Model Railroad System
A collection of utilities for Model Railroaders
C Internals: C/MRI

Robert Heller
Deepwoods Software
Wendell, MA, USA

March 16, 2004

This documentation was prepared with LaTeX.
This document describes version 2 of the Model Railroad System package.

The author, Robert Heller, may be contacted electronically (E-Mail) via the following:

FidoNet 1:321/153, Locks Hill BBS.

InterNet heller@deepsoft.com

Web site URL: `http://www.deepsoft.com/`.

Contents

1

C/MRI C++ Serial Port Interface.

This is a Linux implementation of Bruce Chubb's C/MRI[1] QBASIC[7] serial port code ported to C++. This code works with 2.2 kernels and GLIBC 2.1 (RedHat 6.2) and 2.4 kernels and GLIBC 2.2 (RedHat 7.3). And it can use any serial port device supported by these kernels. That is, in addition to the standard four COM ports, it can also use the various supported multi-port cards as well.

The code is presently "hardwired" to use the Linux termios interface. I wanted to get the code up and running and presently I don't have any machines running other operating systems to test other low-level terminal I/O code. MS-Windows users do have access to Bruce Chubb's C/MRI QBasic and Visual Basic code, so there is no rush at this point to support MS-Windows, although for MS-Windows who might want to use my forthcoming Tcl/Tk MRI code I'll probably want to port this code to run under MS-Windows. This header and the class interface specification won't change much. There will probably be lots of fun with ifdef in the C++ file. Since this is open source code, I would hope that some enterprising MS-Windows C++ programmer will take up the "gauntlet" and do the MS-Windows port. (Ditto for MacOSX and FreeBSD programmers.)

Basically, the way this code works is to use a class (described on page 1) to interface to the serial port, which may have one or more serial port cards (a mix of USICs, SUSICs, and SMINIs). A given class instance interfaces to all of the cards on attached to a given serial port. There are three public member functions, one to initialize a given board (described on page 17), one to set the output ports (described on page 16), and one to poll the state of the input ports (described on page 15).

I was inspired to write this code after reading the four part series in *Model Railroader*[3, 4, 5, 6] and reading the download package for the SMINI card[2]. I already have a copy of Bruce Chubb's *Build Your Own Universal Computer Interface*, but the SMINI looks like a great option for small "remote" locations of a layout where there are a few turnouts and a some signals, such as a small junction, interchange yard, or isolated industrial spur.

2

class **List**

Public Members

Private Members

A C++ mapping for a Tcl list. Re-sizable (manually only). Used to pass lists of integers (such as port values) to and from the low-level code, where these values are encoded and decoded to/from the serial interface cards.

2.1

List (int l=0)

Arguments

The constructor. Construct a vector of a specific number of elements.

2.1.1

int **l**

The number of elements to allocate.

2.2

~List ()

The destructor. Free up memory.

2.3

int **Length** () const

The `Length()` member function returns the length of the vector.

2.4

int & operator **[]** (int i)

Arguments

2.4.1 int **i** 5

Read/write indexing accessor. Returns a reference to the i^{th} element.

2.4.1

int **i**

The index to access.

2.5

int operator **[]** (int i) const

Arguments

Read only indexing accessor. Returns the value of the i^{th} element.

2.5.1

int **i**

The index to access.

2.6

void **Resize** (int l)

Arguments

The `Resize()` member function re-sizes the vector. If the vector is shortened, elements off the end are discarded.

2.6.1

int **l**

The number of elements to allocate.

2.7

int **length**

Length of the allocated vector.

2.8

int * **elements**

Vector of elements.

3

enum **CardType**

Members

Card type codes.

3.1

USIC

Classic Universal Serial Interface Card.

3.2

SUSIC

Super Classic Universal Serial Interface Card.

3.3

SMINI

Super Mini node.

4

enum **ASCIICtrlCodes**

Members

Special ASCII codes used in the data-link.

4.1

STX

Start of Text. Used at the start of message blocks.

4.2

ETX

End of text. Used at the end of message blocks.

4.3

DLE

Data Link Escape. Used to escape special codes.

5

enum **MessageTypes**

Members

Message type codes.

5.1

Init

Initialize message. Initialize a serial interface board.

5.2

Transmit

Transmit message. Send data to output ports.

5.3

Poll

Poll message. Request the board to read its input ports.

5.4

Read

Read message. Generated by a board in response to a Poll message.

6

class **CMri**

Public Members

Private Members

Main C/MRI interface class. This class implements the interface logic for all of the boards on a given serial bus, attached to a given serial (COM) port. This class effectively implements in C++ under Linux what the QBasic serial I/O subroutines implemented by Bruce Chubb implement under MS-Windows.

The constructor opens the serial port and does low-level serial I/O setup (BAUD rate, etc.). This is the first part of the `INIT` subroutine.

The `InitBoard()` member function initializes a selected board (the second part of the `INIT` subroutine) and the `Inputs()` and `Output()` member functions correspond to the `INPUTS` and `OUTPUTS` subroutines.

The private members, `transmit()` and `readbyte()` correspond to the TXPACK and RXBYTE subroutines.

