) { //TODO: Change this condition
        serial_->sendPacket(buffer_->peek()); //SEND PACKET
        buffer_->pop();
    }
    Sleep(100);
    serial_->sendPacket(Packet(NTS)); //Send a NTS packet and
}

/*Purpose: Sends the packet*/
//void CommOut::SendPacket() {
//  int nackCount = 0, torCount = 0;
//  while(true) { //TODO: Change this condition
//      if(buffer_->empty()) {  //If the buffer is empty,
//          serial_->sendPacket(Packet(NTS)); //Send a NTS packet and
//          throw GOTO_IDLE_EXCEPTION; //break out of the Sending process
//      }
//      serial_->sendPacket(buffer_->peek()); //SEND PACKET
//      try {
//          /*Grab the control packet from the serial port. Possibilities:
//            ACK1 - Got the packet, keep sending them.
//            ACK0 - Got the packet, return to IDLE
//            NACK - Didn't receive the packet, resend. */
//          packet_ = serial_->getPacket(TIMEOUT_TIME);
//      }
//      catch (...) { //Never received a packet, timeout.
//          torCount++;
//          continue;
//          //ENSURE_EXCEPTION(i, TIMEOUT_EXCEPTION);
//          //throw GOTO_RESET_EXCEPTION;
//      }
//      if(packet_.valid()) {
//          if(packet_.flags() == NACK) {
//              nackCount++;
//          } else if(nackCount >= MAX_NCOUNT) { //NACK count maxed out or received an
ACK0; go back to IDLE.
```

```
//              throw GOTO_IDLE_EXCEPTION;
//          } else if(packet_.flags() == ACK0) {
//              buffer_->pop();
//              throw GOTO_IDLE_EXCEPTION;
//          } else if(torCount >= MAX_TCOUNT) {            //Timouts maxed out, reset
connection.
//              throw GOTO_RESET_EXCEPTION;
//          } else if(packet_.flags() == ACK1) {
//              buffer_->pop();
//              nackCount = 0;
//              torCount = 0;
//          } else if(packet_.flags() == RVI) { //Received an RVI; switch roles!
//              RVIProcess();
//          }
//      }
//  }
//}

/*Purpose: Receives packets when in RVI mode; displays to GUI.*/
void CommOut::RVIProcess() { //Occurs when an RVI is sent
    int timeoutCount = 0, nackCount = 0;
    buffer_->send(ACK1); //Send an acknoledgement, letting the sender know you received
the RVI
    //Connection Confirmed
    while(true) {
        try {
            packet_ = serial_->getPacket(TIMEOUT_TIME); //Grab the packet sent
        } catch (const int i) {
            ENSURE_EXCEPTION(i, TIMEOUT_EXCEPTION);
            if(++timeoutCount >= MAX_TCOUNT)
                throw GOTO_IDLE_EXCEPTION;
            continue;
        }
        if(packet_.valid()) { //If the packet is valid,
            // TODO: update function
            gui_->DisplayReceivedText(packet_.data().c_str());
            //serial_->sendPacket(Packet(ACK1));
            if(packet_.flags() == RVI) //If the packet is an RVI (second one)
                throw GOTO_IDLE_EXCEPTION;
            nackCount = 0; //Received a valid packet, reset NACK Counter
        }
        else { //If the packet is invalid,
            serial_->sendPacket(Packet(NACK));
            if(++nackCount >= MAX_NCOUNT) { //Keep count of how many NACKs you receive
in a row.
                throw GOTO_IDLE_EXCEPTION;
            }
        }
    }
```

```
}
```

```
#ifndef COMMOUT_H
#define COMMOUT_H

/*
*************************************************************************
* CommOut.cpp                                                           *
* Purpose: Sending completed packets from the buffer to the serial port.*
* Additionally, CommOut handles the RVI process of receiving data and   *
* sending that to the GUI.                                              *
* Author: Max Wardell                                                   *
* Version: 1.0                                                          *
*************************************************************************
*/

#include <windows.h>
#include "Buffer.h"
#include "crc.h"
#include "events.h"
#include "exceptions.h"
#include "packet.h"
#include "serial.h"
#include "utils.h"
#include "s_control.h"
#include "protocol.h"

class CommOut {
public:
    //Contructor, initializes pointers the the GUI, buffer and serial port.
    //CommOut(Buffer *buffer, GUI *gui, Serial *serial):buffer_(buffer), gui_(gui),
serial_(serial) {}
    CommOut(Buffer *buffer, Serial *serial, Controller * gui);
    void ConnectClient();
    void SendPacket();
    void RVIProcess();

private:
    Buffer *buffer_;
    Controller *gui_;
    Serial *serial_;
    Packet packet_;
};

#endif
```

Assignment #4

Wireless Communication

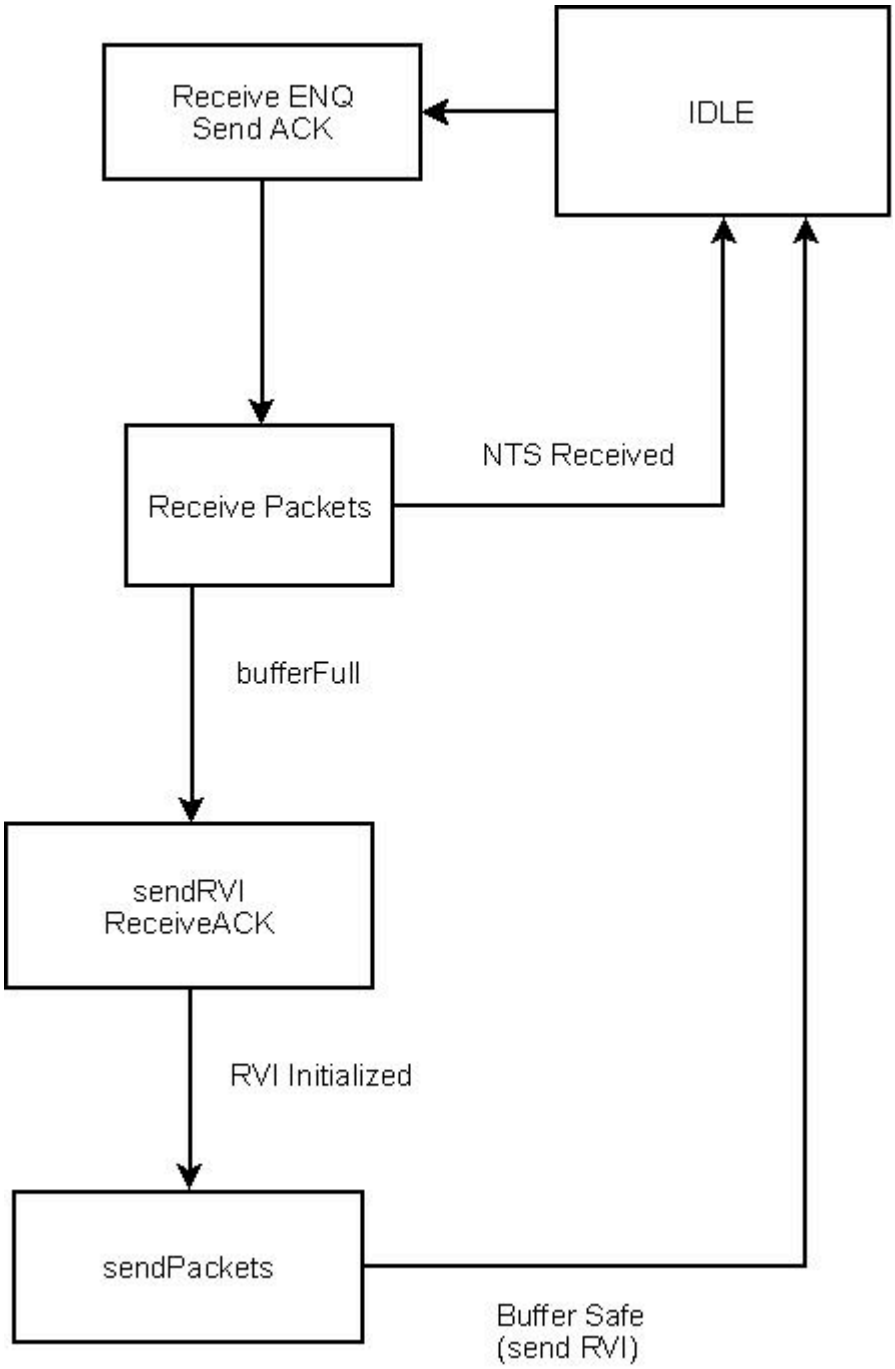
Doug Penner
Kyle Macdonald
Steffen L. Norgren
Max Wardell
Eddie Zhang

COMP 3980 • BCIT • December 5, 2008

```cpp
#include "crc.h"

using namespace std;

unsigned char CRC::calc(string in)
{
    unsigned char crc = 0;
    for (size_t i=0; i < in.length(); ++i) {
        crc = crcTable[crc ^ in[i]];
    }
    return crc;
}
```

