
Myro

Class Scribbler

java.lang.Object

Myro.Scribbler

```
public class Scribbler extends java.lang.Object
```

Class Scribbler defines methods to control and query a Scribbler robot. The methods defined follow the IPRE Myro specification as described in "Learning Computing with Robots" by Deepak Kumar. (see www.roboteducation.org)

Version:

1.1.0

Author:

Douglas Harms

Field Summary

static int	FORWARD FLUKE Constant passed to setForwardness to set the Fluke board facing forward
static int	FORWARD SCRIBBLER Constant passed to setForwardness to set the Scribbler sensors facing forward
static int	IMAGE BLOB Constant passed to takePicture to select a blob image
static int	IMAGE COLOR Constant passed to takePicture to select a color image
static int	IMAGE GRAY Constant passed to takePicture to select a gray scale image
static int	LED ALL Constant passed to setLED to select all Scribbler LEDs
static int	LED CENTER Constant passed to setLED to select the center Scribble LED
static int	LED LEFT Constant passed to setLED to select the left Scribbler LED
static int	LED OFF Constant passed to setLED to turn selected LED Off
static int	LED ON Constant passed to setLED to turn selected LED On
static int	LED RIGHT Constant passed to setLED to select the right Scribbler LED
static int	SENSOR IR CENTER Constant passed to getObstacle to select the center IR sensor.
static int	SENSOR IR LEFT Constant passed to getIR or getObstacle to select the left IR sensor.

static int	<u>SENSOR_IR_RIGHT</u> Constant passed to <u>getIR</u> or <u>getObstacle</u> to select the right IR sensor.
static int	<u>SENSOR_LIGHT_CENTER</u> Constant passed to <u>getLight</u> or <u>getBright</u> to select the center sensor.
static int	<u>SENSOR_LIGHT_LEFT</u> Constant passed to <u>getLight</u> or <u>getBright</u> to select the left sensor.
static int	<u>SENSOR_LIGHT_RIGHT</u> Constant passed to <u>getLight</u> or <u>getBright</u> to select the right sensor.
static int	<u>SENSOR_LINE_LEFT</u> Constant passed to <u>getLine</u> to select the left line sensor.
static int	<u>SENSOR_LINE_RIGHT</u> Constant passed to <u>getLine</u> to select the right line sensor.
static int	<u>VOLUME_OFF</u> Constant passed to <u>setVolume</u> to turn the Scribbler's speaker off
static int	<u>VOLUME_ON</u> Constant passed to <u>setVolume</u> to turn the Scribbler's speaker on

Constructor Summary

[Scribbler](#)()

Construct a Scribbler object that is not connected to any port.

[Scribbler](#)(java.lang.String portName)

Construct a Scribbler object and connect it to port portName.

Method Summary

void	<u>autoCamera</u> () Turn on the Fluke camera's auto-exposure, auto-gain, and auto-color-balance.
void	<u>backward</u> () Starts the Scribbler moving backward at full speed with no rotational movement.
void	<u>backward</u> (double speed) Starts the Scribbler moving backward at a specified speed with no rotational movement.
void	<u>backward</u> (double speed, double numSeconds) Moves the Scribbler in a backward direction at a specified speed with no rotational movement for a specified amount of time.
void	<u>beep</u> (int frequency, double duration) Causes the Scribbler to emit a melodic single frequency tone.
void	<u>beep</u> (int frequency1, int frequency2, double duration) Causes the Scribbler to emit a melodic dual frequency tone.
void	<u>camera</u> () Opens a window that continually displays the Fluke's camera image.
void	<u>close</u> () Close the connection between the computer and the Scribbler.

void	<u>configureBlob</u> (Myro.MyroBlobSpec blob) Defines the blob used by <u>takePicture</u> (Scribbler.IMAGE_BLOB).
boolean	<u>connect</u> (java.lang.String portName) Connect the Scribbler to port portName.
void	<u>darkenCamera</u> (int level) Turn off the Fluke camera's auto-exposure, auto-gain, and lower the gain to the specified value.
boolean	<u>flukeConnected</u> () Indicates whether the robot has a Fluke board attached to it.
void	<u>forward</u> () Starts the Scribbler moving forward at full speed with no rotational movement.
void	<u>forward</u> (double speed) Starts the Scribbler moving forward at a specified speed with no rotational movement.
void	<u>forward</u> (double speed, double numSeconds) Moves the Scribbler in a forward direction at a specified speed with no rotational movement for a specified amount of time.
void	<u>gamepad</u> () Opens a window that permits the user to control the movement of the Scribbler.
double	<u>getBattery</u> () Gets the voltage of the Scribbler's battery.
Myro.MyroBlobImageInfo	<u>getBlob</u> () Get info about the blob image.
double[]	<u>getBright</u> () Returns the values of all three virtual light sensors on the Fluke
double	<u>getBright</u> (int whichSensor) Read one of the Fluke's virtual light sensors.
int	<u>getForwardness</u> () Gets the current forwardness of the robot.
double[]	<u>getFudge</u> () Returns the four "fudge factors" used to tweak the motors.
java.lang.String	<u>getInfo</u> () Returns the info string provided by the Scribbler.
boolean[]	<u>getIR</u> () Returns the state of both of the Scribbler's IR sensors.
boolean	<u>getIR</u> (int whichIR) Returns the state of one of the Scribbler's IR sensors.
double[]	<u>getLight</u> () Returns the state of all three Scribbler light sensors.
double	<u>getLight</u> (int whichLight) Returns the state of one of the Scribbler's light sensors.
boolean[]	<u>getLine</u> () Returns the state of both of the Scribbler's line sensors.
boolean	<u>getLine</u> (int whichSensor)