All of the public member functions can take a pointer to a place to store an allocated (with `new()`) string containing an error message. If `NULL` is passed, no error reporting is done–error checking is still done, just that the calling program gets no indication of it, except that the `Inputs()` function will return `NULL`. If the message pointer is non `NULL`, a `char` array is allocated with `new()` and this array is filled with an error message. The calling program should delete this memory when it is done with it, otherwise the calling program will leak memory. If there are no errors, this pointer is not changed. The calling program should initialize this pointer to `NULL` and then test for a non `NULL` value as an indication of a possible error.

6.1

CMri (const char *port="/dev/ttyS0", int baud=9600, int maxtries=10000, char **outmessage=NULL)

Arguments

The constructor opens the serial port and initializes the port.

6.1.1

const char * **port**

The serial port device file.

6.1.2

int **baud**

The desired BAUD rate.

6.1.3

int **maxtries**

The maximum number of retries.

6.1.4

char ** **outmessage**

This holds a pointer to an error message, if any.

6.2

~CMri ()

The destructor restores the serial port’s state and closes it.

6.3

List * **Inputs** (int ni, int ua=0, char **outmessage=NULL)

Arguments

The `Inputs()` function polls the interface and collects the input port values returned by the serial card.

The result is a freshly allocated `List` object. The calling program should free this memory with `delete()`. `Inputs()` returns a `NULL` pointer if there was an error.

6.3.1

int **ni**

The number of input ports to be read. Must equal the number of ports on the specified card.

6.3.2

int **ua**

The card address.

6.3.3

char ** **outmessage**

This holds a pointer to an error message, if any.

6.4

void **Outputs** (const List *ports, int ua=0, char **outmessage=NULL)

Arguments

The `Outputs()` function sends bytes to the output ports managed by the specified card. Since each element is written to one 8-bit output port, each element is presumed to be a integer in the range of 0 to 255.

6.4.1

const List * **ports**

The list of port values. Should have as many elements as there are output ports.

6.4.2

int **ua**

The card address.

6.4.3

char ** **outmessage**

This holds a pointer to an error message, if any.

6.5

void **InitBoard** (const List *CT, int ni, int no, int ns=0, int ua=0, CardType card=SMINI, int dl=0, char **outmessage=NULL)

Arguments

The `InitBoard()` function initializes a given USIC, SUSIC, or SMINI card.

6.5.1

const List * **CT**

The card type / yellow bi-color LED map. For USIC and SUSIC cards this is the card type map. For the SMINI card this is a 6 element list containing the port pairs for any simulated yellow bi-color LEDs.

The card type map for USIC and SUSIC is a packed array of 2-bit values, packed 4 per element (byte) from low to high. Each 2-bit value is one of 0 (for no card), 1 (for an input card), or 2 (for an output card). The cards must be “packed” with no open slots except at the end of the bus.

For the simulated yellow LEDs (SMINI card) the paired bits must be adjacent red/green bits and cannot span ports.

6.5.2

int **ni**

The total number of input ports (must be 3 for SMINI).

6.5.3

int **no**

The total number of output ports (must be 6 or SMINI).

6.5.4

int **ns**

The number of yellow bi-color LED signals. Only used for SMINI cards. For USIC and SUSIC cards the Length() member function of the `CT` parameter is used.

6.5.5

int **ua**

The card address.

6.5.6

CardType **card**

The card type.

6.5.7

int **dl**

The delay value to use.

6.5.8

char ** **outmessage**

This holds a pointer to an error message, if any.

6.6

int **ttyfd**

Terminal file descriptor.

6.7

struct termios **savedtermios**

Saved serial port settings.

6.8

struct termios **currenttermios**

Current serial port settings.

6.9

int **MaxTries**

Maximum number of input I/O retries.

6.10

bool **transmit** (int ua, char mt, unsigned char ob[], int lm)

Arguments

Data transmitter. The data is built into a proper message and sent out the serial port to the selected card. Returns `false` on error and `true` on success.

6.10.1

int **ua**

The card address.

6.10.2

char **mt**

The message type.

6.10.3

unsigned char **ob[]**

The data buffer (not used for Poll messages).

6.10.4

int **lm**

The length of the data buffer (pass 0 for Poll messages).

6.11

bool **readbyte** (unsigned char& thebyte)

Arguments

Read a single byte from the serial interface. Used by the `Inputs()` function. Returns `false` on error and `true` on success.

6.11.1

unsigned char& **thebyte**

A place to put the byte read. Undefined if there was an error.

7

References

[1] Bruce Chubb. *Build Your Own Universal Computer Interface*. Tab Books, 1989.

[2] Bruce Chubb. Downloadable smini information package. On the web at the URL: `http://www.jlcenterprises.net/Files/SMINI%20Information%20Package.zip`., 2004.

[3] Bruce Chubb. Signaling make easier, part 1. *Model Railroader*, pages 130–134, Janurary 2004.

[4] Bruce Chubb. Signaling make easier, part 2. *Model Railroader*, pages 68–73, Feburary 2004.

[5] Bruce Chubb. Signaling make easier, part 3. *Model Railroader*, pages 86–91, March 2004.

[6] Bruce Chubb. Signaling make easier, part 4. *Model Railroader*, pages 78–81, April 2004.

[7] Bruce Chubb. Ustpqb.txt. On the web at the URL: `http://www.jlcenterprises.net/Files/USTPQB.TXT`., 2004.

Class Graph