```
/*
    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);
    }
}
```