



# ICubeX, openFrameworks and M+M Workshop Instructions

[ICubeX, openFrameworks and M+M Workshop](#)

[ICubeX Components](#)

[I-CubeX-oF Integration](#)

[Project Setup](#)

[Coding for I-CubeX+openFrameworks](#)

[Make the sensor control something!](#)

[M+M Middleware](#)

[Download + Install](#)

[Check Installation](#)

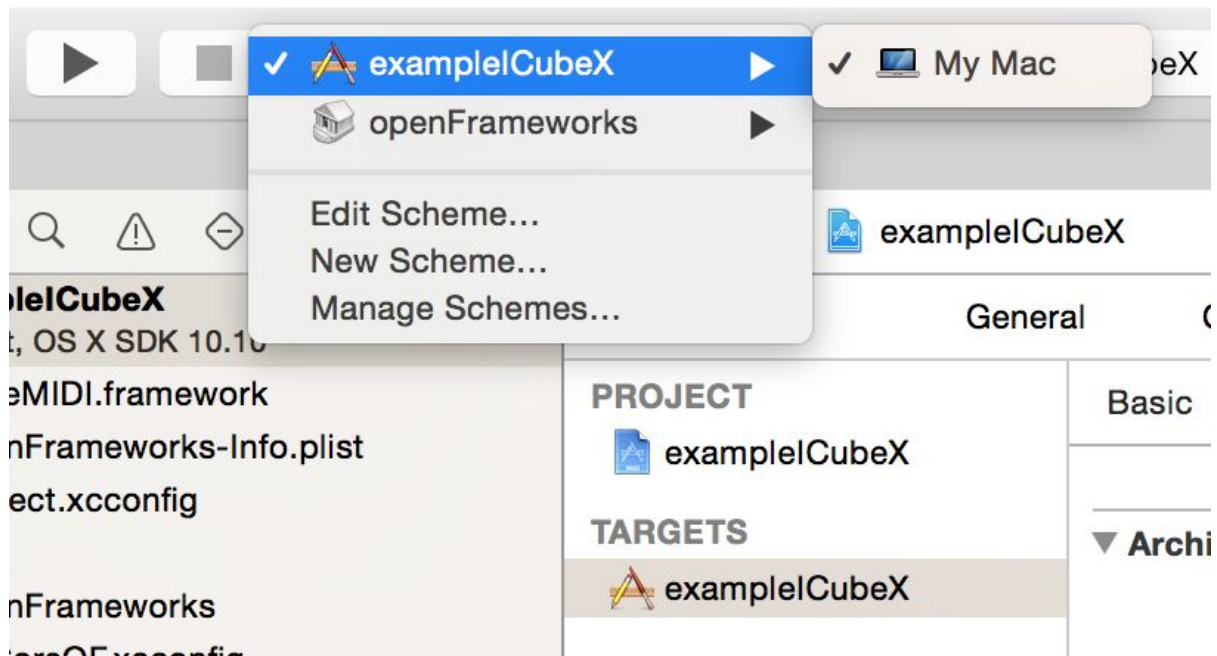
[Add Additional Dependencies to XCode Project](#)

[Yarp Bindings to oF Application](#)

[Sending Data from I-CubeX Sensor to network](#)

## ICubeX Components

- Pre-requisites: OSX 10.10 with latest version of XCode. Prior versions should work too but one might need to set the "Base SDK" setting under "Build Settings" to the correct earlier version of the OSX SDK.
- 1. Download and unzip of `_v0.8.4_osx_release_icube.zip` (see <http://icubex.com/isea>).
- 2. Go into `addons/ofxICube/exampleICubeX` folder, and open the XCode project file, and try to compile it. The first time will take a while as the oF library is big. Make sure the application is selected to compile the final executable:

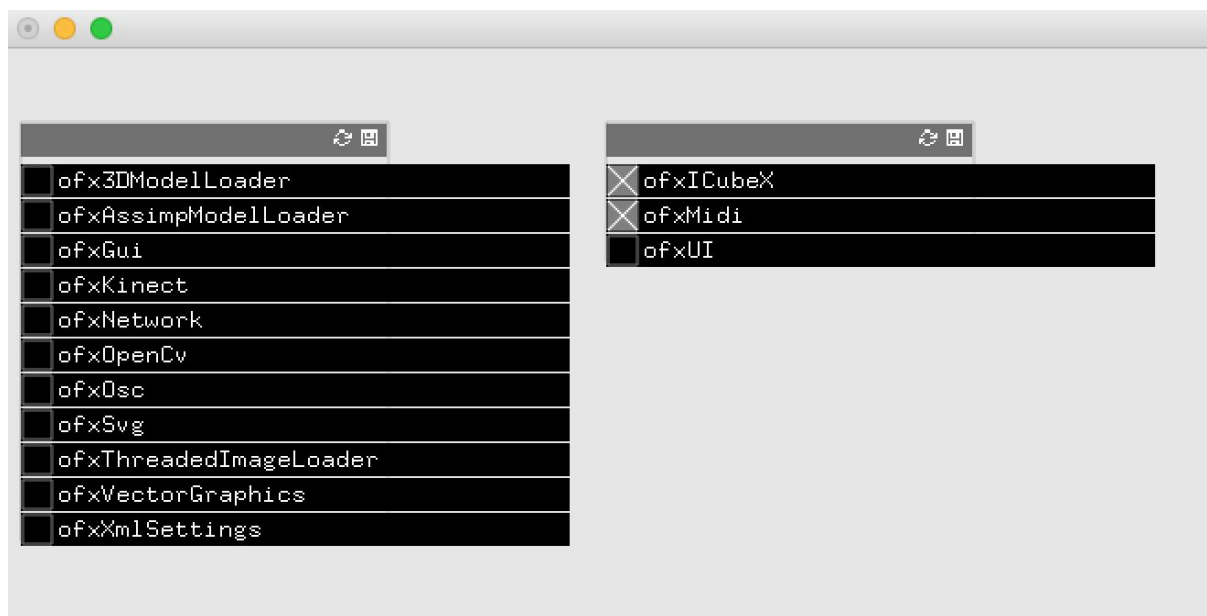


3. Run the app, and assuming I-CubeX Connect (<http://icubex.com/connect>) is running, you should be able to just press "auto connect" and start controlling the digitizer!

## I-CubeX-oF Integration

### Project Setup

- Make sure I-CubeX Connect (<http://icubex.com/connect>) is running.
- Go to openframeworks root/projectGenerator\_osx folder.
- Run the project generator executable.
- Name the project, and click on Addons. Note: if you want to change the location of the project, make sure it is in the same position relative to openframeworks root folder!



- Check ofxICubeX and ofxMidi, and go back to previous page.



Name: myICubeX\_test

<< CLICK TO CHANGE THE NAME

Path: /Users/johnty/Downloads/  
of\_v0.8.4\_osx\_release\_icube/apps/myApps

<< CLICK TO CHANGE THE DIRECTORY

Platforms: osx (xcode)

