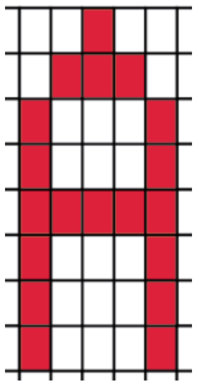
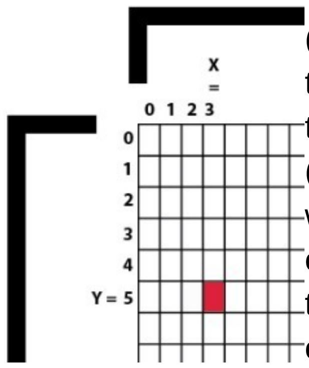
Sparki With Arduino, Guide #3 – LCD Screen

Sparki features a small LCD display that will let us make drawings, write text, and show the status of other components of the robot, such as the readings from its sensors.



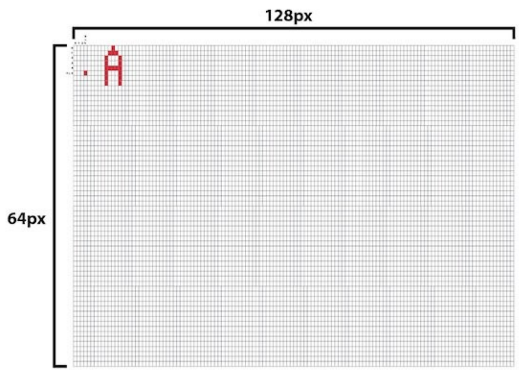
Every display out there is composed by pixels. Pixels are the smallest points that we can control in a display, be it a cellphone screen, a computer monitor, a TV, or Sparki's LCD. In our tiny single color LCD, pixels can be in one of two states: **ON** or **OFF**. By turning different pixels ON and OFF, we can draw something, or write text into the display. For example, the image to the right shows capital "A" letter made of square pixels, like the ones in Sparki's LCD. There are several commands that we can use in our programs to control the pixels, but first we need to understand how each pixel is identified in order to control it.



We can address the pixels using a coordinate system, in this case X (horizontal, or left to right position) and Y coordinates (vertical, or top to bottom position). We type coordinates using the notation (X, Y). In the image to the left, we have a starting (or **zero**) point. This point is (0,0). The colored in pixel has an X value of 3, and a Y value of 5, so we talk about it as the (3,5) pixel. In programing we often start counting at zero, so you'll notice that X=3 is actually the 4th pixel, not the 3rd. This can be a little confusing at first, but you'll get the hang of it in no time.

Here you can see the complete Sparki display. It is 128 pixels wide and 64 pixels tall:

Here are some examples of pixels with their corresponding coordinates:



•The pixel in the upper-left corner is (0, 0).

•The pixel in the first left-right row and in the last top-bottom column is (127, 0).

•The pixel in the last left-right row and in the first top-bottom column is (0, 63).

•The pixel in the last left-right row and in the last top-bottom column is (127, 63).

Use the command drawPixel() to turn on a single pixel. For example, the command drawPixel(3,5) will draw a pixel at the coordinates with X value of 3 and a Y value of 5, or the 4th across and the 6th down.

Pixels are a small part of the screen. If you were to write a letter, or draw a circle, or almost anything else by turning on each pixel it needed, that would be a LOT of work! Imagine the number of lines of code you would need to write in order to create a square that is 20 pixels by 20 pixels? It would take 400 lines of code just to make a simple square! Luckily, Sparki gives us a bunch of commands for doing common things like drawing lines, rectangles, circles, and even text. You can see the full list below.

```
sparki.clearLCD();
sparki.updateLCD();

sparki.drawPixel(xPixel, yPixel);
sparki.readPixel(xPixel, yPixe);
sparki.drawChar(xPixel, yLine, char);
sparki.drawString(xPixel, yLine, *char);
sparki.drawLine(xStart, yStart, xEnd, yEnd);
sparki.drawRect(xCenter, yCenter, width, heigh);
sparki.drawRectFilled(xCenter, yCenter, width, height);
sparki.drawCircle(xCenter, yCenter, radius);
sparki.drawCircleFilled(xCenter, yCenter, radius);
sparki.drawBitmap(xStart, yStart, *bitmap, width, height);
```