```
#ifndef _CRC_H
#define _CRC_H

#include <string>

const unsigned char crcTable[] = {
      0,  94, 188, 226,  97,  63, 221, 131, 194, 156, 126,  32, 163, 253,  31, 65,
    157, 195,  33, 127, 252, 162,  64,  30,  95,   1, 227, 189,  62,  96, 130,220,
     35, 125, 159, 193,  66,  28, 254, 160, 225, 191,  93,   3, 128, 222,  60, 98,
    190, 224,   2,  92, 223, 129,  99,  61, 124,  34, 192, 158,  29,  67, 161,255,
     70,  24, 250, 164,  39, 121, 155, 197, 132, 218,  56, 102, 229, 187,  89,  7,
    219, 133, 103,  57, 186, 228,   6,  88,  25,  71, 165, 251, 120,  38, 196,154,
    101,  59, 217, 135,   4,  90, 184, 230, 167, 249,  27,  69, 198, 152, 122, 36,
    248, 166,  68,  26, 153, 199,  37, 123,  58, 100, 134, 216,  91,   5, 231,185,
    140, 210,  48, 110, 237, 179,  81,  15,  78,  16, 242, 172,  47, 113, 147,205,
     17,  79, 173, 243, 112,  46, 204, 146, 211, 141, 111,  49, 178, 236,  14, 80,
    175, 241,  19,  77, 206, 144, 114,  44, 109,  51, 209, 143,  12,  82, 176,238,
     50, 108, 142, 208,  83,  13, 239, 177, 240, 174,  76,  18, 145, 207,  45,115,
    202, 148, 118,  40, 171, 245,  23,  73,   8,  86, 180, 234, 105,  55, 213,139,
     87,   9, 235, 181,  54, 104, 138, 212, 149, 203,  41, 119, 244, 170,  72, 22,
    233, 183,  85,  11, 136, 214,  52, 106,  43, 117, 151, 201,  74,  20, 246,168,
    116,  42, 200, 150,  21,  75, 169, 247, 182, 232,  10,  84, 215, 137, 107, 53,
};


class CRC {
public:
    static unsigned char calc(std::string in);
};

#endif
```

Receiving STD:



Receiving Pseudo code:

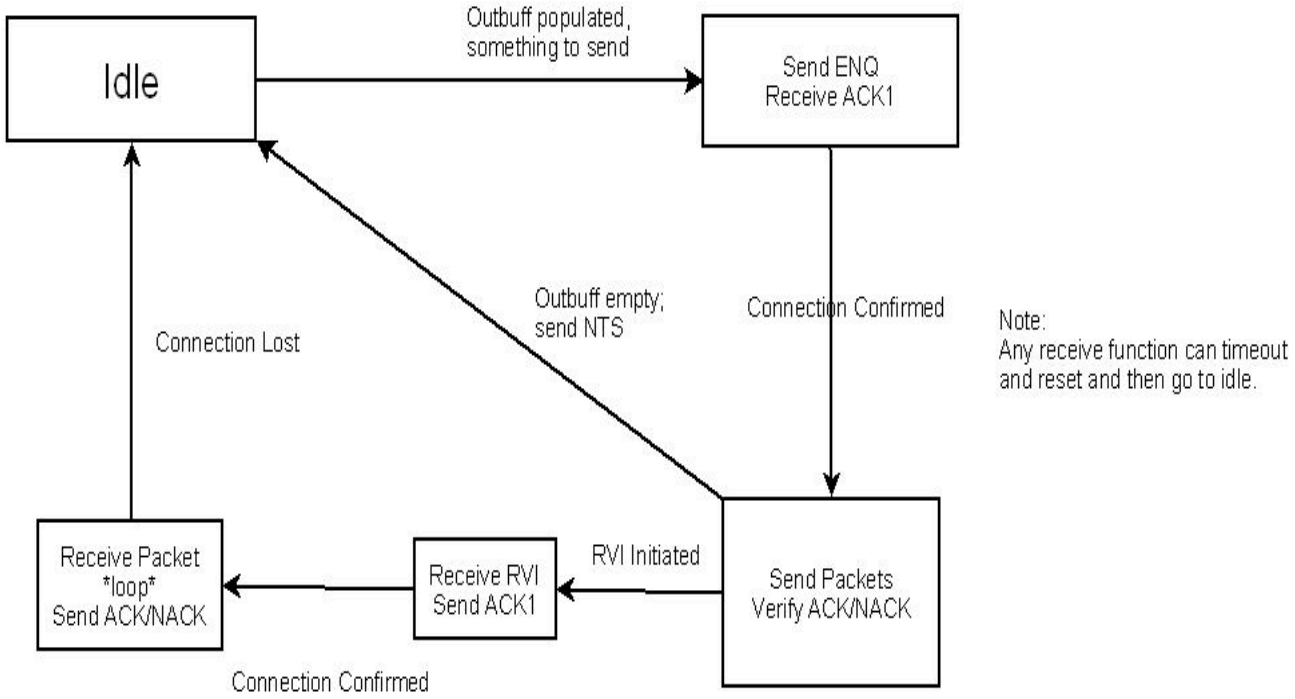
Receiver class is being called once an ENQ I received.
SendACK signal and prepare to recevie packets.

```
    While inbuff is not passed safe size
        keep receivinig packets
        if (inbuffer is not safe)
            start RVImode
```

RVI mode:

```
    sendRVI and wait for ACK1
    keep send packet until buffer is safe
    go back to Idle when buffer is safe.
```

Sending STD:



Sending Pseudo code:

```
//Outbuff gets populated, this function is called.
ConnectClient() {
        while forever
                Grab Packet;
                if timed out
                        continue;
                if packet is valid AND packet is an ACK1
                        break;
                else
                        continue;
        } //Connection Confirmed!
        SendPacket(packet);
}

SendPacket(Packet p) {
        while forever
                if outbuff is emtpy
                        p.send(NTS)
                        goto IDLE
                if RVI received
                        RVIProcess(packet);
                        break;
                p.send(data);
}

RVIProcess(Packet p) {
        p.send(ACK1)
        //Connection Confirmed
        while something to send
                Receive Packet
                if packet is valid
                        packet.send(ACK1)
                        if packet is an RVI
                                goto IDLE
                        NACK count = 0;
                else
                        packet.send(NACK)
                        NACK count++;
                        if NACK count >= 4
                                packet.send(NACK)
                                goto IDLE
}
```

Technical Report:

Challenges Encountered

1. **Time management – Trying to fit in time to work on this while working on other assignments at the same time.**
2. **Technical difficulties – Having strange errors that are very difficult to debug.**
3. **Transition between C and C++ - Inexperience with programming in C++ using the Win32 Library.**
4. **Under estimating project difficulty results in challenges of time usage on program testing.**
5. **Difficulties implementing the protocol into code.**
6. **Having the program alternate roles between sending and receiving.**
7. **Collaborating everyone's code together, then running and testing, which didn't work. What we could have done differently was start out simple send/receiving protocol, then work up from there.**

```
#ifndef _EVENTS_H
#define _EVENTS_H

#define GET_NEW_PACKET_EVENT    TEXT("getNewPacketEvent")
#define PACKET_FOUND_EVENT      TEXT("packetFoundEvent")
#define GLOBAL_DIE_EVENT        TEXT("globalDieEvent")
#define COMMIN_START_EVENT      TEXT("commInStartEvent")
#define COMMOUT_START_EVENT     TEXT("commOutStartEvent")
#define SERIAL_THREAD_DIE_EVENT TEXT("serialThreadDieEvent")

#endif
```

```
#ifndef _EXCEPTIONS_H
#define _EXCEPTIONS_H

#define TIMEOUT_EXCEPTION           1
#define PACKET_TOO_SMALL_EXCEPTION  2
#define GOTO_IDLE_EXCEPTION         3
#define GOTO_RESET_EXCEPTION        4
#define COMM_READ_EXCEPTION         5

#endif
```

```
#include <windows.h>
#include "idle.h"
#include "events.h"
#include "utils.h"

Idle::Idle(Buffer *buffer, Serial *serial, Controller *gui):
                commOut_(new CommOut(buffer, serial, gui)),
                commIn_(new Receiver(buffer, serial, gui)) {}

DWORD WINAPI Idle::thread(PVOID pvoid) {
    Idle *inst = (Idle*)pvoid;
    HANDLE listen[] = {
        CreateEvent(NULL, FALSE, FALSE, COMMOUT_START_EVENT),
        CreateEvent(NULL, FALSE, FALSE, PACKET_FOUND_EVENT),    // start receiving
        CreateEvent(NULL, FALSE, FALSE, GLOBAL_DIE_EVENT)
    };
    int who;


    while (true) {
        who = WaitForMultipleObjects(3, listen, FALSE, INFINITE);
        try {
            switch (who) {
                case WAIT_OBJECT_0+0:
                    Sleep(1);
                    inst->commOut_->ConnectClient();
                    break;
                case WAIT_OBJECT_0+1:
                    Sleep(1);
                    inst->commIn_->run();
                    break;
                case WAIT_ABANDONED:
                    return 0;
            }
        } catch (...) {
            //ENSURE_EXCEPTION(i, GOTO_IDLE_EXCEPTION);
        }
        for (int i=0; i < 3; ++i) {
            ResetEvent(listen[i]);
        }
    }
    return 0;
}
```

```
#ifndef _IDLE_H
#define _IDLE_H

#include "serial.h"
#include "Buffer.h"
#include "CommOut.h"
#include "Receiver.h"
#include "s_control.h"

class Idle {
public:
    Idle(Buffer *buffer, Serial *serial, Controller *gui);
    static DWORD WINAPI thread(PVOID pvoid);
private:
    CommOut *commOut_;
    Receiver *commIn_;
};

#endif
```

```
#include "message.h"
using namespace std;

void Message::ChoosePort() {
    MessageBox(NULL, TEXT("No port selected, please choose a port."), TEXT(""), MB_OK);
}

void Message::FailedPort() {
    MessageBox(NULL, TEXT("Could not open port. Try another port."), TEXT(""), MB_OK);
}

void Message::ErrorFailedPort() {
    MessageBox(NULL, TEXT("Error opening COM port."), TEXT(""), MB_OK);
}

void Message::FailedSavePort() {
    MessageBox(NULL, TEXT("Error, could not save port."), TEXT(""), MB_OK);
}

void Message::FailedConnection() {
    MessageBox(NULL, TEXT("Could not connect."), TEXT(""), MB_OK);
}

void Message::Connect() {
    MessageBox(NULL, TEXT("Successfully connected!."), TEXT(""), MB_OK);
}

void Message::DConnect() {
    MessageBox(NULL, TEXT("Successfully disconnected!."), TEXT(""), MB_OK);
}

void Message::WriteError() {
    MessageBox(NULL, TEXT("Error, could not write character."), TEXT(""), MB_OK);
}

void Message::CntChngSettings() {
    MessageBox(NULL, TEXT("Cannot change settings while in Connect Mode.\nPlease
disconnect first."), TEXT(""), MB_OK);
}

void Message::ReadThreadError() {
MessageBox(NULL, TEXT("Error creating READ thread"), TEXT("") ,MB_OK);
}

void Message::HelpAbout() {
    MessageBox(NULL,
        TEXT("Welcome to the RFID Scanner, Assignment #3 for COMP 3980\n\nAuthors:  Doug
Penner\n\tKyle Macdonald\n\tSteffen L. Norgren\n\tMax Wardell\n\tEddie Zhang"),
        TEXT("About RFID Scanner"), MB_OK);
```