Addons: ofxICubeX, ofxMidi

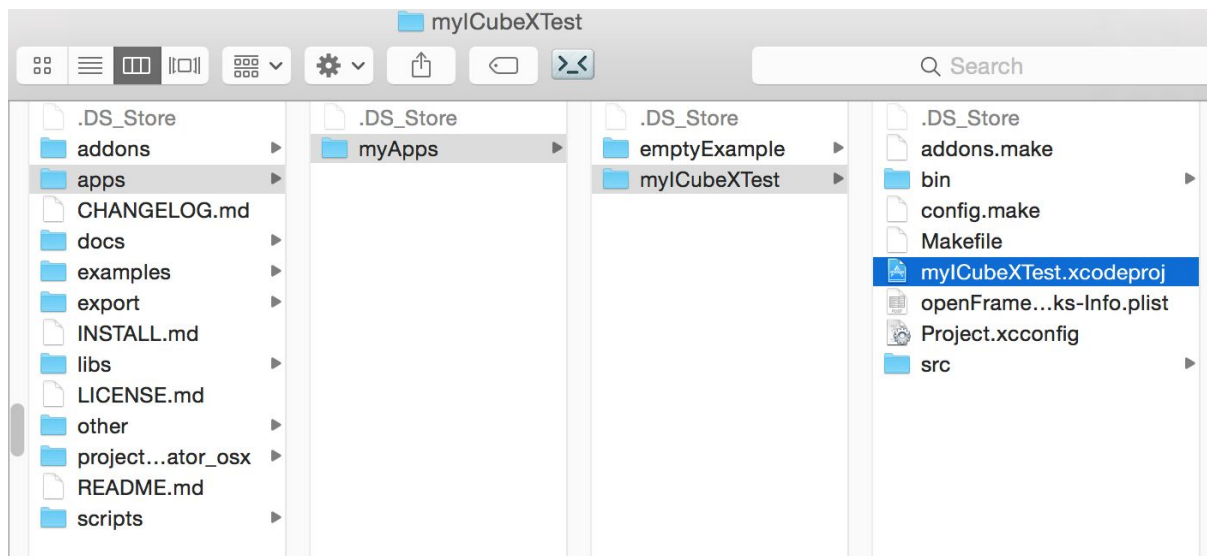
<< CLICK TO SELECT ADDONS

GENERATE PROJECT

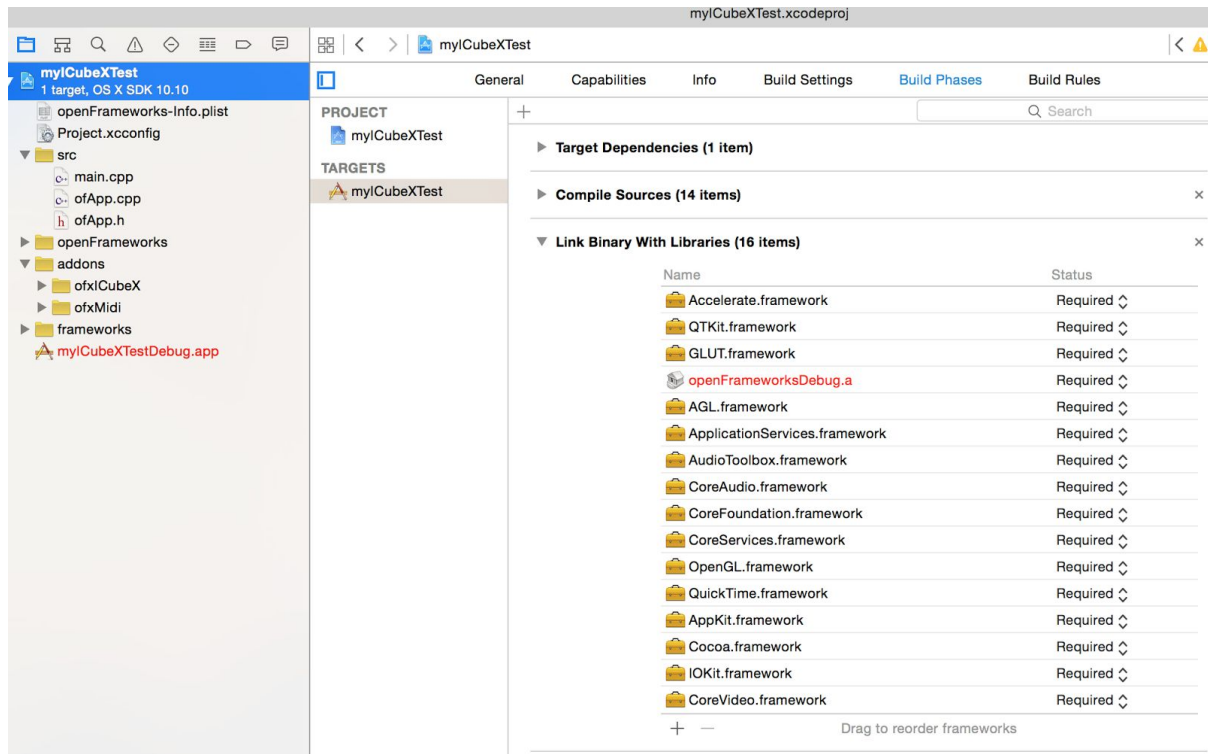
- Click generate, and go into the generated location to open the XCode Project.