	Returns the state of one of the Scribbler's line sensors.
java.lang.String	<u>getName()</u> Returns the name of the Scribbler.
double[]	<u>getObstacle()</u> Returns the values of all three obstacles sensors on the Fluke.
double	<u>getObstacle(int whichSensor)</u> Read one of the Fluke's IR obstacle sensors.
boolean	<u>getStall()</u> Returns whether the Scribbler has stalled (i.e., stopped moving).
void	<u>joyStick()</u> Opens a window that permits the user to control the movement of the Scribbler.
void	<u>manualCamera(int gain, int brightness, int exposure)</u> Set the Fluke camera's gain, brightness, and exposure control to specific values.
void	<u>motors(double left, double right)</u> Starts the Scribbler moving by specifying the amount of power going to each wheel.
void	<u>move(double translate, double rotate)</u> Starts the Scribbler moving in the specified direction.
void	<u>playSong(java.lang.String song, double wholeNote)</u> Causes the Scribbler to play a song comprised of a sequence of 0 or more notes.
boolean	<u>portOpened()</u> Indicates whether the port connecting to the robot has been opened.
void	<u>reset()</u> resets the Scribbler.
void	<u>rotate(double speed)</u> Starts the Scribbler rotating at a specified speed without changing the Scribbler's current forward or backward movement.
boolean	<u>scribbler2Connected()</u> Returns whether a scribbler2 is currently connected.
boolean	<u>scribblerConnected()</u> Returns whether a scribbler or Scribbler2 is currently connected.
void	<u>senses()</u> Opens a window that continually displays the sensor values of the Scribbler and/or Fluke.
void	<u>setForwardness(int forwardness)</u> Sets the forwardness of the robot.
void	<u>setFudge(double fastForward, double slowForward, double fastBackward, double slowBackward)</u> Sets the four "fudge factors" for tweaking the motors.
void	<u>setIRPower(int powerLevel)</u> Sets the power level of the Fluke's IR obstacle sensors.
void	<u>setLED(int position, int onOff)</u> Sets one (or all) of the Scribbler's LEDs on or off.

void	<u>setLEDBack</u> (double brightness) Sets the back LED on the Fluke board to a specified brightness.
void	<u>setLEDFront</u> (int onOff) Sets the front LED on the Fluke board on or off.
void	<u>setName</u> (java.lang.String newName) Sets the name of the Scribbler.
void	<u>setVolume</u> (int onOff) Turns the volume of the scribbler on or off.
void	<u>stop</u> () Causes the Scribbler to stop moving.
Myro.MyroImage	<u>takePicture</u> (int imageType) Takes a picture with Fluke's camera.
void	<u>translate</u> (double speed) Starts the Scribbler moving forward or backward at a specified speed without changing the Scribbler's current rotational movement.
void	<u>turnLeft</u> () Moves the Scribbler in a counterclockwise rotation at full speed with no forward or backward movement.
void	<u>turnLeft</u> (double speed) Moves the Scribbler in a counterclockwise rotation at a specified speed with no forward or backward movement.
void	<u>turnLeft</u> (double speed, double numSeconds) Moves the Scribbler in a counterclockwise rotation at a specified speed with no forward or backward movement for a specified amount of time.
void	<u>turnRight</u> () Moves the Scribbler in a clockwise rotation at full speed with no forward or backward movement.
void	<u>turnRight</u> (double speed) Moves the Scribbler in a clockwise rotation at a specified speed with no forward or backward movement.
void	<u>turnRight</u> (double speed, double numSeconds) Moves the Scribbler in a clockwise rotation at a specified speed with no forward or backward movement for a specified amount of time.

Methods inherited from class java.lang.Object

`equals`, `getClass`, `hashCode`, `notify`, `notifyAll`, `toString`, `wait`, `wait`, `wait`

Field Detail

FORWARD_FLUKE

```
public static final int FORWARD_FLUKE
```

Constant passed to [setForwardness](#) to set the Fluke board facing forward

See Also:

[Constant Field Values](#)

FORWARD_SCRIBBLER

public static final int **FORWARD_SCRIBBLER**

Constant passed to [setForwardness](#) to set the Scribbler sensors facing forward

See Also:

[Constant Field Values](#)

IMAGE_BLOB

public static final int **IMAGE_BLOB**

Constant passed to [takePicture](#) to select a blob image

See Also:

[Constant Field Values](#)

IMAGE_COLOR

public static final int **IMAGE_COLOR**

Constant passed to [takePicture](#) to select a color image

See Also:

[Constant Field Values](#)

IMAGE_GRAY

public static final int **IMAGE_GRAY**

Constant passed to [takePicture](#) to select a gray scale image

See Also:

[Constant Field Values](#)

LED_ALL

public static final int **LED_ALL**

Constant passed to [setLED](#) to select all Scribbler LEDs

See Also:

[Constant Field Values](#)

