PJRC Store

- Teensy 4.1, $26.85
- Teensy 4.0, $19.95
- Teensy 3.6, $29.25
- Teensy 3.5, $24.25
- Teensy 3.2, $19.80
- Teensy LC, $11.65
- Teensy 2.0, $16.00
- Teensy++ 2.0, $24.00

Teensy

- Main Page
- ⊞ Hardware
- ⊞ Getting Started
- ⊞ Tutorial
- ⊞ How-To Tips
- ⊞ Code Library
- Projects
- ▶ Teensyduino
  - Main
  - Download+Install
  - Basic Usage
  - Digital I/O
  - PWM & Tone
  - ⊞ Timing
  - USB Serial
  - USB Keyboard
  - USB Mouse
  - USB Joystick
  - USB MIDI
  - USB Flight Sim
  - Serial
  - ▶ Libraries
    - Main List
    - GLCD
    - LiquidCrystal
    - OctoWS2811
    - FastSPI_LED
    - Matrix/Sprite
    - LedDisplay
    - LedControl
    - DogLcd
    - ST7565
    - AltSoftSerial
    - NewSoftSerial
    - SoftwareSerial
    - MIDI
    - PS2Keyboard
    - DmxSimple
    - Firmata
    - ▶ Wire
    - SPI
    - OneWire
    - XBee
    - VirtualWire
    - X10
    - IRremote
    - TinyGPS
    - USBHostShield
    - Ethernet
    - Bounce
    - Keypad
    - ⊞ Audio
    - Encoder
    - Ping
    - CapacitiveSensor
    - FreqCount
    - FreqMeasure
    - Servo
    - PulsePosition
    - Stepper
    - AccelStepper
    - FrequencyTimer2
    - Tlc5940
    - SoftPWM
    - ShiftPWM
    - Time
    - TimeAlarms
    - DS1307RTC
    - Metro

Wire Library

The Wire library allows you to communicate with I^{2}C devices, often also called "2 wire" or "TWI" (Two Wire Interface).

Download: Wire is included with Arduino

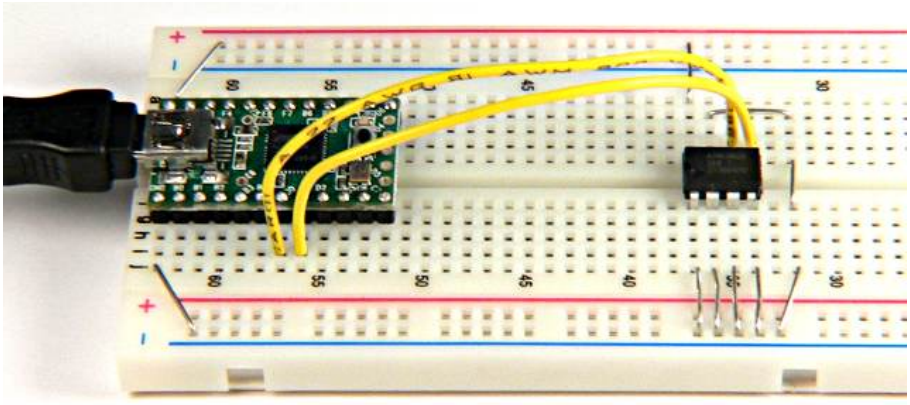
Brian "nox771" has written an improved I2C library for Teensy 3.0.

Hardware Requirements

I^{2}C devices communicate with 2 signals, called SDA and SCL. Normally a 4.7K pullup resistor is connected between each signal and power (+3.3V on Teensy 3.0, +5V on Teensy 2.0). On Teensy 2.0, 4.0, and 4.1, the weak internal pullup resistors may be sufficient for short wires to a single device. Because the internal resistors are so weak, communication may be slower or unreliable if the wires are long.

Signal	Teensy 2.0	Teensy++ 2.0	Teensy LC	Teensy 3.0 - 3.6	Teensy 4.0, 4.1
SCL	Pin 5	Pin 0	Pin 19	Pin 19	Pin 19
SDA	Pin 6	Pin 1	Pin 18	Pin 18	Pin 18

Teensy LC & 3.0-3.6 requires pullup resistors to +3.3V. The on-chip pullups are not used. 4.7K resistors are recommended for most applications.



The Wire library is not compatible with Teensy 1.0.

Basic Usage

Wire.begin()

Begin using Wire in master mode, where you will initiate and control data transfers. This is the most common use when interfacing with most I^{2}C peripheral chips.

Wire.begin(address)

Begin using Wire in slave mode, where you will respond at "address" when other I^{2}C masters chips initiate communication.

Transmitting

Wire.beginTransmission(address)

Start a new transmission to a device at "address". Master mode is used.

Wire.write(data)

Send data. In master mode, beginTransmission must be called first.

Wire.endTransmission()

In master mode, this ends the transmission and causes all buffered data to be sent.

Receiving

Wire.requestFrom(address, count)

Read "count" bytes from a device at "address". Master mode is used.