Let’s start with a small example: drawing some lines to make a large X on the display, corner to corner:

```
#include <Sparki.h>  // include the sparki library

void setup()
{
}

void loop()
{
  sparki.clearLCD(); // wipe the LCD clear

  sparki.drawLine(0,0, 127,63);
  sparki.drawLine(0,63, 127,0);

  sparki.updateLCD(); // put the drawings on the screen
  delay(1000);
}
```

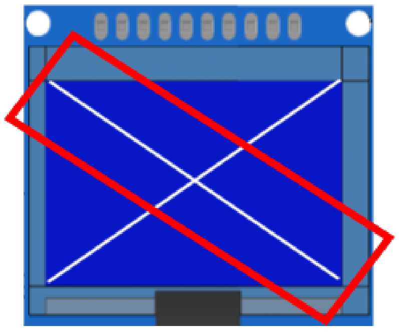
So, how does this work? Let’s take a general look at the code:

- First, everything is written in the loop because the display needs to be updated from time to time. As you can see, there is a **delay(1000);** instruction at the end of the program. The command that updates the display (the **updateLCD()** command) doesn’t need to be executed very often. It just needs to be there when you want the display to be drawn again. Without the **updateLCD()** command the new information would never make its way onto the LCD screen.

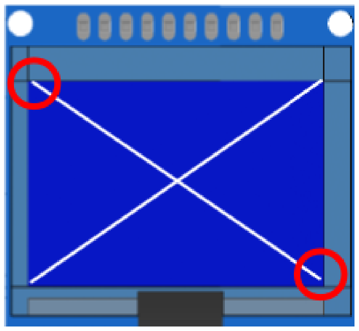
- Second, before we can effectively draw the lines, we have to wipe the LCD clear using the **clearLCD()** command. If you don’t do this then Sparki will leave the previous drawing on the LCD and just draw the new lines over it. That isn’t a big deal for our two lines, but if we wanted to change the position of the lines then without the **clearLCD()**

instruction we could draw a new line but the old line would stay on the screen.

•Finally, in the middle of this program are the line drawing instructions themselves. As you can see, the **drawLine()** command receives 4 numbers as parameters. They are the coordinates belonging to the "important" pixels of our lines. By important pixels we mean the pixels at the beginning and at the end of each line. Let's take a look at the first line **sparki.drawLine(0,0, 127,63)** The first two numbers (0, 0) are located at the upper left corner of the LCD screen and the second set of numbers (127, 63) are located at the lower right corner of the LCD screen. Sparki fills in the pixels that connect these two dots in a straight line.



Line from the first line of code

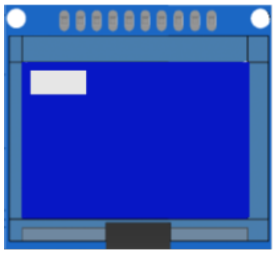


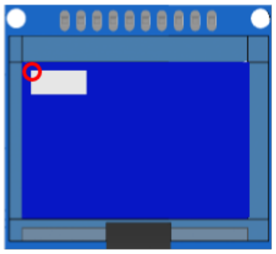
"Important" pixels

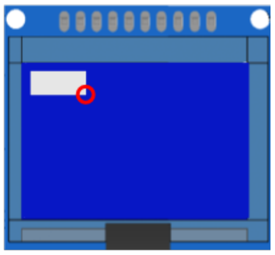
So what about circles, rectangles, or even filled zones? Let's try the following code, and then you can experiment a bit with the coordinates and dimensions of the different elements drawn there:

```
#include <Sparki.h>  // include the sparki library

void setup()
{
}

void loop()
{
  sparki.clearLCD();

  sparki.drawRect(5,5, 30,10);
  sparki.drawRectFilled(15,17, 30,10);

  sparki.drawCircle(55,30, 5);
  sparki.drawCircle(20,45, 12);

  sparki.drawCircleFilled(90,40, 20);

  sparki.updateLCD();
  delay(1000);
}
```

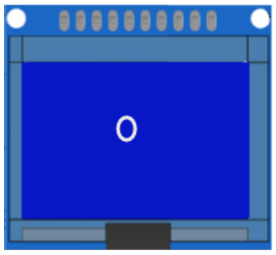
Rectangles- There are four coordinates inside the parenthesis. The first two numbers are the coordinates of one of the corners of the rectangle. The second two numbers are the coordinates of the corner that is the diagonal opposite of the first. Really, the second set of numbers could be the upper left corner and the first set of numbers could be the lower left corner of the rectangle and you could draw the same rectangle using sparki.drawRectFilled(15,17, 30,10) as you can with the sparki.drawRectFilled(30,10,15,17)