LED_CENTER

public static final int **LED_CENTER**

Constant passed to [setLED](#) to select the center Scribble LED

See Also:

[Constant Field Values](#)

LED_LEFT

public static final int **LED_LEFT**

Constant passed to [setLED](#) to select the left Scribbler LED

See Also:
[Constant Field Values](#)

LED_OFF

public static final int **LED_OFF**
Constant passed to [setLED](#) to turn selected LED Off
See Also:
[Constant Field Values](#)

LED_ON

public static final int **LED_ON**
Constant passed to [setLED](#) to turn selected LED On
See Also:
[Constant Field Values](#)

LED_RIGHT

public static final int **LED_RIGHT**
Constant passed to [setLED](#) to select the right Scribbler LED
See Also:
[Constant Field Values](#)

SENSOR_IR_CENTER

public static final int **SENSOR_IR_CENTER**
Constant passed to [getObstacle](#) to select the center IR sensor.
See Also:
[Constant Field Values](#)

SENSOR_IR_LEFT

public static final int **SENSOR_IR_LEFT**
Constant passed to [getIR](#) or [getObstacle](#) to select the left IR sensor.
See Also:
[Constant Field Values](#)

SENSOR_IR_RIGHT

public static final int **SENSOR_IR_RIGHT**
Constant passed to [getIR](#) or [getObstacle](#) to select the right IR sensor.
See Also:
[Constant Field Values](#)

SENSOR_LIGHT_CENTER

public static final int **SENSOR_LIGHT_CENTER**

Constant passed to [getLight](#) or [getBright](#) to select the center sensor.

See Also:

[Constant Field Values](#)

SENSOR_LIGHT_LEFT

public static final int **SENSOR_LIGHT_LEFT**

Constant passed to [getLight](#) or [getBright](#) to select the left sensor.

See Also:

[Constant Field Values](#)

SENSOR_LIGHT_RIGHT

public static final int **SENSOR_LIGHT_RIGHT**

Constant passed to [getLight](#) or [getBright](#) to select the right sensor.

See Also:

[Constant Field Values](#)

SENSOR_LINE_LEFT

public static final int **SENSOR_LINE_LEFT**

Constant passed to [getLine](#) to select the left line sensor.

See Also:

[Constant Field Values](#)

SENSOR_LINE_RIGHT

public static final int **SENSOR_LINE_RIGHT**

Constant passed to [getLine](#) to select the right line sensor.

See Also:

[Constant Field Values](#)

VOLUME_OFF

public static final int **VOLUME_OFF**

Constant passed to [setVolume](#) to turn the Scribbler's speaker off

See Also:

[Constant Field Values](#)

VOLUME_ON

public static final int **VOLUME_ON**

Constant passed to [setVolume](#) to turn the Scribbler's speaker on

See Also:
[Constant Field Values](#)

Constructor Detail

Scribbler

```
public Scribbler()
```

Construct a Scribbler object that is not connected to any port. Method [connect](#) must be called to connect this Scribbler to a port.

Scribbler

```
public Scribbler(java.lang.String portName)
```

Construct a Scribbler object and connect it to port portName. If the connection was successfully made then it is legal to invoke methods that require the scribbler be connected; if the connection was not successful then it is not legal to invoke methods that require the scribbler to be connected. Method [scribblerConnected](#) can be used to determine if the connection was successfully made.

Parameters:

portName - the name of the port the Scribbler is attached to (e.g., "COM1", "/dev/ttyS0")

Method Detail

autoCamera

```
public void autoCamera()
```

Turn on the Fluke camera's auto-exposure, auto-gain, and auto-color-balance.

Precondition:

flukeConnected

backward

```
public void backward()
```

Starts the Scribbler moving backward at full speed with no rotational movement. The Scribbler will continue to move until another movement method is invoked (e.g., [stop](#), [move](#), [forward](#), [backward](#), [turnLeft](#), [turnRight](#), [motors](#), [translate](#), [rotate](#)).

Precondition:

scribblerConnected

backward

```
public void backward(double speed)
```

Starts the Scribbler moving backward at a specified speed with no rotational movement. The Scribbler will continue to move until another movement method is invoked (e.g., [stop](#), [move](#), [forward](#), [backward](#), [turnLeft](#), [turnRight](#), [motors](#), [translate](#), [rotate](#)).

Parameters:

speed - Specifies the speed. Positive values specify backward movement (1.0 is full backward speed), negative values specify forward movement (-1.0 is full forward speed).

Precondition:

scribblerConnected and speed between -1.0 (inclusive) and 1.0 (inclusive)

backward

```
public void backward(double speed,  
                    double numSeconds)
```

Moves the Scribbler in a backward direction at a specified speed with no rotational movement for a specified amount of time. The Scribbler will stop moving at the end of the specified time period. This method will not return until the specified time period has occurred.

Parameters:

speed - Specifies the backward speed. Positive values specify backward movement (1.0 is full backward speed), negative values specify forward movement (-1.0 is full forward speed).

numSeconds - Specifies the length of time to move, in seconds.

Precondition:

scribblerConnected, speed between -1.0 (inclusive) and 1.0 (inclusive), numSeconds \geq 0.0

beep

```
public void beep(int frequency,  
                double duration)
```

Causes the Scribbler to emit a melodic single frequency tone.

