Special Commands

Nathan Seidle edited this page on 31 Jan 2017 · 26 revisions

This page will discuss the Serial 7-Segment Display's special commands:

1. Clear display
2. Cursor control
3. Decimal, colon, and apostrophe control
4. Brightness control
5. Individual segment control
6. Baud rate configuration
7. I2C address configuration
8. Factory reset

First, a quick breakdown of the commands, their control byte and any data bytes:

Special Command	Command byte	Data byte range	Data byte description
Clear display	0x76	None	
Decimal control	0x77	0-63	1 bit per decimal
Cursor control	0x79	0-3	0=left=most, 3=right-most
Brightness control	0x7A	0-100	0=dimmest, 100=brightest
Digit 1 control	0x7B	0-127	1 bit per segment
Digit 2 control	0x7C	0-127	1 bit per segment
Digit 3 control	0x7D	0-127	1 bit per segment
Digit 4 control	0x7E	0-127	1 bit per segment

Clone this wiki locally

```
https://github.com/sparkf
```

Baud rate config	0x7F	0-11	See baud section
I2C address Config	0x80	1-126	Data byte is I2C addres
Factory reset	0x81	None	

These commands can be sent to the display using any of the three communication methods (serial, SPI, I2C). For ease of understanding, Arduino examples are shown using serial.

Clear Display

The clear display command performs two functions:

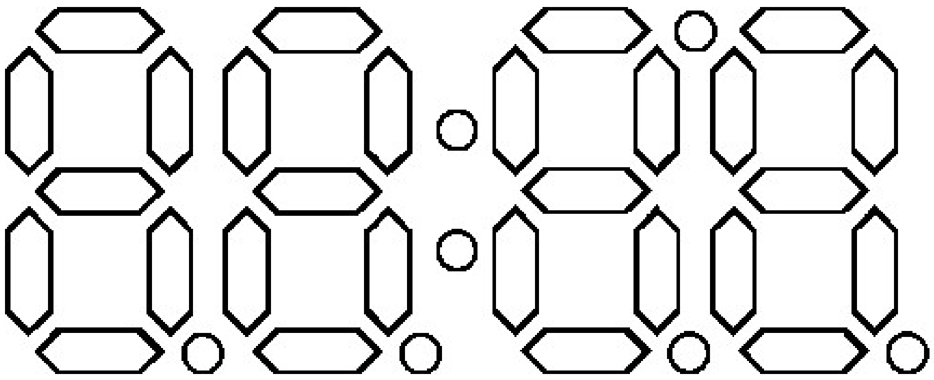
1. Clear the display - all LEDs, including segments and decimal points, are turned off.
2. Reset the cursor to position 1, the left-most digit.

The clear display command byte is `0x76`.

There is no data byte, so any displayable data sent after the clear display command will be displayed on digit 1.

Arduino Sample Snippet (Serial Mode): To make the display read *12Ab*., we can't be guaranteed that the cursor is at position 1. To ensure that it is, we can use the clear display command before sending our data.

```
// ... after initializing Serial at the correct baud rate...
Serial.write(0x76);  // Clear display command, resets cursor
Serial.write(0x01);  // Hex value for 1, will display '1'
Serial.write('2');   // ASCII value for '2', will display '2'
Serial.write(0x0A);  // Hex value for 10, will display 'A'
Serial.write('B');   // ASCII value for 'B', will display 'b'
```



Note: The clear display command byte value is equivalent to the ASCII value for the 'v' character. This value was chosen because 'v' is not all that displayable on a 7-segment display.

Cursor Control

You can control the cursor using the cursor control command. To move the cursor, first send the cursor control byte `0x79` , then send an 8-bit data byte with **value between 0 and 3**. A data value of 0 will set the cursor to position 1 (left-most), a value of 3 will set the cursor to the right-most digit.

###Example: Using the Move Cursor Command To set the cursor to the second digit (the digit immediately left of the colon), send the following 2-byte sequence: `[0x79][0x01]`

Sample Arduino Snippet (Serial Mode):

```
// ... after initializing Serial at the correct baud rate...
Serial.write(0x79); // Send the Move Cursor Command
Serial.write(0x01); // Send the data byte, with value 1
Serial.write(7);    // Write a 7, should be displayed on 2nd digit
```

If the data byte is outside the allowable range (0-3), the cursor command is ignored.

Or, you can reset the cursor to position 1 by issuing the clear display command.