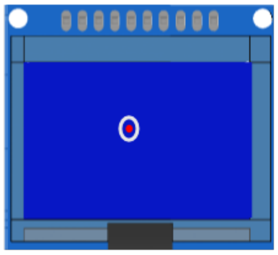

drawRect(5,5, 30,10);

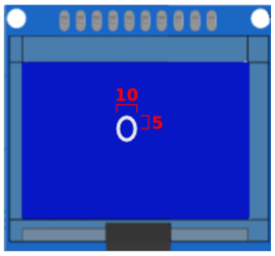

5,5 coordinate


30,10 coordinate

Circles- We'll focus on the command sparki.drawCircle(55,30, 5) The first two numbers are the X and Y coordinates of the center of the circle Sparki is going to draw. This is exactly the same as the code for drawing a pixel. The main difference is that Sparki will draw a circle around this point. The radius of the circle is the third number, 5. So this code will draw a circle that is 10 pixels tall (the diameter) and 10 pixels wide with a center at 55, 30.


drawCircle(55, 30, 5);


center coordinates 55, 30



radius of 5, width of 10

Writing Text and Numbers

Another important thing that we can do with the LCD display is writing text and numbers. This is very useful, especially once we start to work with Sparki's sensors since we can easily use the LCD to display the values of the sensors Sparki is reading. This way, if something is not working as we expect in a program using one or more sensors, we can see the exact readings on the display. You can also use the LCD to print out where Sparki is in the code you have written. There are two ways of writing text: The first would be to write each individual letter. The easier way is just printing the text, and the Sparki LCD will automatically position each printed line after the other in the LCD. Please note that in the example below, the **println()** function can both print text strings and numbers. To have what you print stay on the same line, use print() instead. The "ln" is short for "line" which is short for "newline".

```
#include <Sparki.h>  // include the sparki library

void setup()
{
}

void loop()
{
  sparki.clearLCD(); // wipe the LCD clear
  sparki.print("My name is: ");
  sparki.println("Sparki");

  sparki.println(123);
  sparki.println(456.5);
  sparki.updateLCD(); // put the drawings on the screen
  delay(1000);
}
```

The other way of writing a text in the LCD is a bit more tricky, but it will enable us to position a character or a text string anywhere on the display:

```
#include <Sparki.h>  // include the sparki library

void setup()
{
}

void loop()
{
  sparki.clearLCD(); // wipe the LCD clear
  sparki.drawChar(10, 1, 'a');
  sparki.drawChar(20, 2, 'b');
  sparki.drawChar(30, 3, 'c');

  sparki.drawString(40, 4, "123");
  sparki.updateLCD(); // put the drawings on the screen
  delay(1000);
}
```

With this second way of displaying letters or Strings you can make the words, letters or numbers appear anywhere, which is useful if you are animating something with text or making a diagram of something (maybe a diagram of Sparki) and you want to label parts of the diagram. With the first method you have no control of where the text will show up, but it's great if you just need to display information and don't care what it looks like or where it is on the screen. When using the second method the first two numbers are X and Y coordinates, while the third value inside the parenthesis is the text you wish to display. Please take into account that if we want to print numbers using this last method, we need to convert them as shown in the following example, using the string class from the standard Arduino API:

```
#include <Sparki.h>  // include the sparki library

void setup()
{
}

void loop()
{
  sparki.clearLCD(); // wipe the LCD clear
  sparki.drawChar(10, 1, 'a');
  sparki.drawChar(20, 2, 'b');
  sparki.drawChar(30, 3, 'c');

  char text[20];
  String number = String(123);
  number.toCharArray(text, sizeof(text));

  sparki.drawString(40, 4, text);
  sparki.updateLCD(); // put the drawings on the screen
  delay(1000);
}
```

(***NOTE:*** *if you don't yet understand the conversions used here, don't worry! We are just showing a little more advanced way of visualizing numbers, but we will primarily rely on the simpler* ***print()*** *function in our future lessons*)