Parameters:

frequency - The frequency of the tone to emit.

duration - The length of the tone to be emitted, in seconds.

Precondition:

scribblerConnected

beep

```
public void beep(int frequency1,  
                int frequency2,  
                double duration)
```

Causes the Scribbler to emit a melodic dual frequency tone.

Parameters:

frequency1 - The frequency of one of the tones to emit.

frequency2 - The frequency of the other tone to emit.

duration - The length of the tone to be emitted, in seconds.

Precondition:

scribblerConnected

camera

```
public void camera()
```

Opens a window that continually displays the Fluke's camera image. The image is updated every second. Only one camera window is permitted to be opened for a particular Scribbler/Fluke; no action occurs if this method is invoked when a camera window is already opened. The window will stay opened until the user closes it (by clicking the window's close icon) or the [close](#) method is invoked.

Precondition:

scribblerConnected or flukeConnected

close

```
public void close()
```

Close the connection between the computer and the Scribbler. Any threads associated with this robot (e.g., senses, joystick) will be killed. After calling close the Scribbler cannot be accessed again unless [connect](#) is called to reestablish the connection. It is important to invoke this method at the end of the program. Failure to do so may cause problems when connecting to the Scribbler in the future.

configureBlob

```
public void configureBlob(Myro.MyroBlobSpec blob)
```

Defines the blob used by [takePicture](#)(Scribbler.IMAGE_BLOB). A blob specifies a range of colors and is usually defined by calling [defineBlob](#) or [getUserDefinedBlob](#) in a MyroImage.

Parameters:

blob - Specifies the color range of the blob

Precondition:

flukeConnected, blob not null

connect

```
public boolean connect(java.lang.String portName)
```

Connect the Scribbler to port portName. If the Scribbler is already connected to a port it is first closed. If the connection was successfully made then it is legal to invoke methods that require the scribbler be connected; if the connection was not successful then it is not legal to invoke methods that require the scribbler to be connected. Methods [scribblerConnected](#) and [flukeConnected](#) can be used to determine if the connection was successfully made.

Parameters:

portName - The name of the port the Scribbler is connected to (e.g., "COM1", "/dev/ttyS0")

Returns:

true returned iff the connection to the Scribbler was successful

darkenCamera

```
public void darkenCamera(int level)
```

Turn off the Fluke camera's auto-exposure, auto-gain, and lower the gain to the specified value. This is useful when using [getBright](#) virtual sensors.

Parameters:

level - The camera's gain level, between 0 and 255.

Precondition:

flukeConnected, level between 0 (inclusive) and 255 (inclusive)

flukeConnected

```
public boolean flukeConnected()
```

Indicates whether the robot has a Fluke board attached to it.

Returns:

true iff a Fluke board is present on the robot

forward

```
public void forward()
```

Starts the Scribbler moving forward at full speed with no rotational movement. The Scribbler will continue to move until another movement method is invoked (e.g., [stop](#), [move](#), [forward](#), [backward](#), [turnLeft](#), [turnRight](#), [motors](#), [translate](#), [rotate](#)).

Precondition:

scribblerConnected

forward

```
public void forward(double speed)
```

Starts the Scribbler moving forward at a specified speed with no rotational movement. The Scribbler will continue to move until another movement method is invoked (e.g., [stop](#), [move](#), [forward](#), [backward](#), [turnLeft](#), [turnRight](#), [motors](#), [translate](#), [rotate](#)).

Parameters:

speed - Specifies the speed. Positive values specify forward movement (1.0 is full forward speed), negative values specify backward movement (-1.0 is full backward speed).

Precondition:

scribblerConnected and speed between -1.0 (inclusive) and 1.0 (inclusive)

forward

```
public void forward(double speed,  
                   double numSeconds)
```

Moves the Scribbler in a forward direction at a specified speed with no rotational movement for a specified amount of time. The Scribbler will stop moving at the end of the specified time period. This method will not return until the specified time period has occurred.

Parameters:

speed - Specifies the forward speed. Positive values specify forward movement (1.0 is full forward speed), negative values specify backward movement (-1.0 is full backward speed).

numSeconds - Specifies the length of time to move, in seconds.

Precondition:

scribblerConnected, speed between -1.0 (inclusive) and 1.0 (inclusive), numSeconds >= 0.0

gamepad

```
public void gamepad()
```

Opens a window that permits the user to control the movement of the Scribbler. The window allows the user to control the Scribbler using a joystick-like interface, permitting forward, backward, right, and left movement. Only one joystick window is permitted to be opened for a particular Scribbler; no action occurs if this method is invoked when a joystick window is already opened. The window will stay opened until the user closes it (by clicking the window's close icon) or the [close](#) method is invoked.

Precondition:

getBattery

```
public double getBattery()
```

Gets the voltage of the Scribbler's battery. If the battery voltage drops below ~6.1V the Fluke's back LED will flash to alert you to change (or preferably recharge) the batteries.

Returns:

The voltage of the Scribbler's battery

Precondition:

flukeConnected

getBlob

```
public Myro.MyroBlobImageInfo getBlob()
```

Get info about the blob image. The returned MyroBlobImageInfo contains the number of pixels in the blob, the average X coordinate and the average Y coordinate.