Decimal, Colon and Apostrophe Control

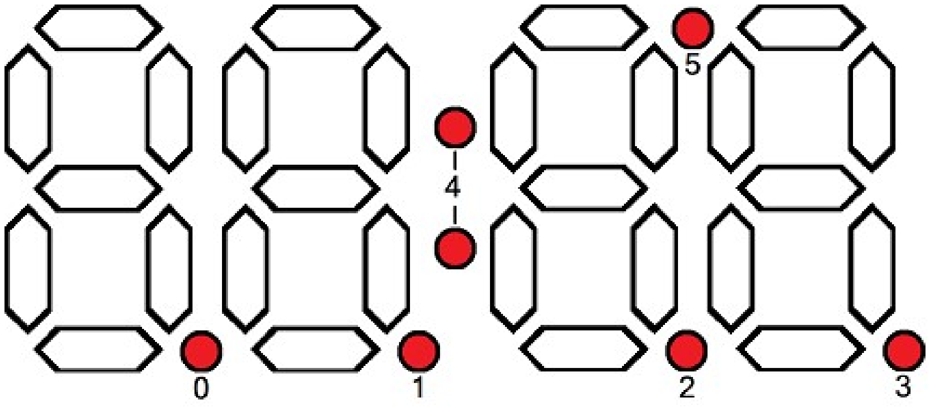
This command gives you control over each of the four decimal points, the colon, and the apostrophe between digits 3 and 4.

To turn on or off any of the decimal points first send the decimal control command `0x77` followed by a data byte.

Each of the six least significant bytes of the **data byte** represent one decimal point (or two in the case of the colon). A 1 will turn the decimal point on, a 0 will turn it off. The most significant two bits in the data byte have no effect on the display.

The following figure shows the bit representation for each decimal point (0 is the least significant bit):

Bit:	7 (msb)	6	5	4	3	2	1
Segment:	X	X	Apostrophe	Colon	Digit 4	Digit 3	Digit 2



Note: Because they're wired together inside the LED, the two decimal points cannot be individually controlled. Both will either be on or off.

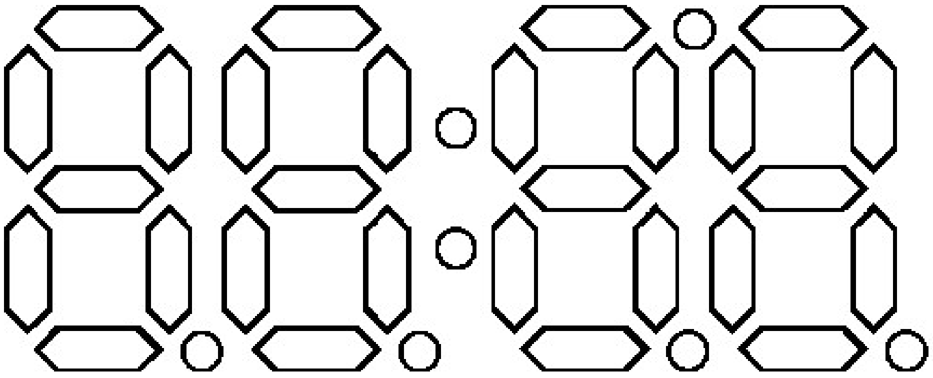
Example: Turning on the colon, apostrophe, and far-right decimal point

To turn on the colon, apostrophe, and far-right decimal point we need to set bits 4, 5, and 3, respectively. Our data byte will therefore be: `0b00111000` (ie. hex `0x38` , or decimal `56`).

So a two byte sequence of `0x77``0x38` should be sent.

Sample Arduino Snippet (Serial mode)

```
// ... after initializing Serial at the correct baud rate...
Serial.write(0x77);  // Decimal control command
Serial.write(0b00111000);  // Turns on colon, apostrophoe, and far-rig
```



Brightness Control

The brightness of the display can be controlled via internal PWM.

To control the brightness of the display, first send the special command character `0x7A` , followed by a data byte. The data byte can be any number **between 0 and 100**. 0 represents the dimmest setting, while 100 sets the display to its brightest. A value greater than 100 will be rendered as 100.

After the screen receives the brightness command, the brightness value is written into the processor's non-volatile memory. So the display will retain the same brightness level upon power cycling.

