CSerial - A C++ Class for Serial Communications

Posted by **Tom Archer and Rick Leinecker** on **August 7th, 1999**
This article was contributed by Tom Archer and Rick Leinecker.

Preface

This class is meant as a very simple alternative to the more robust, feature-rich CSerialPort class presented by Remon Spekreijse. In other words, if you need a very simple class to read or write data to the serial port, then this class might be perfect for you. However, if you need more control over just how the serial communications is to be conducted, then Remon's very fine class will probably be what you want.

Introduction

Common uses of remote communications are business applications that link differnet sites in order to keep tabs on inventories and transactions. For example, I once wrote a large drug-dispensing proram in which many clinic sites automatically received new orders and updated pharmacy information each evening from a centry host location. The centry host gets inventory levels and transaction information from each site during evening hours. A large part of my time was spent construction telecommunications routines that sent and received information packets. To that extent, I offer this simple CSerial class in the hopes that it will save someone from the grungy world of serial communications that I had to endure and will free your mind up to concentrate on the problem domain.

CSerial class member functions

- **CSerial::CSerial()** -
- **CSerial::CSerial()** - Basic c'tor that takes no arguments.
- **CSerial::Open(int nPort = 2, int nBaud = 9600)** - This member function is used to open the serial port. It takes two interger arguments. The first argument contains the port number where the valid entries are 1 through 4. The second argument is the baud rate. Valid values for this argument are 1200, 2400, 4800, 9600, 19200, 38400 and 76800. This function returns TRUE if successful. Otherwise, it returns a value of FALSE.
- **CSerial::Close()** - While the d'tor will automatically close the serial port for you, this function has been added just in case there is a reason that you need to explicit close the port.
- **CSerial::SendData(const char *, int)** - This function writes data from a buffer to the serial port. The first argument it takes is a const char* to a buffer that contains the data being sent. The second argument is the number of bytes being sent. This function will return the actual number of bytes that are succesfully transmitted.
- **CSerial::ReadDataWaiting(void)** - This function simply returns the number of bytes that waiting in the communication port's buffer. It basically allows you to "peek" at the buffer without actually retrieving the data.
- **CSerial::ReadData(void*, int)** - This function reads data from the port's incoming buffer. The first argument that it takes is a void* to a buffer into which the data will be placed. The second argument is

an integer value that gives the size of the buffer. The return value of this function contains the number of bytes that were successfully read into the provided data buffer.

Example Usage

Here are some examples of how easy it is to use this class.

Opening the serial port

```

CSerial serial;
if (serial.Open(2, 9600))
 AfxMessageBox("Port opened successfully");
else
 AfxMessageBox("Failed to open port!");
```

Sending Data

```

CSerial serial;
if (serial.Open(2, 9600))
{
 static char* szMessage[] = "This is test data";
 int nBytesSent = serial.SendData(szMessage, strlen(szMessage));
 ASSERT(nBytesSent == strlen(szMessage));
}
else
 AfxMessageBox("Failed to open port!");
```

Reading Data

```

CSerial serial;
if (serial.Open(2, 9600))
{
 char* lpBuffer = new char[500];
 int nBytesRead = serial.ReadData(lpBuffer, 500);
 delete []lpBuffer;
}
else
 AfxMessageBox("Failed to open port!");
```

Downloads

Download source - 2 Kb