Returns:

A MyroBlobImageInfo instance that has information about the blob image (number of pixels in the blob, and average location of the blob)

Precondition:

flukeConnected

getBright

```
public double[] getBright()
```

Returns the values of all three virtual light sensors on the Fluke

Returns:

A 3-element array containing the values of the left (in element 0), center (in element 1), and right (in element 2) virtual light sensors.

Precondition:

flukeConnected

getBright

```
public double getBright(int whichSensor)
```

Read one of the Fluke's virtual light sensors. The Fluke's virtual light sensors report the total intensity on the left, center, and right sides of the Fluke's camera.

Parameters:

whichSensor - Specifies the sensor to use. Must be Scribbler.SENSOR_LIGHT_LEFT (or 0), Scribbler.SENSOR_LIGHT_CENTER (or 1), or Scribbler.SENSOR_LIGHT_RIGHT (or 2).

Returns:

The intensity of the light in the selected sensor area. The value is between 0.0 and 1.0.

Precondition:

flukeConnected, and whichSensor is Scribbler.SENSOR_LIGHT_LEFT (or 0), Scribbler.SENSOR_LIGHT_CENTER (or 1), or Scribbler.SENSOR_LIGHT_RIGHT (or 2).

getForwardness

```
public int getForwardness()
```

Gets the current forwardness of the robot. This is an indication of which end of the robot is considered to be "forward", which is either the Fluke board or the Scribbler light sensors.

Returns:

The forwardness of the robot. This will be either Scribbler.FORWARD_SCRIBBLER or Scribbler.FORWARD_FLUKE

Precondition:

flukeConnected

getFudge

```
public double[] getFudge()
```

Returns the four "fudge factors" used to tweak the motors. Each value is between 0.0 (inclusive) and 2.0 (inclusive). A value of 1.0 indicates no tweaking, values between 0.0 and 1.0 indicate a leftward adjustment, and values between 1.0 and 2.0 indicate a rightward adjustment. The further a value is away from 1.0, the larger the adjustment.

Returns:

A four element array. Element 0 is the adjustment for high forward speeds (i.e., > 0.5), element 1 is the adjustment for slow forward speeds (i.e., <= 0.5), element 2 is the adjustment for high backward speeds, element 3 is the adjustment for slow backward speeds.

Precondition:

scribblerConnected()

getInfo

```
public java.lang.String getInfo()
```

Returns the info string provided by the Scribbler. The specific information contains such things as the firmware version, the type of robot (i.e., Scribbler) and the communication mode (e.g., Serial).

Returns:

A String containing information about the connected robot, such as robot type (e.g., Scribbler), firmware version number, and communication mode (e.g., Serial).

Precondition:

portOpened

getIR

```
public boolean[] getIR()
```

Returns the state of both of the Scribbler's IR sensors.

Returns:

A two element boolean array containing the values of the IR sensor. True means that an obstacle is NOT detected by the selected IR sensor, and false means that an obstacle IS detected by the sensor.

Precondition:

scribblerConnected()

getIR

```
public boolean getIR(int whichIR)
```

Returns the state of one of the Scribbler's IR sensors. whichIR specifies the IR sensor to query.

Parameters:

whichIR - Specifies the IR sensor to query. Should be [Scribbler.SENSOR_IR_LEFT](#) (or 0) or [Scribbler.SENSOR_IR_RIGHT](#) (or 1).

Returns:

The value of the selected IR sensor. True means that an obstacle is NOT detected by the selected IR sensor, and false means that an obstacle IS detected by the sensor.

Precondition:

scribblerConnected() and whichIR is [Scribbler.SENSOR_IR_LEFT](#) (or 0) or [Scribbler.SENSOR_IR_RIGHT](#) (or 1).

getLight

```
public double[] getLight()
```

Returns the state of all three Scribbler light sensors.

Returns:

A three element array. element 0 contains the value of the left sensor, element 1 contains the value of the center sensor, and element 2 contains the value of the right sensor. All values are between 0 and 1000, and low values indicate low light, high values indicate high light.

Precondition:

scribblerConnected()

getLight

```
public double getLight(int whichLight)
```

Returns the state of one of the Scribbler's light sensors. whichLight specifies the light sensor to query.

Parameters:

whichLight - Specifies the light sensor to query. Must be [Scribbler.SENSOR_LIGHT_LEFT](#) (or 0), [Scribbler.SENSOR_LIGHT_CENTER](#) (or 1), or [Scribbler.SENSOR_LIGHT_RIGHT](#) (or 2).

Returns:

The value of the selected light sensor. The value will be between 0.0 and 1.0, and a low value indicates low light, a high value indicates bright light.

Precondition:

scribblerConnected() and whichLight is [Scribbler.SENSOR_LIGHT_LEFT](#) (or 0), [Scribbler.SENSOR_LIGHT_CENTER](#) (or 1), or [Scribbler.SENSOR_LIGHT_RIGHT](#) (or 2).

getLine

```
public boolean[] getLine()
```

Returns the state of both of the Scribbler's line sensors.

Returns:

A two element boolean array containing the values of the line sensor. True means that a (dark) line is detected by the selected line sensor, and false means that a (dark) line is not detected by the sensor.

Precondition:

scribblerConnected()

getLine

```
public boolean getLine(int whichSensor)
```

Returns the state of one of the Scribbler's line sensors. whichSensor specifies the line sensor to query.