```
}

void Message::Debug(string str) {
    LPCSTR result = str.c_str();
    MessageBox(NULL, result, TEXT(""), MB_OK);
}
```

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

class Message {
public:
    static void ChoosePort();
    static void FailedPort();
    static void ErrorFailedPort();
    static void FailedSavePort();
    static void FailedConnection();
    static void Connect();
    static void DConnect();
    static void WriteError();
    static void CntChngSettings();
    static void ReadThreadError();
    static void HelpAbout();
    static void Message::Debug(std::string str);
};
```

```
/*
    MODULE: model.h

    PURPOSE: Window model

    AUTHORS: Doug Penner
             Kyle Macdonald
             Steffen L. Norgren
             Max Wardell
             Eddie Zhang
*/

#ifndef _MODEL_H_
#define _MODEL_H_

#include <string.h>

class Model {
    enum { TEXT_SIZE = 20 };
public:
    Model(char const * str) {
        DisplayText(str);
        _text[TEXT_SIZE] = '\0';
    }

    void DisplayText(char const * str) {
        strncpy_s(_text, str, TEXT_SIZE);
    }

    char const * GetText() const { return _text; }
    int GetLen() const { return (int)strlen(_text); }

private:
    char _text[TEXT_SIZE + 1];
};

#endif
```

```
#include <windows.h>
#include <string>
#include "packet.h"
#include "crc.h"
#include "utils.h"

using namespace std;

Packet::Packet(): packet_(""), length_(0) {}

Packet::Packet(const Packet& packet): packet_(packet.packet_), length_(packet.length_)
{}  // copy constructor

Packet::Packet(std::string data): packet_(data), length_(data.length() + 4) {
    // turn string into packet
    packet_.insert(0, 1, SOH);
    packet_.insert(1, 1, (char)data.length() + 4);
    packet_.insert(2, 1, NONE);
}

Packet::Packet(char flag): length_(minLength_) {
    packet_.insert(0, 1, SOH);
    packet_.insert(1, 1, (char)0x04);
    packet_.insert(2, 1, flag);
}

void Packet::seq(bool toggle) { // only call once on a packet!!!
    packet_[2] = (toggle) ? (packet_[2] | SEQ) : (packet_[2] & (!SEQ));     // set SEQ
bit
}

bool Packet::seq() {
    return (packet_[2] & SEQ) == SEQ;
}

void Packet::append(char c) {
    packet_.append(1, c);
}

bool Packet::complete() {
    return packet_.length() >= minLength_ && packet_.length() == packet_[1];
}

bool Packet::valid() {
    ENSURE_BOOL(packet_.length() >= minLength_);
    ENSURE_BOOL(packet_[0] == SOH);
    ENSURE_BOOL(packet_[1] == packet_.length());
    ENSURE_BOOL(CRC::calc(packet_.substr(0, packet_.length()-2)));
    return true;
```

```
}

int Packet::flags() {
    return (packet_[2] & (!SEQ));
}

string Packet::data() {
    return packet_.substr(4, packet_.length()-2);
}

void Packet::calcCRC() {
    if (packet_.length() == length_) {  // remove CRC
        packet_.erase(packet_.end());
    }
    // re-calculate CRC
    //packet_ += CRC::calc(packet_);    // re-calculate CRC
    packet_.insert(packet_.end(), 1, CRC::calc(packet_));
}

string Packet::toString() {
    calcCRC();
    // return packet
    return packet_;
}

void Packet::clear() {
    length_ = 0;
    packet_.clear();
}

int Packet::length() {
    return (int)length_;
}

bool Packet::cmd() {
    return length_ == minLength_;
}
```

```
#ifndef _PACKET_H
#define _PACKET_H

#include <string>

// Packet bytes
#define NONE    (char)0x00
#define SOH     (char)0x01

// Flags
#define ENQ     (char)0x80
#define ACK0    (char)0x40
#define ACK1    (char)0x20
#define NACK    (char)0x10
#define RVI     (char)0x08
#define SEQ     (char)0x04
#define CON     (char)0x02
#define NTS     (char)0x01

class Packet {
public:
    Packet();
    Packet(const Packet& packet);   // cpconst
    Packet(char flag);              // control packet
    Packet(std::string data);       // data packet (from GUI)
    void append(char c);
    bool valid();           // validates packet
    bool complete();        // checks if enough bits have been received (for input from
serial port)
    int flags();            // flags byte from packet (seq bits set to 0!!!)
    std::string data();     // data section from packet
    std::string toString(); // entire packet (for sending)
    void seq(bool toggle);  // sets the seq bit
    bool seq();             // checks the seq bit
    void clear();
    int length();
    bool cmd();
    void calcCRC();
private:
    std::string packet_;    // entire packet (header + data + crc)
    size_t length_;
    static const size_t minLength_  = 4;
};

#endif
```

```
#ifndef _PROTOCOL_H
#define _PROTOCOL_H

// TODO: shorten after testing
#define TIMEOUT_TIME    100
#define MAX_TCOUNT      4
#define MAX_NCOUNT      4

#endif
```

```
#include "Receiver.h"


//
//DWORD WINAPI Receiver::thread(PVOID pvoid){
//  Receiver *instance = (Receiver*)pvoid;
//
//
//  static HANDLE hEvent[
//
//
//  bool running = true;
//
//  hEvent[0] = CreateEvent(NULL, TRUE,FALSE,TEXT("START EVENT"));
//  hEvent[1] = CreateEvent(NULL, TRUE,FALSE,TEXT("DIE EVENT"));
//
//  while(running){
//      if (WaitForMultipleObjects(2, hEvent, FALSE, INFINITE) == 1);
//          break;
//      instance->run();
//  }
//  return 0;
//}

void Receiver::run() {
    Packet packet;
    int tCount = 0;

    pSerial_->sendPacket(Packet(ACK0));

    while(true){
        // TODO: check received packet vs RVI mode order
        if(pBuffer_->danger()) {
            enterRVIMode();
        }

        // Get Packet
        try{
            packet = pSerial_->getPacket(TIMEOUT_TIME);
        }catch(const int i) {
            ENSURE_EXCEPTION(i, TIMEOUT_EXCEPTION);
                throw GOTO_RESET_EXCEPTION;
        }

        // Check Packet
        if (!packet.valid()) {
            pSerial_->sendPacket(Packet(NACK));
        } else {
            if (!packet.cmd()) {
```

```
                pGUI_->DisplayReceivedText(packet.data().c_str());
                pSerial_->sendPacket(Packet(ACK1));
                continue;
            } else {
                if(packet.flags() == NTS) {
                    throw GOTO_IDLE_EXCEPTION;
                }
            }
        }
    }
}

void Receiver::enterRVIMode(){
    int flag;
    int tCount = 0 ;
    pSerial_->sendPacket(Packet(RVI));
    while(true){
        try{
            flag = pSerial_->getPacket(TIMEOUT_TIME).flags();
            if (flag == ACK1) {
                break;
            }
        }
        catch(int i){
            ENSURE_EXCEPTION(i, TIMEOUT_EXCEPTION);
                throw GOTO_RESET_EXCEPTION;

        }
    }

    while(!(pBuffer_->safe())) {
        pSerial_->sendPacket((pBuffer_->peek()));
        pBuffer_->pop();
    }
    if(pBuffer_->safe()) {
        throw GOTO_IDLE_EXCEPTION;
    }
}
```

```
#ifndef RECEIVER_H
#define RECEIVER_H
// TODO: remove these
//#define GOTO_IDLE_EXCEPTION 1000
//#define TIMEOUT_TIME 100


#include<windows.h>
#include "Buffer.h"
#include "s_control.h"
#include "utils.h"
#include "serial.h"
#include "exceptions.h"
#include "protocol.h"


class Receiver{
public:

    //static DWORD WINAPI thread(PVOID pvoid);
    Receiver();
    Receiver(Buffer *pBuffer, Serial *pSerial, Controller *pGUI):pBuffer_(pBuffer),
        pSerial_(pSerial),pGUI_(pGUI){}
    void run();
    void enterRVIMode();
    //void readPacket();

private:
    Buffer *pBuffer_;
    Serial *pSerial_;
    Controller  *pGUI_;
    CRITICAL_SECTION cs_;


};

#endif
```

```
/*
    MODULE: control.cpp

    PURPOSE: Manages GUI control functions.

    AUTHORS: Doug Penner
             Kyle Macdonald
             Steffen L. Norgren
             Max Wardell
             Eddie Zhang
 */

#include <string>

#include "s_control.h"
#include "s_main.h"
#include "s_resource.h"
#include "s_winMaker.h"
#include "serial.h"
#include "CommOut.h"
#include "Buffer.h"
#include "Receiver.h"
#include "idle.h"

using namespace std;

// Window Procedure
LRESULT CALLBACK MainWndProc(HWND hWnd, UINT message, WPARAM wParam, LPARAM lParam) {
    // Pointer to Controller is stored in Window
    static Buffer       *buffer;
    static Serial       *serial;
    static Controller   *gui;
    static Receiver     *commIn;
    static CommOut      *commOut;
    static Idle         *idle;
    static HANDLE hIdleThread;
    static HANDLE hGUIThread;
    static HANDLE hSerialThread;

    switch (message) {
        case WM_CREATE:
            try {
                // Intialize Classes
                buffer = new Buffer();
                serial = new Serial();
                gui = new Controller(hWnd, reinterpret_cast<CREATESTRUCT *>(lParam),
serial, buffer);
                commIn = new Receiver(buffer, serial, gui);
                commOut = new CommOut(buffer, serial, gui);
```

```
                idle = new Idle(buffer, serial, gui);

                // Store pointer to Controller inside Window
                gui->CreateChatWindow();
                gui->PopulateCOMPorts();
                gui->_fConnected = FALSE; // initial connection state

                // Start Threads
                hIdleThread     = CreateThread(NULL, 0, Idle::thread            , idle
    , 0, NULL);
                hGUIThread      = CreateThread(NULL, 0, Controller::TimerThread , gui
    , 0, NULL);
                //hSerialThread = CreateThread(NULL, 0, Serial::thread          , serial
    , 0, NULL);
            }
            catch (...) {
                ::MessageBox(NULL, TEXT("Initialization error."), TEXT(""), MB_OK);
                return -1;
            }
            return 0;

        case WM_SIZE:
            gui->Size(LOWORD(lParam), HIWORD(lParam));
            return 0;

        case WM_PAINT:
            gui->Paint();
            return 0;

        case WM_COMMAND:
            gui->Command(LOWORD(wParam));
            return 0;

        case WM_DESTROY:
            delete gui;
            delete buffer;
            delete serial;
            delete commIn;
            delete commOut;
            delete idle;
            return 0;
    }
    return ::DefWindowProc(hWnd, message, wParam, lParam);
}

Controller::Controller(HWND hWnd, CREATESTRUCT * pCreate, Serial * serial, Buffer *
buffer)
