READING DATA FROM ARDUINO USING C++

Pre-requisites

- ✓ HANDLE: Simply put, it can reference or point to any object and abstract it to such an extent that, the programmer can work with that object without knowing much about it. But, a handle is not just a pointer; it can refer to a memory location that stores a pointer object (which, in turn, is pointing to some other object) [1][2].
- ✓ COMSTAT: A structure defined in the W*indows.h* header to store all the information about a communicating device. For example, cbInQue is a member of this structure which gives the number of bytes available to be read from the device [3].
- ✓ DCB: A structure also defined in *Windows.h* that has members which define the "control settings" for communicating with a serial communication device. Some example members are BaudRate, ByteSize, Parity, StopBits [4].
- ✓ SERIAL: It is a custom class written for coordinating the communication with the serial device. It contains all necessary functions to setup/terminate connection, read/write data from/to the device and manage possible errors in the interfacing. As in general C++ coding, all declarations are done in a separate header file *SerialClass.h* and the constructor is written in the *Serial.cpp* file (which includes the header) [5].

Functions

- CreateFile( ): Creates or opens a file or I/O device. The function returns a handle that can be used to access the file or device for various types of I/O depending on the file or device and the flags and attributes specified. Defined in *Windows.h* header [6].
- GetLastError( ): Returns the error that last occurred with the device. Defined in *Windows.h* header [7].
- GetCommState( ): Retrieves the current control settings for a specified communications device. Returns "false" unless successful. Defined in *Windows.h* [8].
- SetCommState( ): SETs the control settings (passed as a DCB parameter to it) for a specified communications device. Returns "false" unless successful. Defined in *Windows.h* [9].
- CloseHandle( ): Closes the handler passed as the argument. Defined in *Windows.h*.
- ClearCommError( ): The function is called when a communications error occurs, and it clears the device's error flag to enable additional input and output (I/O) operations. Defined in *Windows.h* header [10].
- ReadData(*char* *buffer, *unsigned in*t nbChar): Read "nbChar" (or the number available) bytes from the device into the space pointed by "buffer". Custom defined.
- WriteData(*char* *buffer, *unsigned int* nbChar): Write "nbChar" bytes to the device from the data pointed by "buffer". Custom defined function.
- IsConnected( ): Returns the state of the connection. Custom defined function.

The Process (in Windows)

- ✓ Connect the Arduino to any USB port of the computer and allow it to install the necessary drivers. Once done, note down the COM port assigned to it: Go to Start->Device Manager->Ports (COM & LPT) and find out something like *Arduino* there.
- ✓ Open Eclipse and create a new C++ project with three files (as described later in the tutorial) namely *main.cpp*, *Serial.cpp* and, *SerialClass.h.* Change the COM port specified in *main.cpp* to the one applicable to your computer. Then, Build Project.
- ✓ If using the MinGW toolchain, set the PATH in the *Environment Variables* tab under *Run Configurations* as the install path in your computer (typically, "C:\MinGW\bin").
- ✓ Upload the sketch *ArduinoSerialTest* (found later in this tutorial) into the Arduino.
- ✓ Then, Run the Eclipse project that was just created. The string "Hey there Mr. Computer" will be seen printed repeatedly in the console which is the string being transmitted by the Arduino through the serial interface. The Arduino sketch can be easily modified to transmit any other data which can be parsed suitably in C++.

Setting up the Serial Communication with Arduino [5]

- Instantiate the *Serial* class in *main.cpp* (create an object of class *Serial* by passing the applicable COM port as the argument in the format "\\.\COMx" -change x suitably).
- Initialize the handle *hSerial* using the CreateFile( ) function which again requires passing the COM port as argument. Check if the handle initialized properly.
- If no, declare error and exit; if yes, proceed and try to get the current communication parameters using the GetCommState( ) function.
- If the function returns *false*, declare error and exit; if it returns *true*, set the new communication parameters, to the DCB object, as required for the interfacing. Ex. BaudRate = 9600, ByteSize = 8, StopBits = ONESTOPBIT, Parity = NOPARITY.
- Now, try and set these parameters to the communication interface using the SetCommState( ) function. If *false* returned, declare error and exit; if *true* returned, set the Boolean variable *connected* to TRUE and wait for 2 seconds to start data transfer.