Simple Animations

The last thing that we will learn here is how to do some small and (very) simple animations with our display. Let's start with a simple circle that will move across the screen: Here is the code:

```
#include <Sparki.h>  // include the sparki library

int x = 0;

void setup()
{
}

void loop()
{
  sparki.clearLCD(); // wipe the LCD clear

  if (x < 127)
    x++;
  else
    x = 0;

  sparki.drawCircleFilled(x,32, 10); // small filled circle vertically centered

  sparki.updateLCD(); // put the drawings on the screen

  delay(100); // delay for the animation
}
```

See the trick? We have used a variable which changes instead of a fixed value for the X coordinate. We increment that variable (called “x’) each time through the loop cycle. Once the variable reaches its maximum value (which we have set as 127 here), we reset ‘x’ to zero. This way, the circle crosses the small screen over and over again. You can experiment modifying both the delay time and the **x** variable increment. Regarding the delay time, please note that if you do it too fast, you will not give enough time to the microprocessor to render the images on the LCD display, but push it a bit so you can learn about its graphical processing capabilities! How could you change the code that effects **x** so that the ball moves faster? How about slower? Don’t just think about it, try out different code on Sparki and see how it works! What about making the ball bounce back and forth instead of starting at one side of the screen and traveling to the other side? Here’s some code to get you started on that project, it makes the ball start on one side of the screen and bounce when it hits the right hand side. Making it bounce when it hits the left hand side is up to you.

```
#include <Sparki.h> // include the sparki library

int x = 0;
boolean hitRight;

void setup()
{
}

void loop()
{
  sparki.clearLCD(); // wipe the LCD clear

  if (x < 127 && hitRight == false) //changed this
    x++;
  else
    x--; //changed this

  if(x > 126) //added this
    hitRight = true; //added this

  sparki.drawCircleFilled(x,32, 10); // small filled circle vertically centered

  sparki.updateLCD(); // put the drawings on the screen

  delay(100); // delay for the animation
}
```

(Hint, start with making another variable called hitLeft.)

Another interesting thing that we can animate besides the *position* of an object is its *size*. Let’s try the following program to animate the height of a vertical bar (made with a rectangle):

```
#include <Sparki.h>  // include the sparki library

int y = 10;

void setup()
{
}

void loop()
{
   sparki.clearLCD(); // wipe the LCD clear

   if (y < 60)
     y += 10;
   else
     y = 0;

   sparki.drawRectFilled(10,0, 15, y);

   sparki.updateLCD(); // put the drawings on the screen

   delay(150); // delay for the animation
}
```

Having Fun with Animation

Drawing Eyes- Now let’s do a little bit of animation that gives Sparki some eyes and makes them look right or left depending on which way Sparki turns. The first things we need to draw are some plain old eyes. We can do that using the drawCircle() command. We’ll draw to large unfilled circles for the eyes with two smaller filled circles inside for the pupils.

```
#include &lt;Sparki.h&gt; // include the sparki library

void setup()
{
}

void loop()
{
  sparki.clearLCD(); // wipe the LCD clear

  sparki.drawCircle(32, 25, 20); //eye 1
  sparki.drawCircle(96, 25, 20); //eye 2

  sparki.drawCircleFilled(32, 25, 5); //pupil 1
  sparki.drawCircleFilled(96, 25, 5); //pupil 2

  sparki.drawLine(15,15, 49,15);//eyelid 1
  sparki.drawLine(79,15, 112,15);//eyelid 2

  sparki.updateLCD(); // put the drawings on the screen

  delay(100); // delay for the animation
}
```

Making the Eyes Blink- Now, for fun, we'll make the eyes blink. That means we have to move the code around a little and make a counter so that Sparki only blinks every once in a while. We'll also make our own functions so we can choose what to draw.