    :_hWnd(hWnd), _model("Generic"), _serial(serial), _buffer(buffer) {
}
```

```
Controller::~Controller() {
    ::PostQuitMessage(0);
}

void Controller::CreateChatWindow() {
    // Create the main chat window dialog
    HINSTANCE hInst = WinGetLong<HINSTANCE>(_hWnd, GWL_HINSTANCE);
    _hWndChat = CreateDialog(hInst, MAKEINTRESOURCE(IDD_DLG_CHAT), _hWnd, ChatDlgProc);
}

void Controller::Paint() {
    BeginPaint(_hWnd, &_paint);
    EndPaint(_hWnd, &_paint);
}

// Menu commands processing
void Controller::Command(int cmd) {
    switch (cmd) {
        // File Menu
        case ID_FILE_EXIT:
            ::SendMessage(_hWnd, WM_CLOSE, 0, 0L);
            break;

        // View Menu
        case ID_VIEW_CLEAR:
            ClearText(::GetDlgItem(_hWndChat, IDC_SNT_TEXT)); // Clears Sent Text
            ClearText(::GetDlgItem(_hWndChat, IDC_SND_TEXT)); // Clears Sending Text
            ClearText(::GetDlgItem(_hWndChat, IDC_RCVD_TEXT)); // Clears Received Text
            break;

        case ID_HELP_ABOUT: {
                // Instance handle is available through HWND
                //HINSTANCE hInst = WinGetLong<HINSTANCE>(_hWnd, GWL_HINSTANCE);
                //DialogBox(hInst, MAKEINTRESOURCE(IDD_ABOUT), _hWnd, AboutDlgProc);
            }
            break;
        case ID_HELP_USAGE:
            break;

        // Redirected from the chat dialog
        case IDC_BTN_CONDIS:
            DisplayReceivedText("Catcher!!");
            DisplaySentText("Pitcher!!");
            GetDlgItemText(_hWndChat, IDC_CMB_COMPORT, _szPort, sizeof(_szPort));
            if (_serial->connected()) {
                //HANDLE killThreads = CreateEvent(NULL, FALSE, FALSE, GLOBAL_DIE_EVENT)
;
                //SetEvent(killThreads);
```

```
                //Sleep(500);
                _serial->disconnect();
                ToggleConnect();
            }
            else if (_serial->connect(_szPort)) {
                ToggleConnect();
            }
            break;

        case IDC_BTN_SEND:
            if (::GetWindowTextLength(::GetDlgItem(_hWndChat,IDC_SND_TEXT)) != 0) {
                SendText();
            }
            break;
    }
}

BOOL CALLBACK ChatDlgProc(HWND hWnd, UINT message, WPARAM wParam, LPARAM lParam) {
    switch (message) {
        case WM_INITDIALOG:
            return TRUE;

        // Redirect these messages back to the main windows procedure
        case WM_COMMAND: {
                HWND hWndParent = ::GetParent(hWnd);
                ::SendMessage(hWndParent, message, wParam, lParam);
            }
            break;
        }
    return FALSE;
}

void Controller::PopulateCOMPorts() {
    char    szBuffer[20], szTemp[20]; // character buffers to deal with string concats
    int     maxPorts = 255;
    WORD    wCount;
    BOOL    bSuccess;
    HANDLE  hPort;

    strcpy_s(szTemp, 20, "COM");

    // Cycle through up to MAXPORTS COM ports
    for (wCount = 1; wCount < maxPorts + 1; wCount++) {
        wsprintf(szBuffer, "%s%d", szTemp, wCount);

        // try to open the port
        bSuccess = FALSE;
        hPort = ::CreateFile(szBuffer, GENERIC_READ | GENERIC_WRITE, 0, 0, OPEN_EXISTING
, 0, 0);
```

```
        if (hPort == INVALID_HANDLE_VALUE) {
            DWORD dwError = GetLastError();

            // Check to see if the error was because the port was in use or a general
failure
            if (dwError == ERROR_ACCESS_DENIED || dwError == ERROR_GEN_FAILURE) {
                bSuccess = TRUE;
            }
        }
        else {
            // The port was opened successfully
            bSuccess = TRUE;

            // Release the port handle
            CloseHandle(hPort);
        }

        // Add the COM port to the combo-box
        if (bSuccess) {
            ::SendDlgItemMessage(_hWndChat, IDC_CMB_COMPORT, CB_ADDSTRING, 0, (LPARAM)
(LPSTR)szBuffer);
        }
    }

    // Select the first COM port in the list
    ::SendDlgItemMessage(_hWndChat, IDC_CMB_COMPORT, CB_SETCURSEL, (WPARAM)0, 0L);

    // Update global COM port setting
    ::GetDlgItemText(_hWndChat, IDC_CMB_COMPORT, _szPort, sizeof(_szPort));
}

void Controller::SendText() {
    PSTR    pSntText;
    int     iSntTextLen;

    if (_serial->connected()) {
        iSntTextLen = GetWindowTextLength(GetDlgItem(_hWndChat,IDC_SND_TEXT));

        // Allocate memory for the extracted text
        pSntText = (PSTR) VirtualAlloc((LPVOID) NULL, (DWORD) (iSntTextLen + 1),
            MEM_COMMIT, PAGE_READWRITE);

        GetWindowText(GetDlgItem(_hWndChat,IDC_SND_TEXT), pSntText, iSntTextLen + 1);

        DisplaySentText(pSntText);

        // TODO: remove
        //_serial->sendPacket(Packet(string(pSntText)));
```

```
        Packet p(pSntText);
        _buffer->send(p);
        // :ODOT

        VirtualFree(pSntText, 0, MEM_RELEASE);
        ClearText(GetDlgItem(_hWndChat,IDC_SND_TEXT));
    }
}

// This is the function that anyone calls to display received text onto the screen.
void Controller::DisplayReceivedText(string text) {
    HWND hWndSnt = GetDlgItem(_hWndChat, IDC_RCVD_TEXT);

    // Check to see if the dialog is empty, if not, send CRLF.
    if (GetWindowTextLength(hWndSnt) == 0) {
        SendMessage(hWndSnt, EM_SETSEL, -1, 0);                 // Select zero chars
from the edit of the current text
        SendMessage(hWndSnt, EM_REPLACESEL, FALSE, (LPARAM)"> "); // Replace no chars at
the end with text specified
        SendMessage(hWndSnt, EM_SETSEL, -1, 0);
        SendMessage(hWndSnt, EM_REPLACESEL, FALSE, (LPARAM)text.c_str());
    }
    else {
        SendMessage(hWndSnt, EM_SETSEL, -1, 0);
        SendMessage(hWndSnt, EM_REPLACESEL, FALSE, (LPARAM)"\r\n> ");
        SendMessage(hWndSnt, EM_SETSEL, -1, 0);
        SendMessage(hWndSnt, EM_REPLACESEL, FALSE, (LPARAM)text.c_str());
    }
}

// This is the function that anyone calls to display sent text onto the screen.
void Controller::DisplaySentText(string text) {
    HWND hWndSnt = GetDlgItem(_hWndChat, IDC_SNT_TEXT);

    // Check to see if the dialog is empty, if not, send CRLF.
    if (GetWindowTextLength(hWndSnt) == 0) {
        SendMessage(hWndSnt, EM_SETSEL, -1, 0);                 // Select zero chars
from the edit of the current text
        SendMessage(hWndSnt, EM_REPLACESEL, FALSE, (LPARAM)"> "); // Replace no chars at
the end with text specified
        SendMessage(hWndSnt, EM_SETSEL, -1, 0);
        SendMessage(hWndSnt, EM_REPLACESEL, FALSE, (LPARAM)text.c_str());
    }
    else {
        SendMessage(hWndSnt, EM_SETSEL, -1, 0);
        SendMessage(hWndSnt, EM_REPLACESEL, FALSE, (LPARAM)"\r\n> ");
        SendMessage(hWndSnt, EM_SETSEL, -1, 0);
        SendMessage(hWndSnt, EM_REPLACESEL, FALSE, (LPARAM)text.c_str());
    }
```

```
}

void Controller::ClearText(HWND hDlg) {
    SetWindowText(hDlg, NULL);
}

void Controller::ToggleConnect() {
    if (!_fConnected) {
        SendDlgItemMessage(_hWndChat, IDC_BTN_CONDIS, WM_SETTEXT, (WPARAM)0, (LPARAM)
TEXT("Disconnect"));
        _fConnected = TRUE;
    }
    else {
        SendDlgItemMessage(_hWndChat, IDC_BTN_CONDIS, WM_SETTEXT, (WPARAM)0, (LPARAM)
TEXT("Connect"));
        _fConnected = FALSE;
    }

    // Reset timer state
    _wSeconds = 0;
}

void Controller::ToggleSending() {
    if (IsDlgButtonChecked(_hWndChat, IDC_RAD_SND)) {
        CheckDlgButton(_hWndChat, IDC_RAD_SND, 0);
    }
    else {
        CheckDlgButton(_hWndChat, IDC_RAD_SND, 1);
    }

    // Reset timer state
    _wSeconds = 0;
}

void Controller::ToggleReceiving() {
    if (IsDlgButtonChecked(_hWndChat, IDC_RAD_RCV)) {
        CheckDlgButton(_hWndChat, IDC_RAD_RCV, 0);
    }
    else {
        CheckDlgButton(_hWndChat, IDC_RAD_RCV, 1);
    }

    // Reset timer state
    _wSeconds = 0;
}

/*
    FUNCTION: TimerThread(LPVOID)
```

```
    PURPOSE: Simple timer thread to manage updating the activity timer

*/
DWORD WINAPI Controller::TimerThread(LPVOID pVoid) {
    Controller *inst = (Controller*)pVoid;
    char szBuffer[20], szTemp[20]; // character buffers to deal with string concats
    BOOL loop = TRUE;

    inst->_wSeconds = 0;

    while (loop) {
        strcpy_s(szTemp, 20, " s");
        wsprintf(szBuffer, "%d%s", inst->_wSeconds, szTemp);
        SetWindowText(GetDlgItem(inst->_hWndChat,IDC_LBL_TIMER), szBuffer);
        inst->_wSeconds++;
        Sleep(1000);
    }
    return 0;
}

// Simple About dialog box
BOOL CALLBACK AboutDlgProc(HWND hWnd, UINT message, WPARAM wParam, LPARAM lParam) {
    switch (message) {
        case WM_INITDIALOG:
            return TRUE;

        case WM_COMMAND:
            switch (LOWORD(wParam)) {
                case IDOK:
                    return TRUE;

                case IDCANCEL:
                    EndDialog(hWnd, 0);
                    return TRUE;
            }
            break;
        }
    return FALSE;
}

void Controller::Size(int cx, int cy) {
    // Main window size has changed, need to resize child windows
    RECT wRect;
    int minWidth = 436;

    // Set the dialog size
    ::GetWindowRect(_hWnd, &wRect);
    ::MoveWindow(_hWndChat, 0, 0, cx, cy, TRUE);
```

```
    // Set the group box size & move its elements
    ::MoveWindow(::GetDlgItem(_hWndChat,IDC_GRP_CONNECTION), 5, 5, cx - 10, 50, TRUE);
    ::MoveWindow(::GetDlgItem(_hWndChat,IDC_RAD_SND), cx - 80, 18, 65, 15, TRUE);
    ::MoveWindow(::GetDlgItem(_hWndChat,IDC_RAD_RCV), cx - 80, 35, 65, 15, TRUE);

    // Make sure our lables are positioned properly (top lable doesn't matter)
    ::MoveWindow(::GetDlgItem(_hWndChat,IDC_LBL_SNT), 5, (cy/4) + 90, 60, 20, TRUE);
    ::MoveWindow(::GetDlgItem(_hWndChat,IDC_LBL_SND), 5, (cy/2) + 112, 80, 20, TRUE);

    // Make sure our window elements resize properly
    if (cx <= minWidth && cx > X_MIN_SIZE) {
        ::MoveWindow(::GetDlgItem(_hWndChat,IDC_LBL_ACTIVITY), (minWidth/2) + 13, 28, 65
, 15, TRUE);
        ::MoveWindow(::GetDlgItem(_hWndChat,IDC_LBL_TIMER), (minWidth/2) + 81, 28, 45,
15, TRUE);

        ::MoveWindow(::GetDlgItem(_hWndChat,IDC_RCVD_TEXT), 5, 83, minWidth - 15, (cy/4)
+ 0, TRUE);
        ::MoveWindow(::GetDlgItem(_hWndChat,IDC_SNT_TEXT), 5, (cy/4) + 106, minWidth -
