General
All notes assume a program in the following form

```
#include "scribbler.h"
#include "system.h"

int         gDebugging = 0;

//WinSerial        port(ADDRESS);                  /* Windows with Real Robot */
//PosixSerial      port(ADDRESS);                  /* Linux or Mac with Real Robot */
ClientSerial       port(ADDRESS);                  /* Linux, Mac, Windows with Simulated Robot */

Scribbler      robot(port);

/*********************************************************************/
/*                                                                   */
/*  main                                                             */
/*                                                                   */
/*********************************************************************/


int main( int argc, char *argv[])
{
          if (argc > 1)
            gDebugging = atoi(argv[1]);

          /* example code begins     */



          /* example code ends       */

          return 0;
}
```

Objects

The scribbler interface is built around a small number of Objects that allow the programmer to use the capabilities without needing to know the details.

Scribbler - maintains the dialog with the Scribbler/Fluke pairs
WinSerial - serial communication in Windows OS
PosixSerial - serial communication in a UNIX like OS
ClientSerial - client side of a socket

The Scribbler object uses one of the Serial objects to communicate with the Scribbler/Fluke

Some complex data, particularly the JPEG photos, are managed by the interface.
To avoid the user having to know the details of complex data the following objects are used
Data - a general complex data object
FileOut - can write a Data object to disk
FileIn - can read a Data object from disk

Other objects used are
YUVRange - defines a range of color
VidWindow - defines a rectangular region of the visual field

Group Name
scribbler* functions

Individual functions

```
int     scribblerLeftIR();
int     scribblerRightIR();
int     scribblerLeftLight();
int     scribblerCenterLight();
int     scribblerRightLight();
int     scribblerLineLeft();
int     scribblerLineRight();
int     scribblerStall();
```

Group Description

These functions access the local copies of the Scribblers sensors.
The local copies are updated by the updateScribblerSensors function
and the setScribbler* functions.

Individual description

scribblerLeftIR()	returns 1 if clear to left, otherwise 0
scribblerRightIR()	returns 1 if clear to left, otherwise 0
scribblerLeftLight()	returns light value on left photo cell
scribblerCenterLight()	returns light value on center photo cell
scribblerRightLight()	returns light value on right photo cell
scribblerLineLeft()	returns 1 if Scribbler has detected a line on the left, otherwise 0
scribblerLineRight()	returns 1 if Scribbler has detected a line on the right, otherwise 0
scribblerStall()	returns 1 if Scribbler is stalled, otherwise 0

N.B. stall condition = pushing a heavy object

Example usage

```
for (int i = 0;i < 100;i++)
{
        robot.updateScribblerSensors();
        if (robot.scribblerLeftIR())
                printf("We can go Left ");
        if (robot.scribblerRightIR())
                printf("We can go Right ");
        printf("\n");
}
```

Group Name
readScribbler*

Individual functions

```
int     readScribblerLeftIR(int &);
int     readScribblerRightIR(int &);
int     readScribblerLeftLight(int &);
int     readScribblerCenterLight(int &);
int     readScribblerRightLight(int &);
int     readScribblerLineLeft(int &);
int     readScribblerLineRight(int &);
int     readScribblerStall(int &);
```

Group Description

These functions read current sensor value of the Scribbler Robot.
The value is returned in the parameter and the return code indicates
if the read was successful (1) or not (0);

Individual description

readScribblerLeftIR(parm)	parm is 1 if clear to left, 0 otherwise
readScribblerRightIR(parm)	parm is 1 if clear to right, 0 otherwise
readScribblerLeftLight(parm)	parm is light value from left photo cell
readScribblerCenterLight(parm)	parm is light value from center photo cell
readScribblerRightLight(parm)	parm is light value from right photo cell
readScribblerLineLeft(parm)	parm is 1 if line detected on left, 0 otherwise
readScribblerLineRight(parm)	parm is 1 if line detected on right, 0 otherwise
readScribblerStall(parm)	parm is 1 if robot is stuck, 0 otherwise

Example usage

```
int     speed = 50;
int     stall;

for (int i = 0;i < 100;i++)
{
        robot.drive(speed);
        robot.readScribblerStall(stall);
        if (stall)
                speed = 0 - speed;
}
robot.stop();
```

BUGS

Stall may be a misnomer, Stuck may be a better term.

Group Name
setScribbler*Led

Individual functions

```
int     setScribblerLeftLed(int value);
int     setScribblerCenterLed(int value);
int     setScribblerRightLed(int value);
int     setScribblerAllLeds(int value);
int     setScribblerLeds(int left, int center, int right);
```

Group description

Turn on or off the Light Emitting Diodes on the Scribbler.
return code specifies if the set was successful (1) or not (0).