Wire.available()

- TimerOne
- MsTimer2
- EEPROM

⊞ **Reference**

Retuns the number of bytes available by calling receive.

Wire.read()

Receive 1 byte.

Pin Configuration (Teensy LC & 3.x Only)

Wire.setSDA(pin)

Configure the pin used for data. Possible alternate pins are shown on the pinout card. This function may be called before or after Wire.begin(). All pins used must have real pullup resistors.

Wire.setSCL(pin)

Configure the pin used for clock. Possible alternate pins are shown on the pinout card. This function may be called before or after Wire.begin(). All pins used must have real pullup resistors.

Responding in Slave Mode

Wire.OnReceive(myReceiveHandlerFunction)

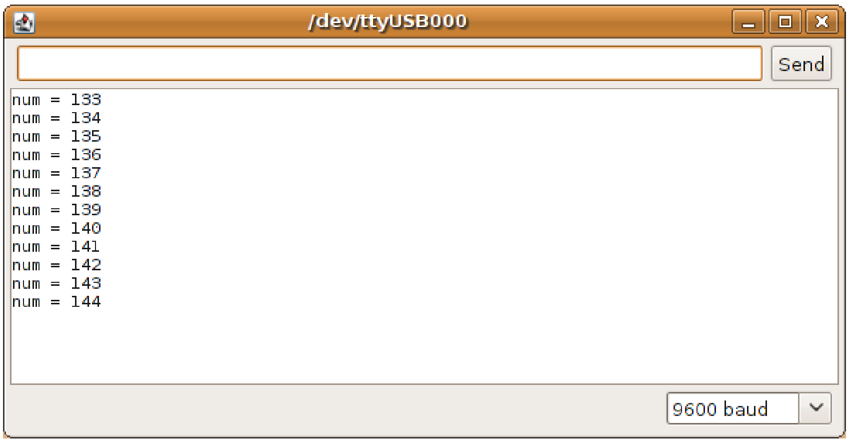
Causes "myReceiveHandlerFunction" to be called when a master device sends data. This only works in slave mode.

Wire.OnRequest(myRequestHandlerFunction)

Causes "myRequestHandlerFunction" to be called when a master device wishes to read data. This only works in slave mode.

Example Program

This simple example uses a 24C256 I^{2}C EEPROM. The first byte is read, incremented, and written back.



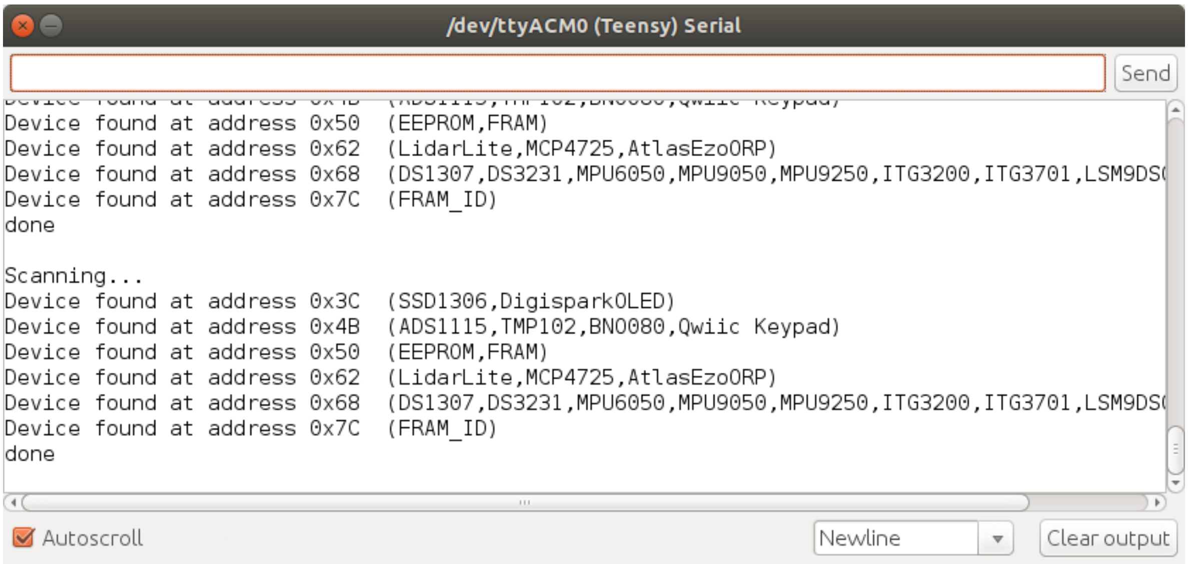
```
num = 133
num = 134
num = 135
num = 136
num = 137
num = 138
num = 139
num = 140
num = 141
num = 142
num = 143
num = 144
```