15, (cy/4) + 0, TRUE);
        ::MoveWindow(::GetDlgItem(_hWndChat,IDC_SND_TEXT), 5, (cy/2) + 128, minWidth -
85,
            (wRect.bottom - wRect.top) - (cy/2 + 180), TRUE);
        ::MoveWindow(::GetDlgItem(_hWndChat,IDC_BTN_SEND), minWidth - 75, (cy/2) + 128,
65,
            (wRect.bottom - wRect.top) - (cy/2 + 180), TRUE);
    }
    else {
        ::MoveWindow(::GetDlgItem(_hWndChat,IDC_LBL_ACTIVITY), (cx/2) + 13, 28, 65, 15,
TRUE);
        ::MoveWindow(::GetDlgItem(_hWndChat,IDC_LBL_TIMER), (cx/2) + 81, 28, 45, 15,
TRUE);

        ::MoveWindow(::GetDlgItem(_hWndChat,IDC_RCVD_TEXT), 5, 83, cx - 10, (cy/4) + 0,
TRUE);
        ::MoveWindow(::GetDlgItem(_hWndChat,IDC_SNT_TEXT), 5, (cy/4) + 106, cx - 10, (cy
/4) + 0, TRUE);
        ::MoveWindow(::GetDlgItem(_hWndChat,IDC_SND_TEXT), 5, (cy/2) + 128, cx - 80,
            (wRect.bottom - wRect.top) - (cy/2 + 180), TRUE);
        ::MoveWindow(::GetDlgItem(_hWndChat,IDC_BTN_SEND), cx - 70, (cy/2) + 128, 65,
            (wRect.bottom - wRect.top) - (cy/2 + 180), TRUE);
    }

    // Make sure we don't size the window beyond some minimum constraints
    if (cx <= X_MIN_SIZE && cy > Y_MIN_SIZE) {
        ::MoveWindow(_hWnd, wRect.left, wRect.top, X_MIN_SIZE + 10, wRect.bottom - wRect
.top, TRUE);
    }
    else if (cx <= X_MIN_SIZE && cy <= Y_MIN_SIZE) {
```

```
        ::MoveWindow(_hWnd, wRect.left, wRect.top, X_MIN_SIZE + 10, Y_MIN_SIZE + 10,
TRUE);
    }
    else if (cx > X_MIN_SIZE && cy <= Y_MIN_SIZE) {
        ::MoveWindow(_hWnd, wRect.left, wRect.top, wRect.right - wRect.left, Y_MIN_SIZE
+ 10, TRUE);
    }
}
```

```
/*
    MODULE: modelControl.h

    PURPOSE: Main controller

    AUTHORS: Doug Penner
             Kyle Macdonald
             Steffen L. Norgren
             Max Wardell
             Eddie Zhang
 */

#ifndef _CONTROL_H_
#define _CONTROL_H_

#include <windows.h>
#include "serial.h"
#include "Buffer.h"

class Model {
    enum { TEXT_SIZE = 20 };
public:
    Model(char const * str) {
        DisplayText(str);
        _text[TEXT_SIZE] = '\0';
    }

    void DisplayText(char const * str) {
        strncpy_s(_text, str, TEXT_SIZE);
    }

    char const * GetText() const { return _text; }
    int GetLen() const { return (int)strlen(_text); }

private:
    char _text[TEXT_SIZE + 1];
};

class Controller {
public:
    Controller(HWND hWnd, CREATESTRUCT * pCreate, Serial *serial, Buffer *buffer);
    ~Controller();
    static DWORD WINAPI TimerThread(PVOID pVoid);
    void Size(int x, int y);
    void Paint();
    void Command(int cmd);
    void CreateChatWindow();
    void PopulateCOMPorts();
    void SendText();
```

```
    void DisplayReceivedText(std::string);
    void DisplaySentText(std::string);
    void ClearText(HWND);
    void ToggleConnect();
    void ToggleSending();
    void ToggleReceiving();

    BOOL        _fConnected;
    WORD        _wSeconds;
    HWND        _hWndChat;
private:
    PAINTSTRUCT _paint;
    HWND        _hWnd;
    Model       _model;
    Serial *    _serial;
    Buffer *    _buffer;
    TCHAR       _szPort[10];
};

#endif
```

```
/*
    MODULE: main.cpp

    PURPOSE: Manages window creation and message processing

    AUTHORS: Doug Penner
             Kyle Macdonald
             Steffen L. Norgren
             Max Wardell
             Eddie Zhang
 */

#include "s_main.h"
#include "s_resource.h"
#include "s_winMaker.h"
#include <new.h>

int WINAPI WinMain(HINSTANCE hInst, HINSTANCE hPrevInst, char * cmdParam, int cmdShow) {
    try {
        // Create top window class
        TopWinClass topWinClass(ID_MAIN, hInst, MainWndProc);

        //Is there a running instance of this program?
        //  HWND hWndOther = topWinClass.GetRunningWindow();
        //if (hWndOther != 0) {
        //  ::SetForegroundWindow(hWndOther);
        //  if (::IsIconic(hWndOther)) {
        //      ::ShowWindow(hWndOther, SW_RESTORE);
        //  }
        //  return 0;
        //}
        topWinClass.Register();

        // Create top window
        ResString caption(hInst, ID_CAPTION);
        TopWinMaker topWin(topWinClass, caption);
        topWin.Create();
        topWin.Show(cmdShow);

        // The main message loop
        MSG  msg;
        int status;
        while ((status = ::GetMessage(&msg, 0, 0, 0)) != 0) {
            if (status == -1) {
                return -1;
            }
            ::TranslateMessage(&msg);
            ::DispatchMessage(&msg);
        }
```

```
            return (int)msg.wParam;
        }
        catch (...) {
            char buf[50];
            ::MessageBox(0, buf, "Exception", MB_ICONEXCLAMATION | MB_OK);
        }

        return 0;
    }
```

```
/*
    MODULE: main.h

    PURPOSE: Main WinProc definitions

    AUTHORS: Doug Penner
             Kyle Macdonald
             Steffen L. Norgren
             Max Wardell
             Eddie Zhang
 */

#if !defined MAIN_H
#define MAIN_H

#include <windows.h>

LRESULT CALLBACK MainWndProc(HWND hWnd, UINT message, WPARAM wParam, LPARAM lParam);
BOOL CALLBACK ChatDlgProc(HWND hWnd, UINT message, WPARAM wParam, LPARAM lParam);
BOOL CALLBACK AboutDlgProc(HWND hWnd, UINT message, WPARAM wParam, LPARAM lParam);
DWORD WINAPI TimerThread(LPVOID);

#endif
```

```
//{{NO_DEPENDENCIES}}
// Microsoft Visual C++ generated include file.
// Used by s_resource.rc
//
#define IDR_MENU                        102
#define IDD_DLG_CHAT                    103
#define ID_MAIN                         105
#define ID_CAPTION                      106
#define IDC_GRP_CONNECTION              1001
#define IDC_COMBO1                      1002
#define IDC_CMB_COMPORT                 1002
#define IDC_BTN_CONDIS                  1003
#define IDC_SNT_TEXT                    1005
#define IDC_RCVD_TEXT                   1006
#define IDC_SND_TEXT                    1008
#define IDC_BTN_SEND                    1009
#define IDC_RAD_SND                     1012
#define IDC_RAD_RCV                     1013
#define IDC_LBL_ACTIVITY                1014
#define IDC_LBL_TIMER                   1015
#define IDC_LBL_SNT                     1016
#define IDC_LBL_RCVD                    1017
#define IDC_LBL_SND                     1018
#define ID_HELP_ABOUT                   40004
#define ID_HELP_USAGE                   40005
#define ID_VIEW_CLEAR                   40006
#define ID_EDIT_FONT                    40007
#define ID_EDIT_COPY40008               40008
#define ID_EDIT_PASTE40009              40009
#define ID_FILE_EXIT                    40010
#define ID_EDIT_COPYSEL                 40011
#define ID_EDIT_PASTESEL                40012

// Next default values for new objects
//
#ifdef APSTUDIO_INVOKED
#ifndef APSTUDIO_READONLY_SYMBOLS
#define _APS_NEXT_RESOURCE_VALUE        106
#define _APS_NEXT_COMMAND_VALUE         40020
#define _APS_NEXT_CONTROL_VALUE         1019
#define _APS_NEXT_SYMED_VALUE           101
#endif
#endif
```

```
// Microsoft Visual C++ generated resource script.
//
#include "s_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
BEGIN
    "s_resource.h\0"
END

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

3 TEXTINCLUDE
BEGIN
    "\r\n"
    "\0"
END

#endif    // APSTUDIO_INVOKED
```

```
//////////////////////////////////////////////////////////////////////////////
//
// String Table
//

STRINGTABLE
BEGIN
    ID_MAIN                 "GenericClass"
    ID_CAPTION              "Wireless Communicator - DataComm Assignment #4"
END

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


//////////////////////////////////////////////////////////////////////////////
// English (Canada) resources

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

//////////////////////////////////////////////////////////////////////////////
//
// Dialog
//

IDD_DLG_CHAT DIALOGEX 0, 0, 396, 211
STYLE DS_ABSALIGN | DS_SETFONT | DS_FIXEDSYS | WS_CHILD | WS_VISIBLE | WS_CLIPSIBLINGS
FONT 8, "MS Shell Dlg", 0, 0, 0x0
BEGIN
    GROUPBOX        "Connection",IDC_GRP_CONNECTION,3,3,386,31
    COMBOBOX        IDC_CMB_COMPORT,46,15,48,80,CBS_DROPDOWNLIST | CBS_SORT | WS_VSCROLL
| WS_TABSTOP
    PUSHBUTTON      "Connect",IDC_BTN_CONDIS,98,14,50,14
    LTEXT           "COM Port:",IDC_STATIC,10,17,34,8
    LTEXT           "Received Text:",IDC_LBL_RCVD,4,41,50,8
    LTEXT           "Sent Text:",IDC_LBL_SNT,4,104,35,8
    LTEXT           "Text to Send:",IDC_LBL_SND,3,168,45,8
    PUSHBUTTON      "Send",IDC_BTN_SEND,339,178,50,26
    EDITTEXT        IDC_SND_TEXT,3,178,328,26,ES_MULTILINE | ES_AUTOHSCROLL | WS_VSCROLL
    EDITTEXT        IDC_SNT_TEXT,3,114,386,49,ES_MULTILINE | ES_AUTOHSCROLL |
ES_READONLY | WS_VSCROLL
    EDITTEXT        IDC_RCVD_TEXT,3,51,386,49,ES_MULTILINE | ES_AUTOVSCROLL |
ES_AUTOHSCROLL | ES_READONLY | WS_VSCROLL
    RADIOBUTTON     "Sending",IDC_RAD_SND,339,11,41,10
    RADIOBUTTON     "Receiving",IDC_RAD_RCV,339,21,47,10
```

```
    LTEXT           "Last Activity:",IDC_LBL_ACTIVITY,205,17,43,8
    LTEXT           "0 seconds",IDC_LBL_TIMER,250,17,33,8
END


//////////////////////////////////////////////////////////////////////////////
//
// Menu
//

ID_MAIN MENU
BEGIN
    POPUP "&File"
    BEGIN
        MENUITEM "E&xit",                       ID_FILE_EXIT
    END
    POPUP "&View"
    BEGIN
        MENUITEM "&Clear",                      ID_VIEW_CLEAR
    END
    POPUP "&Help"
    BEGIN
        MENUITEM "&About",                      ID_HELP_ABOUT
        MENUITEM "&Usage",                      ID_HELP_USAGE
    END
END


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

#ifdef APSTUDIO_INVOKED
GUIDELINES DESIGNINFO
BEGIN
    IDD_DLG_CHAT, DIALOG
    BEGIN
        LEFTMARGIN, 3
        RIGHTMARGIN, 389
        TOPMARGIN, 3
        BOTTOMMARGIN, 204
    END
END
#endif    // APSTUDIO_INVOKED

#endif    // English (Canada) resources
//////////////////////////////////////////////////////////////////////////////
```

```
#ifndef APSTUDIO_INVOKED
////////////////////////////////////////////////////////////////////////////
//
// Generated from the TEXTINCLUDE 3 resource.
//


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