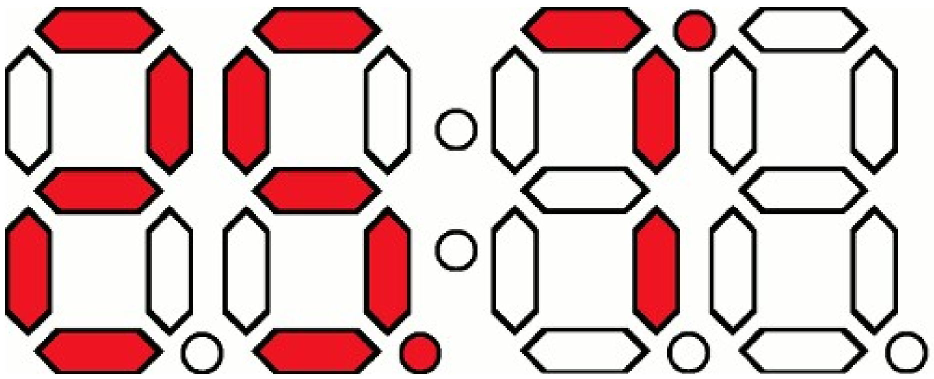
Note: The brightness command will have no effect on the currently displayed values. The display will remain the same.

Example: Setting the display to its dimmest level

To set the display to the dimmest possible setting, we need to send 0 for the data byte. The required two-byte stream would be: `0x7A``0x00` .

Sample Arduino Snippet (Serial Mode)

```
// ... after initializing Serial at the correct baud rate...
Serial.write(0x7A);  // Brightness control command
Serial.write((byte) 0);  // dimmest value (must type-def 0)
```



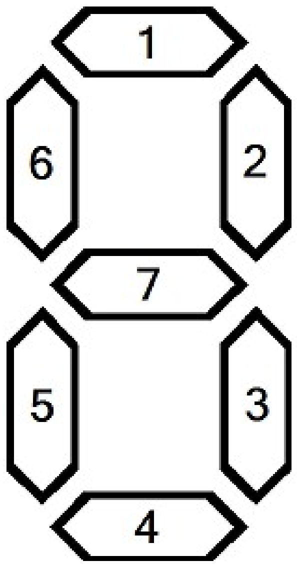
Individual Segment Control

Single character control allows you to control each individual segment of each digit. Four different command bytes control each of the four digits:

Digit Position	Command Byte

1	0x7B
2	0x7C
3	0x7D
4	0x7E

Like the decimal command, the data byte uses individual bits to control each of the segments. The 7 least-significant bits control each of the segments, as diagrammed below:



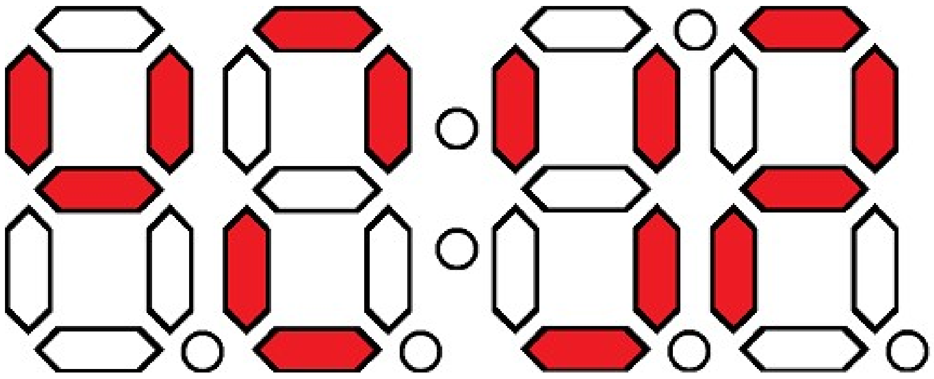
Bit:	7 (msb)	6	5	4	3	2	1	0 (lsb)
Segment:	X	7	6	5	4	3	2	1

The most significant bit has no effect on the display.

The individual segment control commands will have no effect on the cursor, which will remain at whatever position it was before the command was issued.