9600 baud

```
#include <Wire.h>

void setup()
{
  Wire.begin();
  Serial.begin(9600);
}

void loop()
{
  byte num=0;

  // set the 24C256 eeprom address to 0
  Wire.beginTransmission(80);
  Wire.write(0);  // address high byte
  Wire.write(0);  // address low byte
  Wire.endTransmission();

  // read 1 byte, from address 0
  Wire.requestFrom(80, 1);
  while(Wire.available()) {
    num = Wire.read();
  }
  Serial.print("num = ");
  Serial.println(num, DEC);

  // increment num
  num = num + 1;

  // write "num" to 24C256 eeprom at address zero
  Wire.beginTransmission(80);
  Wire.write(0);    // address high byte
  Wire.write(0);    // address low byte
  Wire.write(num);  // any more send starts writing
  Wire.endTransmission();

  // next time loop runs, it should retrieve the
  // same number it wrote last time... even if you
  // shut off the power
  delay(5000);
}
```

Scanner Program

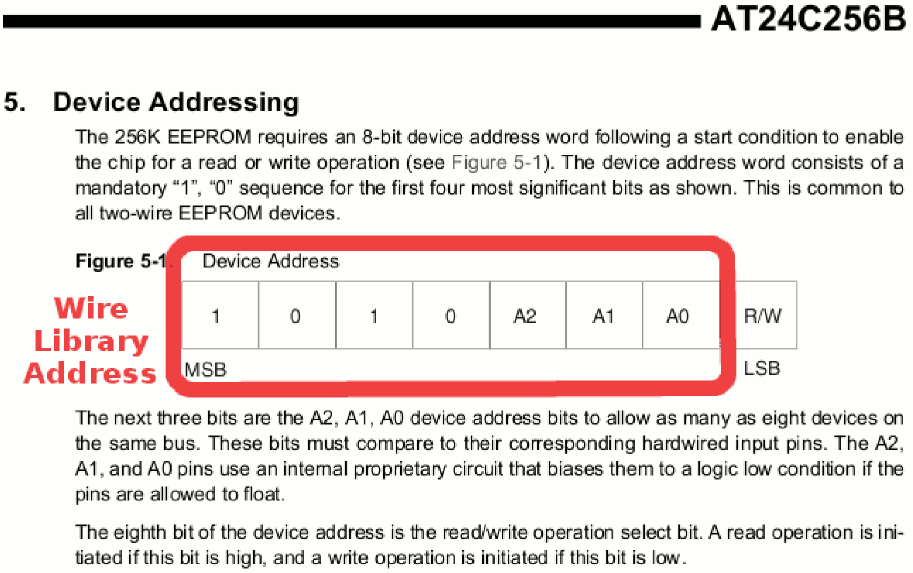
The Wire library comes with an example program which scans for all I2C devices. Open it from **File > Examples > Wire > Scanner**.



```
Device found at address 0x4B  (ADS1115,TMP102,BNO080,Qwiic Keypad)
Device found at address 0x50  (EEPROM,FRAM)
Device found at address 0x62  (LidarLite,MCP4725,AtlasEzoORP)
Device found at address 0x68  (DS1307,DS3231,MPU6050,MPU9050,MPU9250,ITG3200,ITG3701,LSM9DS
Device found at address 0x7C  (FRAM_ID)
done

Scanning...
Device found at address 0x3C  (SSD1306,DigisparkOLED)
Device found at address 0x4B  (ADS1115,TMP102,BNO080,Qwiic Keypad)
Device found at address 0x50  (EEPROM,FRAM)
Device found at address 0x62  (LidarLite,MCP4725,AtlasEzoORP)
Device found at address 0x68  (DS1307,DS3231,MPU6050,MPU9050,MPU9250,ITG3200,ITG3701,LSM9DS
Device found at address 0x7C  (FRAM_ID)
done
```

Using Addresses

Many datasheets will document I^{2}C addresses as 8 bit numbers including a R/W bit. Here is an excerpt from the AT24C256B datasheet, the chip used in the example above.



AT24C256B

5. Device Addressing

The 256K EEPROM requires an 8-bit device address word following a start condition to enable the chip for a read or write operation (see Figure 5-1). The device address word consists of a mandatory "1", "0" sequence for the first four most significant bits as shown. This is common to all two-wire EEPROM devices.

Figure 5-1 Device Address

1	0	1	0	A2	A1	A0	R/W
MSB							LSB

The next three bits are the A2, A1, A0 device address bits to allow as many as eight devices on the same bus. These bits must compare to their corresponding hardwired input pins. The A2, A1, and A0 pins use an internal proprietary circuit that biases them to a logic low condition if the pins are allowed to float.

The eighth bit of the device address is the read/write operation select bit. A read operation is initiated if this bit is high, and a write operation is initiated if this bit is low.

The Wire library requires addresses which do not include the R/W bit. Based only on the datasheet, you might conclude the address is 160 when writing and 161 when reading. The Wire library needs address 80 to communicate with this chip. The R/W bit is automatically created based on your use of the send or receive functions.

More Details

Please refer to the official Wire library documentation for more details.