THIS DOCUMENT IS DISTRIBUTED STRICTLY "AS-IS", as a "user contributed" document written by Clark Jones, with absolutely no warranties implied or expressed.

To be quite honest, the documentation for the SparkFun Alphanumeric Display isn't all that good, but that's not surprising since they are "Spark-X" products and so are things that are put out there as "experimental" rather than fully-qualified production units.

I'm writing this to give myself a "reference document", and in order to improve the situation for others have decided to make it available in the spirit of "OSS".

This info comes from digging through two files, both found in
~/Documents/Arduino/libraries/SparkFun_Qwiic_Alphanumeric_Display_Arduino_Library/src
(note that I'm using OSX, the path may be different on an M$ or a Linux box).
The two files are named SparkFun_Alphanumeric_Display.cpp and SparkFun_Alphanumeric_Display.h.

These files communicate (via I2C, usually using a Qwiic connection) from the processor to the HT16K33 device on the Display board, which in turn controls the four 14-segment (plus colon and one period) LEDs. I did eventually find a copy of the manufacturer's datasheet for the LEDs in the hardware GitHub library (under "DOCUMENTS") for the boards, but it's not easy to read, and has a slightly different segment name assignment than the software uses.

The "class" for the software is HT16K33, so you'll need to have an instantiation of it with a statement something like

```
HT16K33 display;
```

Anyway, here's a recap of the available functions, with a bit better description of what they do:

```
bool begin(uint8_t addressLeft = DEFAULT_ADDRESS,
        uint8_t addressLeftCenter = DEFAULT_NOTHING_ATTACHED,
        uint8_t addressRightCenter = DEFAULT_NOTHING_ATTACHED,
        uint8_t addressRight = DEFAULT_NOTHING_ATTACHED,
        TwoWire &wirePort = Wire);
```

This function initializes the "private" portions of the library, and checks to see that it can talk to the HT16K33. If it succeeds, it will return true, else it will return false. Note also that since every argument has default values, you can call it with just display.begin() being careful to check the return value. For the four addresses, DEFAULT_ADDRESS is defined as 0x70 and DEFAULT_NOTHING_ATTACHED is 0xFF. If you use multiple units, you'll have to add solder jumpers to A0 and/or A1, and adjust the address specified here appropriately.

```
bool isConnected(uint8_t displayNumber);
```

Checks to see that there's a display connected. Note that the argument is NOT the I2C address of the display.

```
bool initialize();
```

Initializes all of the displays that the instance knows about, turning on the "system" (scan) clock, maximum brightness, blinking off, and display on.

```
bool clear();
```
Clears all of the displays that the instance knows about.

```
bool setBrightness(uint8_t duty);
```
Sets the brightness of all the displays that the instance knows about. duty is an integer of the number of 16ths duty cycle.

```
bool setBrightnessSingle(uint8_t displayNumber, uint8_t duty);
```
Sets brightness of a single display. duty is an integer of the number of 16ths duty cycle.

```
bool setBlinkRate(float rate);
```
Sets the blink rate of all the displays that the instance knows about. Valid values for rate are 2, 1, or 0.5 and are expressed in Hz (i.e., blinks per second). An invalid value will result in no blinking.

```
bool setBlinkRateSingle(uint8_t displayNumber, float rate);
```
Sets the blink rate of a single display. Valid values for rate are 2, 1, or 0.5 and are expressed in Hz (i.e., blinks per second). An invalid value will result in no blinking.

```
bool displayOn();
bool displayOff();
```
Turns all of the displays on or off.

```
bool displayOnSingle(uint8_t displayNumber);
bool displayOffSingle(uint8_t displayNumber);
bool setDisplayOnOff(uint8_t displayNumber, bool turnOnDisplay);
```
Turn a single display on or off.

```
bool enableSystemClock();
bool disableSystemClock();
bool enableSystemClockSingle(uint8_t displayNumber);
bool disableSystemClockSingle(uint8_t displayNumber);
```
Enabling or disabling the “system” clocks for displays is not something most users will need, but it's there if you do need it.

```
void illuminateSegment(uint8_t segment, uint8_t digit);
```
Will turn on a single segment (can be called multiple times to turn on multiple segments) in a given digit. The segment values are in the range 'A' – 'N' (inclusive) and the digit is in the range 0-15. Note that it will not take effect until a call to updateDisplay();

```
void illuminateChar(uint16_t disp, uint8_t digit);
```
Will turn on segments indicated by a '1' bit in disp. digit is in the range 0-15. Note that the '0' bits will not be specifically turned off. Also note that it will not take effect until a call to updateDisplay();

```
void printChar(uint8_t displayChar, uint8_t digit);
```
If displayChar is a “printable” character (with exception of ':' and '.') it will be displayed in the position indicated by digit (range 0-15). Note that it will not take effect until a call to updateDisplay();

```
bool updateDisplay();
```
Sends the contents of the display buffer to the display.

```
bool defineChar(uint8_t displayChar, uint16_t segmentsToTurnOn);
```

Allow the user to change the segments displayed for a given character (in the range '!' - '~', inclusive). Note that this will only be in effect until the controller is rebooted, as it is done as a table on the controller to what segments to send to the HT16K33s. The least significant bit of segmentsToTurnOn represents segment 'A'.

```
uint16_t getSegmentsToTurnOn (uint8_t charPos);
```

Used internally.

Note that each display has only one decimal and one colon, and that these are controlled separately from the "text" that gets sent to the displays. The names of thee next several functions are pretty much self explanatory.

```
bool decimalOn();
bool decimalOff();
bool decimalOnSingle(uint8_t displayNumber);
bool decimalOffSingle(uint8_t displayNumber);
bool setDecimalOnOff(uint8_t displayNumber, bool turnOnDecimal);
bool colonOn();
bool colonOff();
bool colonOnSingle(uint8_t displayNumber);
bool colonOffSingle(uint8_t displayNumber);
bool setColonOnOff(uint8_t displayNumber, bool turnOnColon);
```

The functions above that have the displayNumber argument will only affect the specified display. Those without it will affect all of the displays.

The datasheet for the LED display is a bit hard to find: it's in the Github repository for the hardware documents. And once you do find it, it's a bit difficult to read. The critical thing for most users is the mapping of segment letters to the segment positions. Here's what I've found (experimentally):

'A' – top horizontal
'B' – upper right vertical
'C' – lower right vertical
'D' – bottom horizontal
'E' – lower left vertical
'F' – upper left vertical
'G' – middle left horizontal
'H' – upper left diagonal
'I' – middle right horizontal
'J' – upper middle vertical
'K' – upper right diagonal
'L' – lower right diagonal
'M' – lower middle vertical
'N' – lower left vertical

Note 1: What is called "G" here is called "G1" on the datasheet, and what is called "I" here is "G2" on the datasheet.

Note 2: The colon and decimal point are not assigned segment letters.