Parameters:

whichSensor - Specifies the line sensor to query. Should be [Scribbler.SENSOR_LINE_LEFT](#) (or 0) or [Scribbler.SENSOR_LINE_RIGHT](#) (or 1).

Returns:

The value of the selected line sensor. True means that a (dark) line is detected by the selected line sensor, and false means that a (dark) line is not detected by the sensor.

Precondition:

scribblerConnected() and whichSensor is [Scribbler.SENSOR_LINE_LEFT](#) (or 0) or [Scribbler.SENSOR_LINE_RIGHT](#) (or 1).

getName

```
public java.lang.String getName()
```

Returns the name of the Scribbler. The name is set with [setName](#).

Returns:

The name of the Scribbler.

Precondition:

scribblerConnected

getObstacle

```
public double[] getObstacle()
```

Returns the values of all three obstacles sensors on the Fluke.

Returns:

a 3-element array containing the values of the left (in element 0), center (in element 1), and right (in element 2) obstacle sensors.

Precondition:

flukeConnected

getObstacle

```
public double getObstacle(int whichSensor)
```

Read one of the Fluke's IR obstacle sensors.

Parameters:

whichSensor - Selects the Fluke IR sensor. Should be [Scribbler.SENSOR_IR_LEFT](#) (or 0), [Scribbler.SENSOR_IR_CENTER](#) (or 2), or [Scribbler.SENSOR_IR_RIGHT](#) (or 1)

Returns:

The value of the selected sensor. A low value means there are no obstacles detected, a high value means there is an obstacle detected. The return value is in the range 0.0 to 1.0.

Precondition:

flukeConnected, whichSensor is [Scribbler.SENSOR_IR_LEFT](#) (or 0), [Scribbler.SENSOR_IR_CENTER](#) (or 2), or [Scribbler.SENSOR_IR_RIGHT](#) (or 1)

getStall

```
public boolean getStall()
```

Returns whether the Scribbler has stalled (i.e., stopped moving). Returns true iff the Scribbler has stalled.

Precondition:
scribblerConnected()

joyStick

```
public void joyStick()
```

Opens a window that permits the user to control the movement of the Scribbler. The window allows the user to control the Scribbler using a joystick-like interface, permitting forward, backward, right, and left movement. Only one joystick window is permitted to be opened for a particular Scribbler; no action occurs if this method is invoked when a joystick window is already opened. The window will stay opened until the user closes it (by clicking the window's close icon) or the [close](#) method is invoked.

Precondition:
scribblerConnected

manualCamera

```
public void manualCamera(int gain,  
                        int brightness,  
                        int exposure)
```

Set the Fluke camera's gain, brightness, and exposure control to specific values. The default values are: gain:0, brightness: 120, exposure:65.

Precondition:
flukeConnected, all parameters between 0 (inclusive) and 255 (inclusive)

motors

```
public void motors(double left,  
                 double right)
```

Starts the Scribbler moving by specifying the amount of power going to each wheel. The Scribbler will continue to move until another movement method is invoked (e.g., [stop](#), [move](#), [forward](#), [backward](#), [turnLeft](#), [turnRight](#), [motors](#), [translate](#), [rotate](#)).

Parameters:

`left` - Specifies the speed of the left wheel. Values > 0 specify forward speed (with 1.0 specifying full forward speed), values < 0 specify backward speed (with -1.0 specifying full backward speed). 0 specifies no forward or backward speed.

`right` - Specifies the speed of the right wheel. Values > 0 specify forward speed (with 1.0 specifying full forward speed), values < 0 specify backward speed (with -1.0 specifying full backward speed). 0 specifies no forward or backward speed.

Precondition:
scribblerConnected, and left and right are both between -1.0 (inclusive) and 1.0 (inclusive)

move

```
public void move(double translate,  
               double rotate)
```

Starts the Scribbler moving in the specified direction. The Scribbler will continue to move until another movement method is invoked (e.g., [stop](#), [move](#), [forward](#), [backward](#), [turnLeft](#), [turnRight](#), [motors](#), [translate](#), [rotate](#)).

Parameters:

`translate` - Specifies the forward movement speed. Values > 0 specify forward speed (with 1.0 specifying full forward speed), values < 0 specify backward speed (with -1.0 specifying full backward speed). 0 specifies no forward or backward speed.

`rotate` - Specifies rotational speed. Values > 0 specify counterclockwise rotation (with 1.0 specifying full counterclockwise rotation), values < 0 specify clockwise rotation (with -1.0 specifying full clockwise rotation). 0 specifies no rotation at all.

Precondition:

`scribblerConnected`, and `translate` and `rotate` are both between -1.0 (inclusive) and 1.0 (inclusive)

playSong

```
public void playSong(java.lang.String song,  
                     double wholeNote)
```

Causes the Scribbler to play a song comprised of a sequence of 0 or more notes. The song is passed as a String comprised of note specifications separated by semicolons. A note specification is either a single note followed by a duration or a chord followed by a duration. A single note is either a note name (e.g., A A# Bb B) or a frequency (e.g., 440); a chord consists of two single notes. Duration is expressed in terms of fraction of a wholenote.