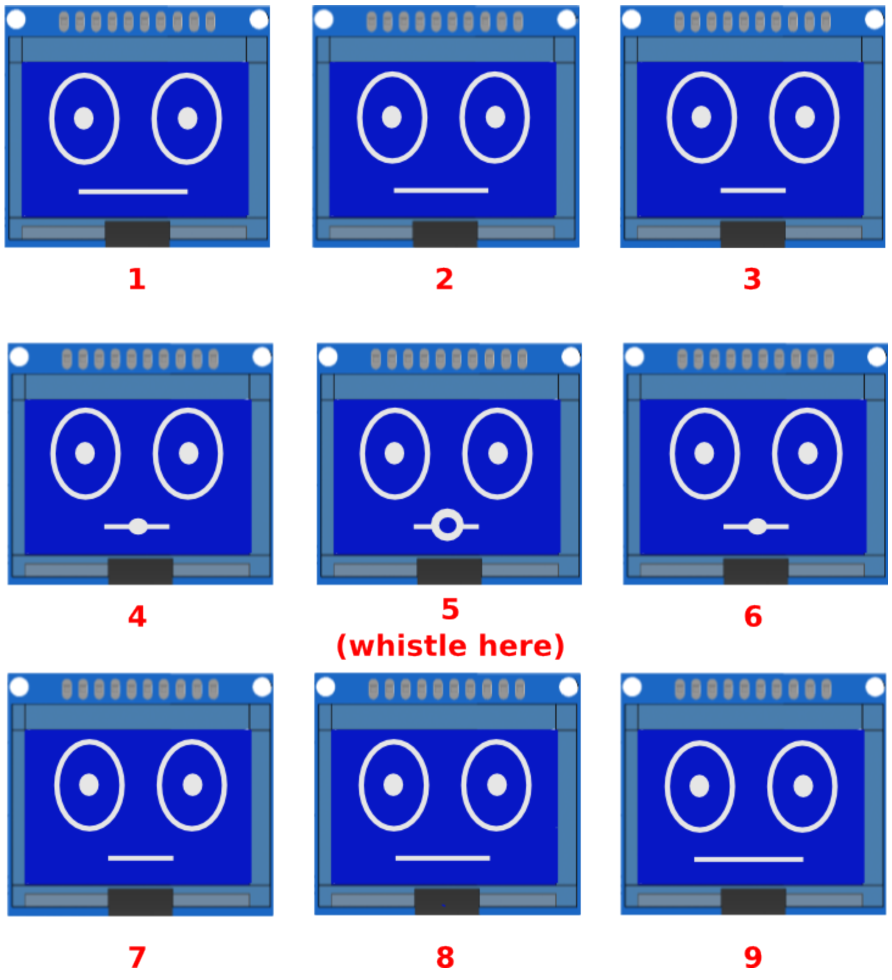
```
#include &lt;Sparki.h&gt; // include the sparki library

int blinkCounter;

void setup()
{
}

void loop()
{
  sparki.clearLCD(); // wipe the LCD clear
  if(blinkCounter > 195)
    blinkEyes();
  else
    drawEyes();

  sparki.updateLCD(); // put the drawings on the screen

  delay(100); // delay for the animation

  blinkCounter ++; //make counter count up
  if(blinkCounter == 200){
    blinkCounter = 0; //reset counter
  }
}

void drawEyes()
{
  sparki.drawCircle(32, 25, 20); //eye 1
  sparki.drawCircle(96, 25, 20); //eye 2

  sparki.drawCircleFilled(32, 25, 5); //pupil 1
  sparki.drawCircleFilled(96, 25, 5); //pupil 2

  sparki.drawLine(15,15, 49,15);//eyelid 1
  sparki.drawLine(79,15, 112,15);//eyelid 2
}

void blinkEyes()
{
  sparki.drawCircle(32, 25, 20); //eye 1
  sparki.drawCircle(96, 25, 20); //eye 2

  sparki.drawLine(12,24, 52,24);//eyelid 1
  sparki.drawLine(76,24, 115,24);//eyelid 2

  sparki.drawLine(12,25, 52,25);//eyelid 3
  sparki.drawLine(76,25, 115,25);//eyelid 4
}
```

See how useful counters are? If you're really looking for a challenge try adding eyelashes to your animation. Don't forget that the eyelashes will need to move when Sparki blinks!

Making the Eyes Look Right and Left- Now we're going to combine what we learned in the moving ball section and our current code. We'll write some code that makes Sparki's pupil's look right and left while Sparki turns left and right. The code that deals with right hand turns you can find under **File->Sketchbook->Bright Ideas LCD**, I'll leave the left hand turns up to you.

Additional Challenges

After you've made Sparki look to the left, how about giving your Sparki face a nose and a mouth? Looking for an animation challenge? Try giving Sparki a straight line for a mouth and every once in a while making the line get shorter until it is only seven or eight pixels wide and then making a circle that gets larger until Sparki looks like it is whistling. (Hint, you may need to draw two lines and a circle between then to do this.) During this portion of the animation you can even make Sparki's piezo buzzer (which you'll learn about in the next guide) whistle a little tune to match the animation! Here's a picture to help you figure out what each frame in your animation should look like


1
2
3
4
5
(whistle here)
6
7
8
9