Initializing the Wire library

Master

```
#include <Wire.h>
```

```
...
```

```
Wire.begin ();
```

Slave

```
#include <Wire.h>
```

```
...
```

```
const byte SLAVE_ADDRESS = 0x68;  // eg. DS1307 clock
```

```
...
```

```
Wire.begin (SLAVE_ADDRESS);
```

Changing I2C speed

Master only, as that sends out the clock pulses.

After doing:

```
Wire.begin ();
```

Choose another speed using TWBR (Two Wire Bit Rate) register:

```
TWBR = 12;  // 400 kHz (maximum)
```

or:

```
TWBR = 32;  // 200 kHz
```

or:

```
TWBR = 72;  // 100 kHz (default)
```

or:

```
TWBR = 152;  // 50 kHz
```

or:

```
TWBR = 78;  // 25 kHz
TWSR |= bit (TWPS0);  // change prescaler
```

or:

```
TWBR = 158;  // 12.5 kHz
TWSR |= bit (TWPS0);  // change prescaler
```

Examples are for Atmega328P running at 16 MHz (eg. Arduino Uno, Duemilanove, etc.)

Master: send data to slave

You can send one or more bytes (up to **32**), like this:

```
  Wire.beginTransmission (SLAVE_ADDRESS);
  Wire.write (0x20);
  Wire.write (0x30);
  Wire.write (0x40);
  Wire.write (0x50);
  if (Wire.endTransmission () == 0)
     {
     // success!
     }
  else
     {
     // failure ... maybe slave wasn't ready or not connected
     }
```

Maximum of 32 bytes can be sent by the standard Wire library!

The actual writing occurs on the Wire.endTransmission function call. You should test on that line whether or not the send succeeded. If not there is no point in trying to get a response.

Slave: receive data from master

```
// set up receive handler  (in setup)
Wire.onReceive (receiveEvent);  // interrupt handler for incoming messages
```

```
...
```

```
volatile byte buf [32];
```

```
// called by interrupt service routine when incoming data arrives
void receiveEvent (int howMany)
  {
  for (byte i = 0; i < howMany; i++)
    {
    buf [i] = Wire.read ();
    }  // end of for loop
  }  // end of receiveEvent
```

Note that receiveEvent is called from an Interrupt Service Routine (ISR)!

You should **not**:

- Do serial prints
- Use "delay"
- Do anything lengthy
- Do anything that requires interrupts to be active

Master: request information from slave

```
  byte buf [10];

  if (Wire.requestFrom (SLAVE_ADDRESS, 10) == 10)  // if request for 10 bytes
satisfied
    {
    for (byte i = 0; i < 10; i++)
      buf [i] = Wire.read ();
    }
  else
    {
    // failure ... maybe slave wasn't ready or not connected
    }
```

Important! You do not need to test for Wire.available here. The Wire.requestFrom function **does not return** until either the requested data is fully available, or an error occurred. Building in a wait for Wire.available simply makes it possible for the code to hang forever if the data is not available.

The correct method is to test whether the number of bytes you wanted was returned from Wire.requestFrom, as in the example above (10 bytes in this case).

You can use I2C_Anything (see below) to receive multiple bytes.

Slave: respond to request for information

```
// set up request handler  (in setup)
Wire.onRequest (requestEvent);  // interrupt handler for when data is wanted
```

```
...
```

```
// called by interrupt service routine when response is wanted
void requestEvent ()
  {
  Wire.write (0x42);  // send response
  }  // end of requestEvent
```

Note that requestEvent is called from an Interrupt Service Routine (ISR)!

You should **not**:

- Do serial prints
- Use "delay"
- Do anything lengthy
- Do anything that requires interrupts to be active

You can only do **one** Wire.write in a requestEvent! You do **not** do a Wire.beginTransmission or a Wire.endTransmission. There is a limit of 32 bytes that can be returned.

You can use I2C_Anything (see below) to return multiple bytes.

Example of returning multiple bytes:

```
void requestEvent ()
  {
  byte buf [4] = { 1, 2, 3, 4 };
  Wire.write (buf, sizeof buf);   // send response
  }  // end of requestEvent
```

Send/receive any data type

If you want to send or receive things other than simple bytes (eg. int, float, structures) you can use the I2C_Anything which simplifies this process.

The I2C_Anything library can be downloaded from:

http://gammon.com.au/Arduino/I2C_Anything.zip

Just unzip the downloaded file, and put the resulting folder "I2C_Anything" into your Arduino "libraries" folder. Then restart the IDE and you are ready to use it.

Example of sending with I2C_Anything:

```
#include <I2C_Anything.h>

...

  long foo = 42;
  float bar = 123.456;
  Wire.beginTransmission (SLAVE_ADDRESS);
  I2C_writeAnything (foo);
  I2C_writeAnything (bar);
  if (Wire.endTransmission () == 0)
      {
      // success!
      }
    else
      {
      // failure ... maybe slave wasn't ready or not connected
      }
```

Example of receiving with I2C_Anything:

```
#include <I2C_Anything.h>
```

```
...
```

```
volatile boolean haveData = false;
volatile long foo;
volatile float bar;
```

```
// called by interrupt service routine when incoming data arrives
void receiveEvent (int howMany)
 {
 if (howMany >= (sizeof foo) + (sizeof bar))
   {
   I2C_readAnything (foo);
   I2C_readAnything (bar);
   haveData = true;
   }  // end if have enough data
 }  // end of receiveEvent
```

Note that variables set within an ISR have to be declared volatile. Note the use of a flag to signal the main loop that we have received something.

Pull-up resistors

For reliable operation both SDA and SCL should be "pulled up" by connecting them via a resistor each to Vcc (+5V). Typically 4.7K resistors would be suitable. Note that some devices may already have such resistors installed, in which case you don't need another set.

Connections

As shown earlier in this thread, SDA on the Master is connected to SDA on the Slave, and SCL on the Master is connected to SCL on the Slave. Ground should also be connected together.

Some of the more modern Arduinos (eg, Uno R3) have dedicated header pins for SDA and SCL.