Serial.cpp

- In *main.cpp*, check if the variable *connected* is TRUE using the IsConnected( ) function and proceed with data reception only if it is TRUE; else, exit the program.
- To start receiving data, declare a character array to store the data. "While" IsConnected( ) returns TRUE, read data from the interface up to the size of the array using the ReadData( ) function defined in *Serial.cpp*.
- Parse the received data suitably or try to typecast it to any suitable data type (as required in the application) and print it when ReadData( ) returns a positive value (meaning that some valid data was indeed received).
- Wait for at least 300 milliseconds until the next read. This limits speed of data transmission.
- Include suitable *cout* statements in the code to debug state of the connection.

Code [5]

SerialClass.h:

```
#ifndef SERIALCLASS_H_INCLUDED
#define SERIALCLASS_H_INCLUDED

#define ARDUINO_WAIT_TIME 2000

#include <windows.h>
#include <stdio.h>
#include <stdlib.h>

class Serial
{
    private:
        //Serial comm handler
        HANDLE hSerial;
        //Connection status
        bool connected;
        //Get various information about the connection
        COMSTAT status;
        //Keep track of last error
        DWORD errors;

    public:
        //Initialize Serial communication with the given COM port
        Serial(char *portName);
        //Close the connection
        //NOTE: for some reason you can't connect again before exiting
        //the program and running it again
        ~Serial();
        //Read data in a buffer, if nbChar is greater than the
        //maximum number of bytes available, it will return only the
        //bytes available. The function return -1 when nothing could
        //be read, the number of bytes actually read.
        int ReadData(char *buffer, unsigned int nbChar);
        //Writes data from a buffer through the Serial connection
        //return true on success.
        bool WriteData(char *buffer, unsigned int nbChar);
        //Check if we are actually connected
        bool IsConnected();


};

#endif // SERIALCLASS_H_INCLUDED
```

Serial.cpp:

```
#include "SerialClass.h"

Serial::Serial(char *portName)
{
    //We're not yet connected
    this->connected = false;
```

```
    //Try to connect to the given port through CreateFile
    this->hSerial = CreateFile(portName,
            GENERIC_READ | GENERIC_WRITE,
            0,
            NULL,
            OPEN_EXISTING,
            FILE_ATTRIBUTE_NORMAL,
            NULL);

    //Check if the connection was successfull
    if(this->hSerial==INVALID_HANDLE_VALUE)
    {
        //If not successful display an Error
        if(GetLastError()==ERROR_FILE_NOT_FOUND){

            //Print Error if neccessary
            printf("ERROR: Handle was not attached. Reason: %s not available.\n",
portName);

        }
        else
        {
            printf("ERROR!!!");
        }
    }
    else
    {
        //If connected we try to set the comm parameters
        DCB dcbSerialParams = {0};

        //Try to get the current
        if (!GetCommState(this->hSerial, &dcbSerialParams))
        {
            //If impossible, show an error
            printf("failed to get current serial parameters!");
        }
        else
        {
            //Define serial connection parameters for the arduino board
            dcbSerialParams.BaudRate=CBR_9600;
            dcbSerialParams.ByteSize=8;
            dcbSerialParams.StopBits=ONESTOPBIT;
            dcbSerialParams.Parity=NOPARITY;

             //Set the parameters and check for their proper application
             if(!SetCommState(hSerial, &dcbSerialParams))
             {
                printf("ALERT: Could not set Serial Port parameters");
             }
             else
             {
                 //If everything went fine we're connected
                 this->connected = true;
                 //We wait 2s as the arduino board will be reseting
                 Sleep(ARDUINO_WAIT_TIME);
             }
        }
    }

}
```

```
Serial::~Serial()
{
    //Check if we are connected before trying to disconnect
    if(this->connected)
    {
        //We're no longer connected
        this->connected = false;
        //Close the serial handler
        CloseHandle(this->hSerial);
    }
}

int Serial::ReadData(char *buffer, unsigned int nbChar)
{
    //Number of bytes we'll have read
    DWORD bytesRead;
    //Number of bytes we'll really ask to read
    unsigned int toRead;

    //Use the ClearCommError function to get status info on the Serial port
    ClearCommError(this->hSerial, &this->errors, &this->status);

    //Check if there is something to read
    if(this->status.cbInQue>0)
    {
        //If there is we check if there is enough data to read the required number
        //of characters, if not we'll read only the available characters to
prevent
        //locking of the application.
        if(this->status.cbInQue>nbChar)
        {
            toRead = nbChar;
        }
        else
        {
            toRead = this->status.cbInQue;
        }

        //Try to read the require number of chars, and return the number of read
bytes on success
        if(ReadFile(this->hSerial, buffer, toRead, &bytesRead, NULL) && bytesRead
!= 0)
        {
            return bytesRead;
        }

    }

    //If nothing has been read, or that an error was detected return -1
    return -1;

}


bool Serial::WriteData(char *buffer, unsigned int nbChar)
{
    DWORD bytesSend;

    //Try to write the buffer on the Serial port
    if(!WriteFile(this->hSerial, (void *)buffer, nbChar, &bytesSend, 0))
    {
        //In case it don't work get comm error and return false
        ClearCommError(this->hSerial, &this->errors, &this->status);
```

```
        return false;
    }

    else
        return true;
}

bool Serial::IsConnected()
{
    //Simply return the connection status
    return this->connected;
}
```