```
/*
    MODULE: winMaker.cpp

    PURPOSE: Manages the creation and control of generic windows.

    AUTHORS: Doug Penner
             Kyle Macdonald
             Steffen L. Norgren
             Max Wardell
             Eddie Zhang
 */

#include "s_winMaker.h"

// String Resource
ResString::ResString (HINSTANCE hInst, int resId) {
    if (!::LoadString(hInst, resId, _buf, MAX_RESSTRING + 1)) {
        ::MessageBox(NULL, TEXT("Load String failed"), TEXT(""), MB_OK);
    }
}

WinSimpleClass::WinSimpleClass(int resId, HINSTANCE hInst)
    : _hInstance(hInst) {

    ResString resStr(hInst, resId);
    _name = resStr;
}

WinClass::WinClass(char const * className, HINSTANCE hInst, WNDPROC wndProc)
    : WinSimpleClass(className, hInst) {

    _class.lpfnWndProc = wndProc;
    SetDefaults();
}

WinClass::WinClass(int resId, HINSTANCE hInst, WNDPROC wndProc)
    : WinSimpleClass(resId, hInst) {

    _class.lpfnWndProc = wndProc;
    SetDefaults();
}

void WinClass::SetDefaults () {
    // Provide reasonable default values
    _class.cbSize          = sizeof(WNDCLASSEX);
    _class.style           = 0;
    _class.lpszClassName   = GetName();
    _class.hInstance       = GetInstance();
    _class.hIcon           = 0;
```

```
    _class.hIconSm          = 0;
    _class.lpszMenuName     = 0;
    _class.cbClsExtra       = 0;
    _class.cbWndExtra       = 0;
    _class.hbrBackground    = reinterpret_cast<HBRUSH>(COLOR_WINDOW + 1);
    _class.hCursor          = ::LoadCursor(0, IDC_ARROW);
}

HWND WinSimpleClass::GetRunningWindow() {
    HWND hWnd = ::FindWindow(GetName(), 0);

    if (::IsWindow(hWnd)) {
        HWND hWndPopup = ::GetLastActivePopup(hWnd);
        if (::IsWindow(hWndPopup)) {
            hWnd = hWndPopup;
        }
    }
    else {
        hWnd = 0;
    }

    return hWnd;
}

void WinClass::Register() {
    if (::RegisterClassEx(&_class) == 0) {
        ::MessageBox(NULL, TEXT("Internal error: RegisterClassEx failed."), TEXT(""),
MB_OK);
    }
}

// Makes top window class with icons and menu
TopWinClass::TopWinClass(int resId, HINSTANCE hInst, WNDPROC wndProc)
    : WinClass(resId, hInst, wndProc) {

    _class.lpszMenuName = MAKEINTRESOURCE(resId);
}

// The maker of a window of a given class
WinMaker::WinMaker(WinClass & winClass)
  : _hWnd(0),
    _class(winClass),
    _exStyle(0),            // extended window style
    _windowName(0),         // pointer to window name
    _style(WS_OVERLAPPED),  // window style
    _x(CW_USEDEFAULT),      // horizontal position of window
    _y(0),                  // vertical position of window
    _width(X_SIZE),         // window width
    _height(Y_SIZE),            // window height
```

```
    _hWndParent(0),          // handle to parent or owner window
    _hMenu(0),               // handle to menu, or child-window identifier
    _data(0)                 // pointer to window-creation data
{ }

void WinMaker::Create() {
    _hWnd = ::CreateWindowEx(
            _exStyle,
            _class.GetName(),
            _windowName,
            _style,
            _x,
            _y,
            _width,
            _height,
            _hWndParent,
            _hMenu,
            _class.GetInstance (),
            _data);

    if (_hWnd == 0) {
        ::MessageBox(NULL, TEXT("Internal error: Window Creation Failed."), TEXT(""),
MB_OK);
    }
}

void WinMaker::Show(int nCmdShow) {
    ::ShowWindow(_hWnd, nCmdShow);
    ::UpdateWindow(_hWnd);
}