Parameters:

`song` - A sequence of 0 or more notes

`wholeNote` - The duration of a whole note, in seconds

Precondition:

`scribblerConnected`, `wholeNote` > 0

portOpened

```
public boolean portOpened()
```

Indicates whether the port connecting to the robot has been opened. Note that if true is returned it isn't necessarily know what kind of robot is connected (e.g., `scribbler` or `fluke`); [flukeConnected](#) or [scribblerConnected](#) should be used to determine this.

Returns:

true iff the connection to the robot has been established.

reset

```
public void reset()  
  resets the Scribbler.
```

Precondition:

`scribblerConnected`()

rotate

```
public void rotate(double speed)
```

Starts the Scribbler rotating at a specified speed without changing the Scribbler's current forward or backward movement. The Scribbler will continue to move until another movement method is invoked (e.g., [stop](#), [move](#), [forward](#), [backward](#), [turnLeft](#), [turnRight](#), [motors](#), [translate](#), [rotate](#)).

Parameters:

speed - Specifies the rotational speed. Positive values specify counterclockwise rotation (1.0 is full counterclockwise speed), negative values specify clockwise rotation (-1.0 is full clockwise speed).

Precondition:

scribblerConnected and speed between -1.0 (inclusive) and 1.0 (inclusive)

scribbler2Connected

```
public boolean scribbler2Connected()
```

Returns whether a scribbler2 is currently connected.

Returns:

true iff a Scribbler2 is currently connected.

scribblerConnected

```
public boolean scribblerConnected()
```

Returns whether a scribbler or Scribbler2 is currently connected.

Returns:

true iff a Scribbler or Scribbler2 is currently connected

senses

```
public void senses()
```

Opens a window that continually displays the sensor values of the Scribbler and/or Fluke. The values are updated every .5 seconds. Only one senses window is permitted to be opened for a particular Scribbler/Fluke; no action occurs if this method is invoked when a senses window is already opened.

The window will stay opened until the user closes it (by clicking the window's close icon) or the [close](#) method is invoked.

Precondition:

scribblerConnected or flukeConnected

setForwardness

```
public void setForwardness(int forwardness)
```

Sets the forwardness of the robot. The forwardness specifies which end of the robot is considered to be "forward" for the purposes of movement methods. The two possible directions are either the Fluke board is "forward" or the Scribbler light sensors are "forward". The default is FORWARD_FLUKE.

Parameters:

forwardness - Indicates which robot end is "forward". Must be Scribbler.FORWARD_FLUKE or Scribbler.FORWARD_SCRIBBLER.

Precondition:

flukeConnected and forwardness is Scribbler.FORWARD_SCRIBBLER (or 0) or Scribbler.FORWARD_FLUKE (or 1)

setFudge

```
public void setFudge(double fastForward,  
                    double slowForward,  
                    double fastBackward,  
                    double slowBackward)
```

Sets the four "fudge factors" for tweaking the motors. Each value is between 0.0 (inclusive) and 2.0 (inclusive). A value of 1.0 indicates no tweaking, values between 0.0 and 1.0 indicate a leftward adjustment, and values between 1.0 and 2.0 indicate a rightward adjustment. The further a value is away from 1.0, the larger the adjustment.

Parameters:

`fastForward` - Tweak value for fast forward speeds (i.e., speed > 0.5)

`slowForward` - Tweak value for slow forward speeds (i.e., speed <= 0.5)

`fastBackward` - Tweak value for fast backward speeds (i.e., speed > 0.5)

`slowBackward` - Tweak value for slow backward speeds (i.e., speed <= 0.5)

Precondition:

`scribblerConnected` and all four parameters between 0.0 (inclusive) and 2.0 (inclusive)

setIRPower

```
public void setIRPower(int powerLevel)
```

Sets the power level of the Fluke's IR obstacle sensors. The default value is 135. If [getObstacle](#) always reports high values, try lowering the power level; if it always reports very low values, try increasing the power level.

Parameters:

`powerLevel` - Specifies the power level

Precondition:

`flukeConnected`, `powerLevel` between 0 (inclusive) and 255 (inclusive)

setLED

```
public void setLED(int position,  
                  int onOff)
```

Sets one (or all) of the Scribbler's LEDs on or off.

Precondition:

`scribblerConnected`; `position` is `Scribbler.LED_LEFT` (or 0), `Scribbler.LED_CENTER` (or 1), `Scribbler.LED_RIGHT` (or 2), or `Scribbler.LED_ALL` (or 3); `onOff` is `Scribbler.LED_OFF` (or 0) or `Scribbler.LED_ON` (or 1)

setLEDBack

```
public void setLEDBack(double brightness)
```

Sets the back LED on the Fluke board to a specified brightness.

Parameters:

`brightness` - A value between 0.0 and 1.0 that specifies the brightness of the LED

Precondition:

`flukeConnected`, and `brightness` between 0.0 (inclusive) and 1.0 (inclusive)

setLEDFront

```
public void setLEDFront(int onOff)
```

Sets the front LED on the Fluke board on or off.

Parameters:

onOff - Specifies whether to turn on the LED (Scribbler.LED_ON) or turn it off (Scribbler.LED_OFF)

Precondition:

flukeConnected, onOff either Scribbler.LED_OFF (or 0) or Scribbler.LED_ON (or 1)

setName

```
public void setName(java.lang.String newName)
```

Sets the name of the Scribbler.