Main.cpp:

```
#include <stdio.h>
#include <tchar.h>
#include "SerialClass.h"  // Library described above
#include <string>
#include <iostream>
#include <string>
#include <stdlib.h>

using namespace std;

// application reads from the specified serial port and reports the collected data


int _tmain(int argc, _TCHAR* argv[])
{
	cout << "Connecting to arduino..." << endl;

	Serial* SP = new Serial("\\\\.\\COM7");    // adjust as needed

	if (SP->IsConnected())
		cout << "Connection established with Arduino!!" << endl;   // Let us know the serial is connected

	char incomingData[256] = "";                  // don't forget to pre-allocate memory
	int dataLength = 256;
	int readResult = 0;

	while(SP->IsConnected())
	{
		readResult = SP->ReadData(incomingData,dataLength);
		//cout << "Bytes read: " << readResult << endl;

		string test(incomingData);

		if (readResult > 0) {

			cout << incomingData << endl;
			test = "";
		}

		Sleep(300);
	}
	cin >> argc;
	return 0;
}
```

The ArduinoSerialTest sketch:

```
void setup()                    // run once, when the sketch starts
{
  Serial.begin(9600);           // set up Serial library at 9600 bps
}

int i = 0;

void loop()                       // run over and over again
{
  delay(3000);
  Serial.println("Hey there Mr. Computer");

  /*while(1)
  {
    Serial.println(i);
    delay(100);
    i++;
  }*/

}
```

References

1. Stanley B.Lippman, Josée Lajoie and Barbara E.Moo, *C++ Primer*, 5th Edition

2. *What is a handle in C++*, Hyperlink: http://stackoverflow.com/questions/1303123/what-is-a-handle-in-c, updated on Aug. 19th, 2009.

3. *COMSTAT Structure*, Hyperlink: http://msdn.microsoft.com/en-us/library/windows/desktop/aa363200(v=vs.85).aspx, updated on July 25th, 2013.

4. *DCB Structure*, Hyperlink: http://msdn.microsoft.com/en-us/library/windows/desktop/aa363214(v=vs.85).aspx, updated on July 25th, 2013.

5. *Interfacing Arduino and C++ (for Windows)*, Hyperlink: http://playground.arduino.cc/Interfacing/CPPWindows.

6. *CreateFile Function*, Hyperlink: http://msdn.microsoft.com/en-us/library/windows/desktop/aa363858(v=vs.85).aspx, updated on Aug. 26th, 2013.

7. *GetLastError Function*, Hyperlink: *http://msdn.microsoft.com/en-us/library/windows/apps/ms679360(v=vs.85).aspx*, updated on Sept. 9th, 2013.

8. *GetCommState Function*, Hyperlink: http://msdn.microsoft.com/en-us/library/windows/desktop/aa363260(v=vs.85).aspx, updated on July 25th, 2013.

9. *SetCommState Function*, Hyperlink: http://msdn.microsoft.com/en-us/library/windows/desktop/aa363436(v=vs.85).aspx, updated on July 25th, 2013.

10. *ClearCommError Function*, Hyperlink: http://msdn.microsoft.com/en-us/library/windows/desktop/aa363180(v=vs.85).aspx, updated on July 25th, 2013.