// Makes top overlapped window with caption
TopWinMaker::TopWinMaker(WinClass & winClass, char const * caption)
    : WinMaker(winClass) {

    _style = WS_OVERLAPPEDWINDOW | WS_VISIBLE;
    _windowName = caption;
}
```

```
/*
    MODULE: winMaker.h

    PURPOSE: Window specific classes and templates

    AUTHORS: Doug Penner
             Kyle Macdonald
             Steffen L. Norgren
             Max Wardell
             Eddie Zhang
 */

#ifndef _WINMAKER_H_
#define _WINMAKER_H_

#define X_SIZE      640
#define Y_SIZE      400
#define X_MIN_SIZE  429
#define Y_MIN_SIZE  346

#include <windows.h>
#include <string>

// Allows us to retrieve predefined strings from the resource file
class ResString {
    enum { MAX_RESSTRING = 255 };
public:
    ResString(HINSTANCE hInst, int resId);
    operator char const *() const { return _buf; }
private:
    char _buf[MAX_RESSTRING + 1];
};

// Getting and Setting WindowLong: default is GWL_USERDATA
template <class T>
inline T WinGetLong(HWND hWnd, int which = GWL_USERDATA) {
    return reinterpret_cast<T>(::GetWindowLong(hWnd, which));
}

template <class T>
inline void WinSetLong (HWND hWnd, T value, int which = GWL_USERDATA) {
    ::SetWindowLong(hWnd, which, reinterpret_cast<long>(value));
}

// Use for built-in classes
class WinSimpleClass {
public:
    WinSimpleClass (char const * name, HINSTANCE hInst) : _name (name), _hInstance
(hInst) {}
```

```
    WinSimpleClass(int resId, HINSTANCE hInst);
    char const * GetName() const { return _name.c_str(); }
    HINSTANCE GetInstance() const { return _hInstance; }
    HWND GetRunningWindow();
protected:
    HINSTANCE   _hInstance;
    std::string _name;
};

class WinClass: public WinSimpleClass {
public:
    WinClass(char const * className, HINSTANCE hInst, WNDPROC wndProc);
    WinClass(int resId, HINSTANCE hInst, WNDPROC wndProc);
    void SetBgSysColor(int sysColor) {
        _class.hbrBackground = reinterpret_cast<HBRUSH>(sysColor + 1);
    }

    void Register();
protected:
    void SetDefaults();
    WNDCLASSEX  _class;
};

class TopWinClass: public WinClass {
public:
    TopWinClass(int resId, HINSTANCE hInst, WNDPROC wndProc);
};

class WinMaker {
public:
    WinMaker(WinClass & winClass);
    operator HWND() { return _hWnd; }
    void AddCaption(char const * caption) {
        _windowName = caption;
    }
    void AddSysMenu()       { _style |= WS_SYSMENU; }
    void AddVScrollBar()    { _style |= WS_VSCROLL; }
    void AddHScrollBar()    { _style |= WS_HSCROLL; }
    void Create();
    void Show(int nCmdShow = SW_SHOWNORMAL);
protected:
    WinClass &   _class;
    HWND         _hWnd;

    DWORD        _exStyle;       // extended window style
    char const * _windowName;    // pointer to window name
    DWORD        _style;         // window style
    int          _x;             // horizontal position of window
    int          _y;             // vertical position of window
```

```
    int           _width;         // window width
    int           _height;        // window height
    HWND          _hWndParent;    // handle to parent or owner window
    HMENU         _hMenu;         // handle to menu, or child-window identifier
    void        * _data;          // pointer to window-creation data
};

class TopWinMaker: public WinMaker {
public:
    TopWinMaker (WinClass & winClass, char const * caption);
};

#endif
```

```
#include <windows.h>
#include <string>
#include <sstream>
#include "serial.h"
#include "packet.h"
#include "exceptions.h"
#include "utils.h"
#include "message.h"

using namespace std;

// ========================= Serial ========================= //

/*
Constructs the serial port class
*/
Serial::Serial(): connected_(false), packetAvailable_(false) {
    InitializeCriticalSection(&portGuard_);
}

/*
Gets a new packet from the serial port.
Note: this function is called by Serial::thread!
*/
DWORD WINAPI Serial::thread(PVOID pvoid) {
    Serial *s = (Serial*) pvoid;

    bool live = true;

    bool success = false;
    char inbuff[256];
    DWORD nBytesRead, dwEvent;
    COMSTAT comstat;
    COMMTIMEOUTS timeOuts;
    OVERLAPPED osRead = {0};
    osRead.hEvent = CreateEvent(NULL, TRUE, FALSE, NULL);

    //Set the total timeout interval
    memset(&timeOuts, 0, sizeof(timeOuts));
    timeOuts.ReadTotalTimeoutMultiplier = 5;
    timeOuts.ReadTotalTimeoutConstant = 50;
    SetCommTimeouts(s->hComm_, &timeOuts);

    SetCommMask(s->hComm_, EV_RXCHAR);


    char err = 0;

    while(live) {   // wait for packet to be complete
```

```
        if(WaitCommEvent(s->hComm_, &dwEvent, NULL)) {
            ClearCommError(s->hComm_, NULL, &comstat);
            if ((dwEvent & EV_RXCHAR) && comstat.cbInQue) {
                if (ReadFile(s->hComm_, &inbuff, min(255, comstat.cbInQue), &nBytesRead,
&osRead)) {
                    if (nBytesRead) {
                        live = s->foundString(string(inbuff, nBytesRead));
                    }
                }
            }
        } else {
            err = (char)GetLastError();
            s->sendString(&err, 1);
            myvoid(err);
        }

        ResetEvent(osRead.hEvent);
    }
    PurgeComm(s->hComm_, PURGE_RXCLEAR);
    return 0L;
}

bool Serial::foundString(string str) {
    static HANDLE found = CreateEvent(NULL, FALSE, FALSE, PACKET_FOUND_EVENT);
    static HANDLE listen[3] = {
        listen[0] = CreateEvent(NULL, FALSE, FALSE, GET_NEW_PACKET_EVENT),
        listen[1] = CreateEvent(NULL, FALSE, FALSE, GLOBAL_DIE_EVENT),
        listen[2] = CreateEvent(NULL, FALSE, FALSE, SERIAL_THREAD_DIE_EVENT)
    };
    for (int i=0; i < str.length(); ++i) {
        if (packet_.length() == 0) {
            if (str[i] == SOH) {
                packet_.append(str[i]);
            }
        } else {
            packet_.append(str[i]);
            if (packet_.complete()){
                packetAvailable_ = true;
                SetEvent(found);
                if (WaitForMultipleObjects(3, listen, FALSE, INFINITE) == 1) {
                    return false;
                    break;
                }
                packet_.clear();
            }
        }
    }
    return true;
}
```

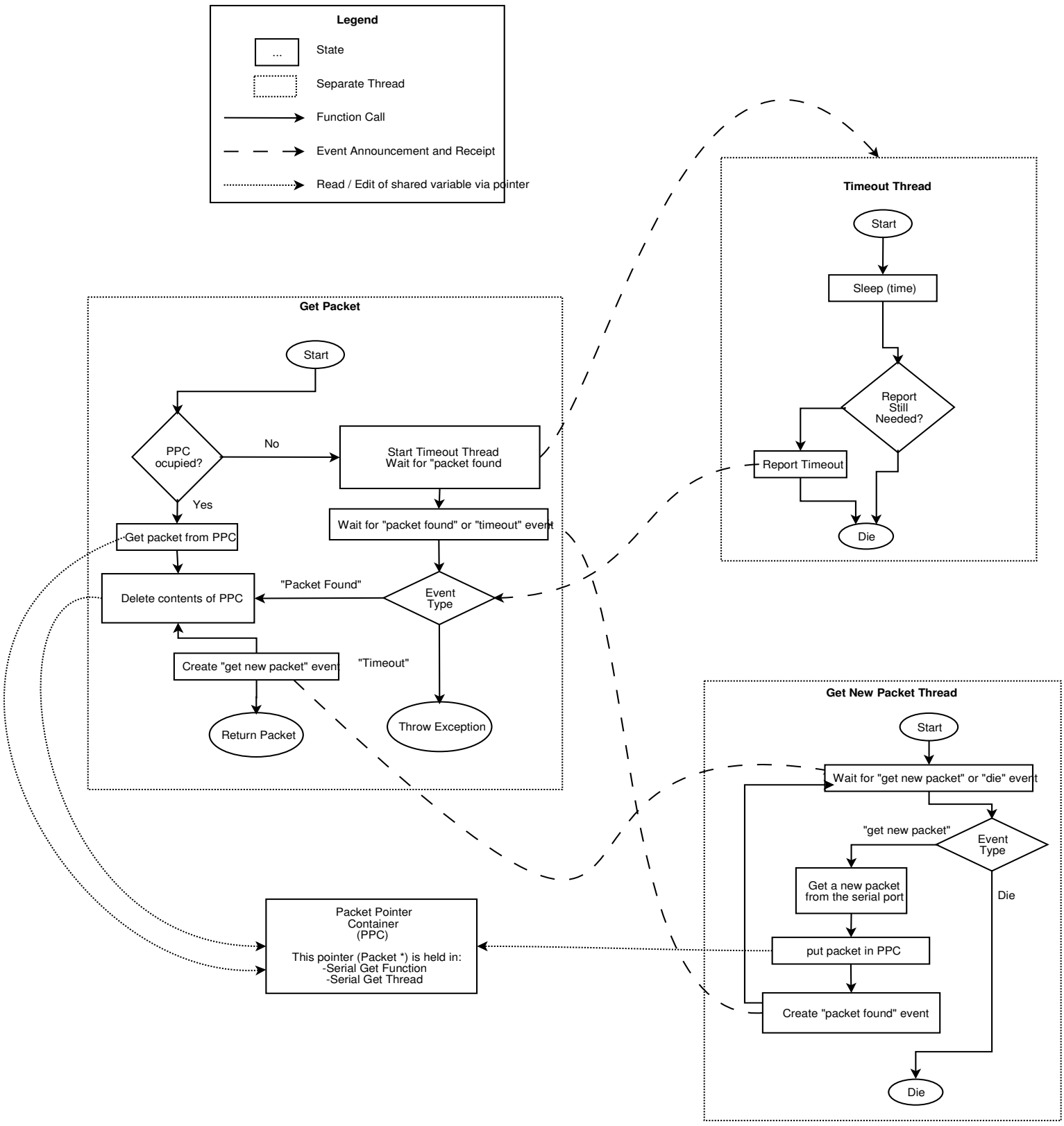
```
/*
Tries to get a packet within a certain timeframe
@throws TIMEOUT_EXCEPTION if no packet found
@param timeout - max time to wait for a packet
@returns the new packet
*/
Packet Serial::getPacket(int timeout) {
    timeout = 100;
    static HANDLE found = CreateEvent(NULL, FALSE, FALSE, PACKET_FOUND_EVENT);
    static HANDLE get   = CreateEvent(NULL, FALSE, FALSE, GET_NEW_PACKET_EVENT);
    if (!packetAvailable_) {
        //WaitForSingleObject(found, timeout);  // give the serial thread time to find a
packet
        WaitForSingleObject(found, 100);
    }
    ResetEvent(found);
    ENSURE_BOOL_THROW(packetAvailable_, TIMEOUT_EXCEPTION); // check if a packet has
been found yet
    Packet p(packet_);
    SetEvent(get);
    return p;
}