Individual description

```
setScribblerLeftLed(0)              Turn off Left LED
setScribblerLeftLed(1)              Turn on  Left LED
setScribblerCenterLed(0)            Turn off Center LED
setScribblerCenterLed(1)            Turn on  Center LED
setScribblerRightLed(0)             Turn off Right LED
setScribblerRightLed(1)             Turn on Right LED
setScribblerAllLeds(0)              Turn off all three LEDS
setScribblerAllLeds(1)              Turn on  all three LEDS
setScribblerLeds(int left, int center, int right)
                                    Individually set each LED
                                    1 is on, 0 is off
```

Example usage

```
int     ret;

for (int i = 0;i < 100;i++)
{
        robot.updateScribblerSensors();
        ret = robot.scribblerLeftIR();
        robot.setScribblerLeftLed(ret);
        ret = robot.scribblerRightIR();
        robot.setScribblerRightLed(ret);
}
robot.stop();
```

BUGS

A common source for bugs is using these functions with out a call of updateScribblerSensors() or some other function that re-loads the local copy of the scribble sensors. Values will not change.

Group Name
Motor functions

Individual functions

```
int     drive(int left, int right, int duration = 0);
int     stop();
```

Group description

functions that set the speed of the motors

Individual description

drive(int left, int right)	set the left and right motor speed in range -100 to +100, negative is reverse
stop()	stop both motors

Example usage

```
robot.drive(50, 50);
sleep(1);
robot.drive(50, -50);
robot.stop();               /* could be robot drive(0, 0); */
```

BUGS

The duration parameter issues a stop after duration hundredths of a second.
The duration parameter is not very accurate and not tested much. My addvice
is to not use it.

Group Name
get and set Scribbler data items

Individual functions

```
int     setScribblerData(char data[8]);
int     getScribblerData(char data[8]);
int     setScribblerName(char name[16]);
int     getScribblerName(char name[16]);
int     setScribblerPass(char pass[16]);
int     getScribblerPass(char pass[16]);
```

Group description

get/set the three data strings on the Scribbler, Data, Name or Pass.
return code indicates success (1) or not (0)

Individual description

setScribblerData(data)	write data string
getScribblerData(data)	read data string
setScribblerName(name)	write name
getScribblerName(name)	read name
setScribblerPass(pass)	write pass
getScribblerPass(pass)	read pass

Example usage

```
char    buffer[16];

robot.getScribblerName(buffer);
printf("Robots name is <%s>\n", buffer);
```

BUGS

The returned character string is not null terminated if the data fills the field.

Group Name
sound

Individual function
int beep(int freq, int duration);

Group description
Scribbler will make a sound

Individual description
beep(int freq, int duration) play note of frequency freq
for duration microseconds

Example usage

```
robot.beep(512, 1000);
```

Group Name
photo functions

Group description

The fluke has a camera and can take and return a photo
Fluke photo functions that return a pointer to a Data object

Individual functions

```
Data    *takePhotoJPEG();
Data    *grabGrayArray();
Data    *readJpegHeader();
Data    *readJpegScan();
Data    *readJpegGrayHeader();
Data    *readJpegGrayScan();
Data    *getImage();
Data    *takePhotoGrayJPEG();
```

Individual description

takePhotoJPEG()	take a color JPEG photo
takePhotoGrayJPEG()	take a gray scale JPEG photo
grabGrayArray()	
readJpegHeader()	get JPEG color header
readJpegScan()	get JPEG color scan
readJpegGrayHeader()	get JPEG gray scale header
readJpegGrayScan()	get JPEG gray scale scan
getImage();	

N.B. on failure these functions return null

Example usage

```
Data    *data;
FileOut  *file;

data = robot.takePhotoJPEG();
if (data)
{
        printf("got a jpeg photo - save it\n");
        file = new FileOut("myphoto.jpeg");
        /* just save the data so we can examine the picture    */
        data->write(*file,  0);
        delete data;
        delete file;
}
```

BUGS

about the only useful thing to do with a photo is to save it to disk and
then view it with a program that understands the JPEG format.
Always check for a return of null.

Group Name

Compressed bitmap functions

Individual functions

```
int     setBitmapParams( unsigned char delay, unsigned char thresh,
                YUVRange &range);
Data    *getCompressedBitmap();
```

Group description

functions to get a highly compressed

Individual description

setBitmapParams(unsigned char delay, unsigned char thresh, YUVRange &range)
set the parameters for the bitmap

getCompressedBitmap() read the bitmap

Example usage

```
YUVRange        range(0x00FE, 0x3388, 0xBEFE);  // the color

Data    *data;
FileOut *file;

robot.setBitmapParams( 90, 4, range);

data = robot.getCompressedBitmap();
if (data)
{
        printf("got a compressed bitmap - save it\n");
        file = new FileOut("trybitmap.dat");
        data->write(*file, LITTLENDIAN);
        delete data;
        delete file;
}
```

BUGS

Considerable post-processing is required to turn the camera into a useful sensor, none is provided.