## Coding for I-CubeX+openFrameworks

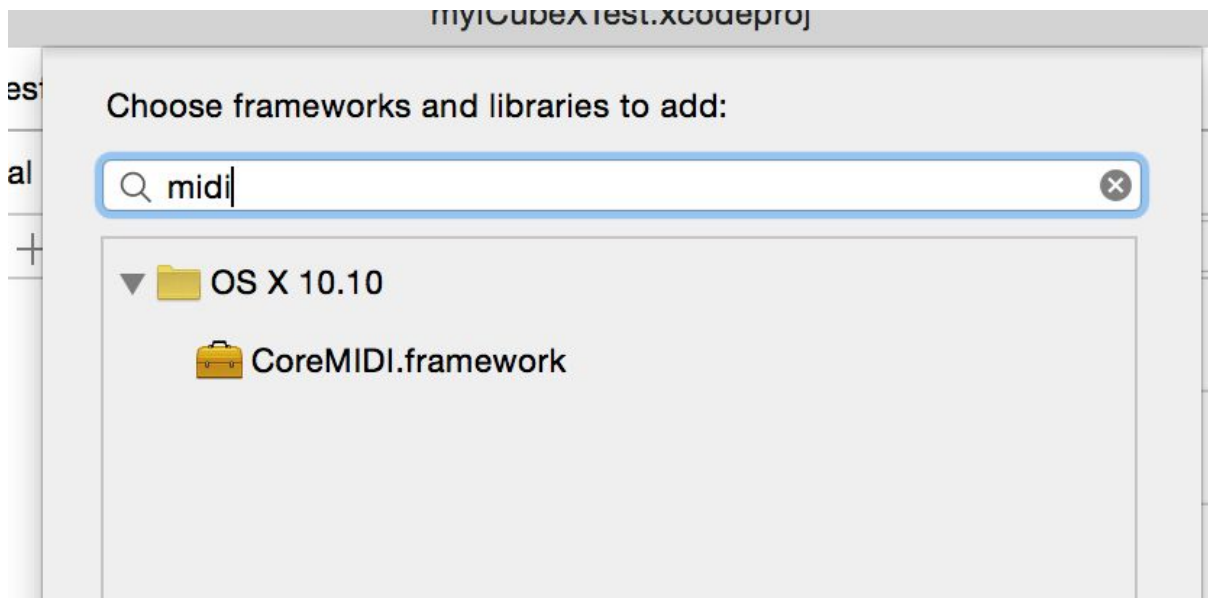
Open the newly created XCode project (default in the location below) :



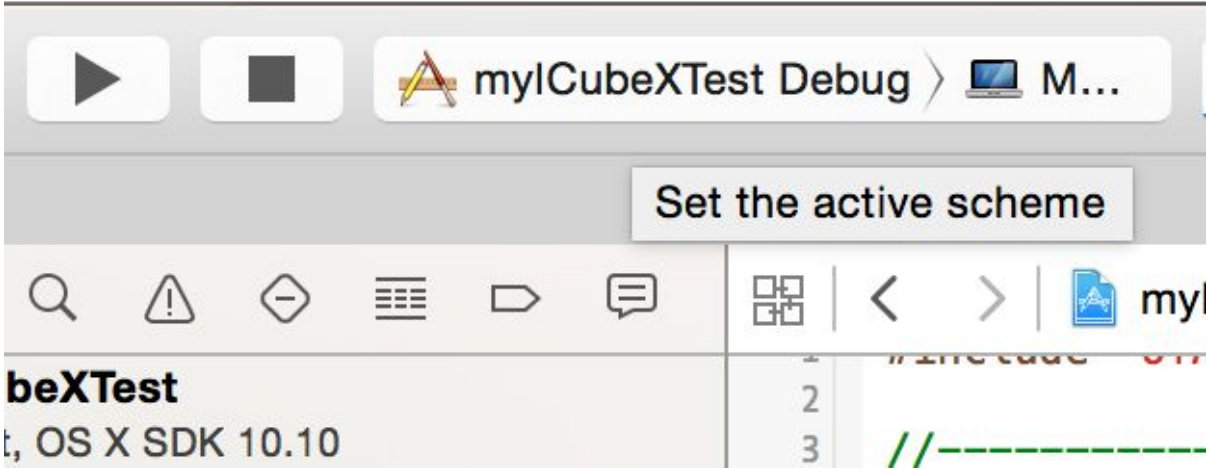
Click on the project, select "Build Phases", and press the + button at the bottom.