bool Serial::sendPacket(Packet& packet){
    Sleep(1);
    return sendString(packet.toString().c_str(), packet.length());
}

bool Serial::sendString(const char *str, size_t len) {
    bool retVal = true;
    EnterCriticalSection(&portGuard_);
    // Create Overlap
    OVERLAPPED osWrite = {0};
    DWORD dwWritten;
    //Create this write operation's OVERLAPPED structure's hEvent.
    osWrite.hEvent = CreateEvent(NULL, TRUE, FALSE, NULL);
    if(osWrite.hEvent == NULL) {
        retVal =  false;
    } else if(WriteFile(hComm_, str, (DWORD)len, &dwWritten, &osWrite)){
        CloseHandle(osWrite.hEvent);
    } else if (GetLastError() != ERROR_IO_PENDING) {
        retVal = false;
    }
    // Done
    LeaveCriticalSection(&portGuard_);
    return retVal;
}
```

```
bool Serial::connect(LPCTSTR port) {
    ENSURE_BOOL(!connected_);
    bool success = false;

    EnterCriticalSection(&portGuard_);

    GetCommState(hComm_, &dcb_);
    // setup settings
    FillMemory(&dcb_, sizeof(dcb_), 0);
    dcb_.BaudRate = CBR_9600;
    dcb_.Parity = NOPARITY;
    dcb_.StopBits = ONESTOPBIT;
    dcb_.ByteSize = 8;

    // Enables the COM port configuration
    SetCommState(hComm_, &dcb_);

    // connect
    if (port == NULL) {
        Message::ChoosePort();
    } // Only change made here was to change FILE_FLAG_OVERLAPPED to
FILE_ATTRIBUTE_NOMAL
    else if((hComm_ = CreateFile(port, GENERIC_READ | GENERIC_WRITE, 0,
                NULL, OPEN_EXISTING, FILE_FLAG_OVERLAPPED, NULL)) == // changed from
FILE_FLAG_OVERLAPPED
                INVALID_HANDLE_VALUE) {
        Message::ErrorFailedPort();
    }
    else if(!SetCommState(hComm_, &dcb_)) {
        Message::FailedSavePort();
    }
    else {
        success = true;
    }

    // clean up
    LeaveCriticalSection(&portGuard_);
    connected_ = success;
    CreateThread(NULL, 0, Serial::thread, this, 0, NULL);
    return success;
}

bool Serial::disconnect() {
    HANDLE die = CreateEvent(NULL, FALSE, FALSE, SERIAL_THREAD_DIE_EVENT);
    SetEvent(die);
    ENSURE_BOOL(connected_);
    PurgeComm(hComm_, PURGE_FLAGS);
    CloseHandle(hComm_);
    connected_ = false;
```

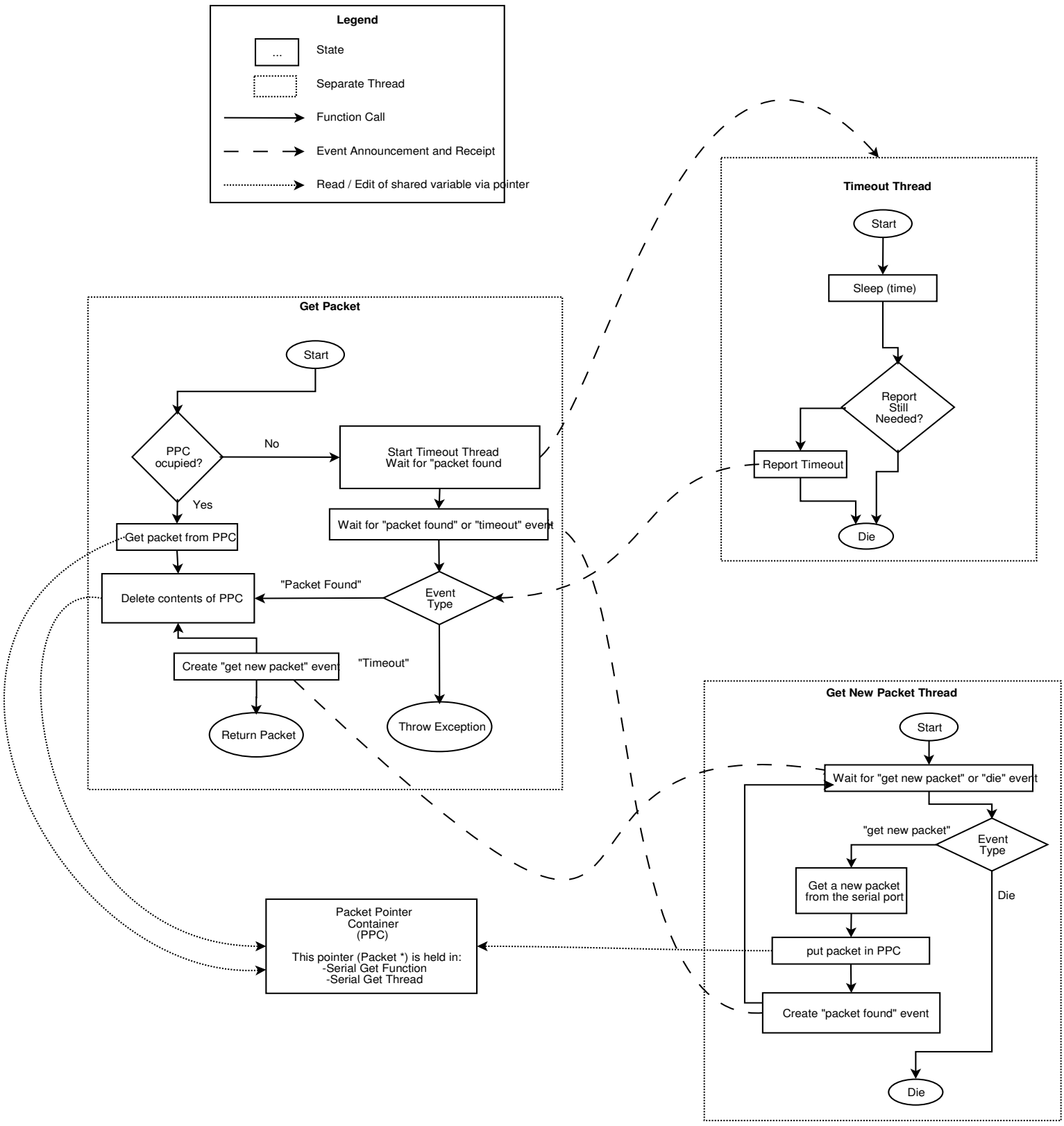
```
    return true;
}

bool Serial::connected() {
    return connected_;
}
```

```
#ifndef _SERIAL_H
#define _SERIAL_H

#include <windows.h>
#include "packet.h"
#include "events.h"

#define PURGE_FLAGS PURGE_TXABORT | PURGE_TXCLEAR | PURGE_RXABORT | PURGE_RXCLEAR

class Serial {
public:
    bool connected_;
    Serial();
    static DWORD WINAPI thread(PVOID pvoid);
    void getNewPacket();
    char getNextChar();
    Packet getPacket(int timeout);
    bool sendPacket(Packet& packet);
    bool Serial::sendString(const char *str, size_t len);
    friend DWORD WINAPI serialThread(PVOID pvoid);
    bool connect(LPCTSTR port);
    bool disconnect();
    bool connected();
    void getString(std::string& data8);
    bool foundString(std::string s);
private:
    CRITICAL_SECTION portGuard_;
    Packet  packet_;
    bool    packetAvailable_;
    // Physical Port
    HANDLE hComm_;
    COMMPROP commProp_;
    COMMCONFIG commConfig_;
    DCB dcb_;
    // Thread
    HANDLE  thread_;
};

#endif
```

Packet Receiver Function State Machine



Packet Receiver Function State Machine



```
#ifndef _UTILS_H
#define _UTILS_H

#define ENSURE_BOOL(cond)       if (!(cond)) return false
#define ENSURE_BOOL_THROW(cond, exc) if (!(cond)) throw exc
#define ENSURE_EXCEPTION(i, exc) if(i != exc) throw exc

void myvoid(int i);

#endif

void myvoid(int i) {}
```