Always check for a return of null

Group name

window functions

Individual functions

```
int     setWindow(int id, VidWindow &win, int xstep, int ystep);
Data    *getWindow(int id);
long    getWindowLight(int win);
int     getBlobWindow(int wind, int &x, int &y);
```

Group description

The Fluke can maintain 4 windows, retangular areas in the visual field and some functions can restrict their attention to one of these windows.

Individual description

```
setWindow(int id, VidWindow &win, int xstep, int ystep);
                                        define a window
getWindow(int id);                      get window image
getWindowLight(int win);return count on on pixels in window
getBlobWindow(int wind, int &x, int &y);
                                        return the "center" of the on pixels
```

Example usage

```
VidWindow       myWindow(0, 10, 0, 10);  // from (0,0) to (10,10)

long    light;

robot.setWindow(1, myWindow, 1, 1);

light = robot.getWindowLight(1);
printf(" light = %ld\n",  light);
```

BUGS

It is impossible to distinguish between a failed getWindowLight and a successful one that has zero on pixels.

Group name

Dongle IR functions

Individual functions

```
int     getDongleIR(int);
int     setDonglePowerIR(int);
```

Group description

the Fluke dongle has a IR capability, with these functions we can access that capability.

Individual description

setDonglePowerIR(int)	set the power level
getDongleIR(int)	get the IR value the parameter indicates which IR leftIR, centerIR, rightIR The higher the value the closer the obstacle.

N.B. left and right is from the dongles point of view,
the reverse of the Scribblers point of view

Example usage

```
int     left, center, right;

robot.setDonglePowerIR(137);

for (int i = 0;i < 100;i++)
{
        center = robot.getDongleIR(centerIR);
        left = robot.getDongleIR(leftIR);
        right = robot.getDongleIR(rightIR);

        printf("%4d - %4d - %4d\n", left, center, right);
}
```

BUGS

It is necessary to tweak the power for each individual Fluke. What works for one Fluke may not work for another.

Group Name

get information

Individual functions

```
int     getInfo(char *);
int     getVersion(char *);
```

Group description

gets a string describing the robot

Individual description

getInfo(char *)	gets a string describing the Scribbler/Fluke
getVersion(char *)	gets a string describing the Scribbler

Example usage

```
char    buf[128]

robot.getInfo(buf);
printf("our robot is %s\n", buf);
```

BUGS

no check for overflow so make sure your buffer is big enough

Group Name

set Fluke Led

Individual function

int setDimmerLED(int);

Group description

The Fluke has a LED whose brightness can be set in range 0 to 255

Individual description

setDimmerLED(0)	off
setDimmerLED(255)	annoyingly bright

Example usage

```
for (int i = 0;i < 256;i++)
    robot.setDimmerLED(i);
```

BUGS

When the battery is low the Fluke flashes this LED.

I have not checked how this interacts with setDimmerLED

Group Name

Fluke miscellaneous

Group description

The Fluke manages the dialog with the Scribbler and can change so of the messages that get received by the Scribbler.
The Fluke is back-to-back with the scribbler these determine which is forward.

Individual

setForwardness(0)	Scribbler is pointing forward, the default
setForwardness(1)	Fluke is pointing forward
getCamParam(int id, int &val)	read a byte of camera parameters
setCamParam(int id, int val)	write a byte of camera parameters
setWhiteBalance(int)	adjust the light balance
setColorId(int Id, YUVRange &range);	set range of “interesting” color

Example usage

```
robot.setForwardness(0);
```

BUGS

The get and setCamParam work from a software point of view, but what each parameter effects has not been explored.

Group Name

The Bad Group

Individual functions

BlobList	*getBlobList(int id);
int	isObstacleAhead();
int	setEmittersIR(int);
int	getMessageIR(char *buf);
int	sendMessageIR(char const buf[]);

Group description

Functions that have not been implemented or are to support obsolete capabilities.

Example usage

Do not use!

BUGS

Yes

The Simulated Scribbler/Fluke

We have provided a Simulated robot for the students to do initial testing of their programs in their home environment.
N.B. No simulation is perfect and this one is no exception. Students should expect to do additional work, in some cases considerable work, to convert a program that works in the simulation to one that works with the real robot.

The simulation is called visual and versions exist that run on Linux, Max OS X and Windows XP. To start execute video in a separate shell (DOS Box in Windows).
A visual picture of the robot in its world should appear.

Visual listens at the named socket "localsocket" on port 101. When a client is accepted by visual it will then interpret any communication from the client as instructions to a Scribbler/Fluke and respond as the robot would and visually show an approximation of the effect on the screen.

The robot world has obstacles (green blocks), open space (white blocks), light sources (yellow blocks) and black lines.
The world is defined in the file "world" with the lines defined in the first section as a sequence of points before the WORLD statement. The obstacles and light sources are defined after the WORLD statement. X indicates an obstacle and L a light source and space indicates the block is clear.

The simulation uses sample files to simulate the response of the Scribbler/Fluke. These files are the actual response of a real robot captured by the program "try". Try saves a number of files starting with the characters "try", video uses the files with the same name but with "sample" instead of "try".