Example: Making wacky displays

Maybe you've got a hankering to create some sort of knock-off *Predator* self-destruct clock? And need to create a new numeral system:



The display example above would require the following byte sequence: `0x7B``0b01100010` `0x7C``0b00011011` `0x7D``0b00101110` `0x7E``0b01010011`

Arduino Sample Snippet (Serial Mode):

```
// ... after initializing Serial at the correct baud rate..
Serial.write(0x76);  // Clear the display
Serial.write(0x7B);  // Digit 1 control
Serial.write(0b01100010);  // set segments B, F, G
Serial.write(0x7C);  // Digit 2 control
Serial.write(0b00011011);  // set segments A, B, D, E
Serial.write(0x7D);  // Digit 3 control
Serial.write(0b00101110);  // set segments B, C, D, F
Serial.write(0x7E);  // Digit 4 control
Serial.write(0b01010011);  // set segments A, B, E, G
```

Baud Rate Configuration

By default the display's baud rate is set to 9600bps. This can be configured using the baud rate configuration command: `0x7F` .

This command should be followed by a single data byte, between 0 and 11. The following table matches the data byte value with the corresponding baud rate setting:

Data Byte Value	Baud rate setting (bps)
0	2400
1	4800
2	9600
3	14400
4	19200

5	38400
6	57600
7	76800
8	115200
9	250000
10	500000
11	1000000

Your mileage may vary (YMMV) with the higher baud rates (higher than 57600), where the error increases significantly.

Once the baud rate command is received, and accepted, the value is stored in **non-volatile memory**. The display will retain this baud setting even upon losing and regaining power, until it is changed again.

Note: Data byte values out of the 0-11 range will cause the baud rate configuration command to be ignored.

Setting the baud rate does not clear the display or reset the cursor.

I2C Address Configuration (0x80)

If you're using more than one display on the same I2C bus, you'll need to change at least one of their addresses. The I2C address configuration command is `0x80`.

A single data byte should follow the command. The data byte will represent the devices *new* I2C address. 7-bit values between 0x01 and 0x7E (1-126) are acceptable. Any values outside this range will be ignored.

Once the I2C address command and a valid data byte have been received, the new I2C address will be stored in **non-volatile memory**. This means the display will retain its new I2C address even up power cycling.

###Example: Setting the I2C Address to 0x42 The following two-byte sequence could be used to set the display's address to 0x42: `0x80``0x42`.

It may be handy to send this command in Serial mode. Before connecting the display to your I2C bus.

Sample Arduino Snippet (Serial Mode)

```
// ... after initializing Serial at the correct baud rate..
Serial.write(0x80);  // I2C Address Config command
Serial.write(0x42);  // Set 7-bit address to 0x42
```

Factory Reset (0x81)

If your display is unresponsive (usually this is due to an unknown baud rate setting), try using the factory reset command: `0x81` . This command does not have a trailing data byte.

After receiving this command, the display performs three actions:

1. Set the baud rate to 9600 bps. This setting is saved in non-volatile memory.
2. Set the I2C address to the default 0x71. This is saved in non-volatile memory too.
3. Rest the brightness to 100%. Again, stored in non-volatile memory.

###Example: Recovering from an unknown baud rate The most common problem with these displays is when they're accidentally configured to an undesired baud rate. They become unresponsive to the expected baud rate, and because this setting is stored in memory it can be hard to recover from. **Side-note**: If you have the display connected to your Arduino's hardware serial pins, which are also used to program the board, there's a chance the display could receive commands - like baud setting - that weren't meant for it.

The following lines of Arduino code have come in handy, when you need to bring the display back to its default setting:

```
int baudRates[12] = {2400, 4800, 9600, 14400, 19200, 38400,
57600, 76800, 115200, 250000, 500000, 1000000};
for (int i=0; i<12; i++)
{
  Serial.begin(baudRates[i]);  // Set new baud rate
  delay(10);  // Arduino needs a moment to setup serial
  Serial.write(0x81);  // Send factory reset command
}

Serial.begin(9600);
delay(10);  // Arduino needs a moment to setup serial
Serial.write(0x76);  // Clear the display
Serial.write('t');
Serial.write('e');
Serial.write('s');
Serial.write('t');
```

The factory reset command is sent from all possible baud rates. The display should show *teSt* once properly configured to 9600 bps.

Datasheet Links:

- Datasheet Homepage
- [Hardware-Specifications](Hardware Specifications) - Electrical characteristics, voltage ratings, current usage, timing
- [Interface-Specifications](Interface Specifications) - UART, SPI, and I2C explanations
- Basic Usage - Displaying Numbers and Characters, Clearing, Cursor Control
- [Arduino-Examples](Arduino Examples) - Example Code for SPI and I2C
- [Customizing-the-Display](Customizing the Display) - Uploading a custom Arduino Sketch