Parameters:

newName - String containing the new name of the Scribbler. Only the first 16 characters of newName are used.

Precondition:

scribblerConnected

setVolume

```
public void setVolume(int onOff)
```

Turns the volume of the scribbler on or off.

Parameters:

onOff - Value indicating whether to turn the volume on (Scribbler.VOLUME_ON or 1) or off (Scribbler.VOLUME_OFF or 0)

Precondition:

scribblerConnected, onOff is Scribbler.VOLUME_OFF (or 0) or Scribbler.VOLUME_ON (or 1)

stop

```
public void stop()
```

Causes the Scribbler to stop moving.

Precondition:

scribblerConnected

takePicture

```
public Myro.MyroImage takePicture(int imageType)
```

Takes a picture with Fluke's camera. The imageType parameter determines what kind of picture is taken.

Parameters:

imageType - Specifies the type of picture to take. Scribbler.IMAGE_COLOR, Scribbler.IMAGE_GRAY, or Scribbler.IMAGE_BLOB

Precondition:

flukeConnected, imageType is Scribbler.IMAGE_COLOR (or 0), Scribbler.IMAGE_GRAY (or 1), or Scribbler.IMAGE_BLOB (or 2)

translate

```
public void translate(double speed)
```

Starts the Scribbler moving forward or backward at a specified speed without changing the Scribbler's current rotational movement. The Scribbler will continue to move until another movement method is invoked (e.g., [stop](#), [move](#), [forward](#), [backward](#), [turnLeft](#), [turnRight](#), [motors](#), [translate](#), [rotate](#)).

Parameters:

speed - Specifies the speed. Positive values specify forward movement (1.0 is full forward speed), negative values specify backward movement (-1.0 is full backward speed).

Precondition:

scribblerConnected and speed between -1.0 (inclusive) and 1.0 (inclusive)

turnLeft

```
public void turnLeft()
```

Moves the Scribbler in a counterclockwise rotation at full speed with no forward or backward movement. The Scribbler will continue to move until another movement method is invoked (e.g., [stop](#), [move](#), [forward](#), [backward](#), [turnLeft](#), [turnRight](#), [motors](#), [translate](#), [rotate](#)).

Precondition:

scribblerConnected

turnLeft

```
public void turnLeft(double speed)
```

Moves the Scribbler in a counterclockwise rotation at a specified speed with no forward or backward movement. The Scribbler will continue to move until another movement method is invoked (e.g., [stop](#), [move](#), [forward](#), [backward](#), [turnLeft](#), [turnRight](#), [motors](#), [translate](#), [rotate](#)).

Parameters:

speed - Specifies the rotational speed. Positive values specify counterclockwise rotation (1.0 is full counterclockwise speed), negative values specify clockwise rotation (-1.0 is full clockwise speed).

Precondition:

scribblerConnected, speed between -1.0 (inclusive) and 1.0 (inclusive)

turnLeft

```
public void turnLeft(double speed,  
                     double numSeconds)
```

Moves the Scribbler in a counterclockwise rotation at a specified speed with no forward or backward movement for a specified amount of time. The Scribbler will stop moving at the end of the specified time period. This method will not return until the specified time period has occurred.

Parameters:

speed - Specifies the rotational speed. Positive values specify counterclockwise rotation (1.0 is full counterclockwise speed), negative values specify clockwise rotation (-1.0 is full clockwise speed).

numSeconds - Specifies the length of time to move, in seconds.

Precondition:

scribblerConnected, speed between -1.0 (inclusive) and 1.0 (inclusive), numSeconds >= 0.0

turnRight

```
public void turnRight()
```

Moves the Scribbler in a clockwise rotation at full speed with no forward or backward movement. The Scribbler will continue to move until another movement method is invoked (e.g., [stop](#), [move](#), [forward](#), [backward](#), [turnLeft](#), [turnRight](#), [motors](#), [translate](#), [rotate](#)).

Precondition:

scribblerConnected

turnRight

```
public void turnRight(double speed)
```

Moves the Scribbler in a clockwise rotation at a specified speed with no forward or backward movement. The Scribbler will continue to move until another movement method is invoked (e.g., [stop](#), [move](#), [forward](#), [backward](#), [turnLeft](#), [turnRight](#), [motors](#), [translate](#), [rotate](#)).

Parameters:

speed - Specifies the rotational speed. Positive values specify clockwise rotation (1.0 is full clockwise speed), negative values specify counterclockwise rotation (-1.0 is full counterclockwise speed).

Precondition:

scribblerConnected, speed between -1.0 (inclusive) and 1.0 (inclusive)

turnRight

```
public void turnRight(double speed,  
                      double numSeconds)
```

Moves the Scribbler in a clockwise rotation at a specified speed with no forward or backward movement for a specified amount of time. The Scribbler will stop moving at the end of the specified time period. This method will not return until the specified time period has occurred.

Parameters:

speed - Specifies the rotational speed. Positive values specify clockwise rotation (1.0 is full clockwise speed), negative values specify counterclockwise rotation (-1.0 is full counterclockwise speed).

numSeconds - Specifies the length of time to move, in seconds.

Precondition:

scribblerConnected, speed between -1.0 (inclusive) and 1.0 (inclusive), numSeconds >= 0.0