Add the CoreMIDI framework to the project :



Set the active scheme to Debug the target app (and not openFrameworks lib), and try building the project. It should compile without error.



Now we are ready to get to the coding.

Add to ofApp.h:

```
#include "ofxCubeX.h"
```

and create the following member variables:

```
ofxCubeX myICubeX;
```

```
ofApp.h
myICubeXTest > src > ofApp.h > class ofApp
1  #pragma once
2
3  #include "ofMain.h"
4  #include "ofxCubeX.h"
5
6  class ofApp : public ofBaseApp{
7
8      public:
9          void setup();
10         void update();
11         void draw();
12
13         void keyPressed(int key);
14         void keyReleased(int key);
15         void mouseMoved(int x, int y );
16         void mouseDragged(int x, int y, int button);
17         void mousePressed(int x, int y, int button);
18         void mouseReleased(int x, int y, int button);
19         void windowResized(int w, int h);
20         void dragEvent(ofDragInfo dragInfo);
21         void gotMessage(ofMessage msg);
22
23         ofxCubeX myICubeX;
24
25     };
26
```

In the ofApp.cpp file, do the following in the setup to check if the virtual MIDI ports are working:

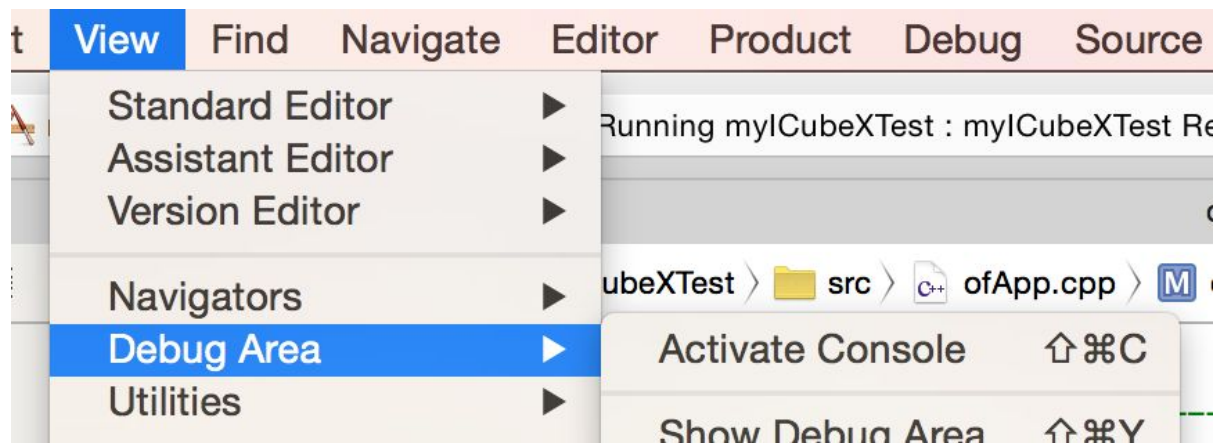
```
ofSetLogLevel(OF_LOG_VERBOSE);
ofxMidiIn::listPorts();
ofxMidiOut::listPorts();
```

```

1  #include "ofApp.h"
2
3  //-----
4  void ofApp::setup(){
5
6      ofSetLogLevel(OF_LOG_VERBOSE);
7      ofxMidiIn::listPorts();
8      ofxMidiOut::listPorts();
9  }
10
11 //-----

```

Build the app, activate the debug console and then run it.



When the app runs, you should see the I-CubeX related MIDI ports (in addition to any other ports you might have installed) listed. If not, check that ICubeX Connect (<http://icubex.com/connect>) is running.



```
[notice ] ofxMidiIn: 1 ports available
[notice ] ofxMidiIn: 0: USB-microDig 0183
[notice ] ofxMidiOut: 1 ports available
[notice ] ofxMidiOut: 0: USB-microDig 0183
```

All Output ↕



Now, we can connect to the port and start controlling the digitizer!

Add the following lines to the `setup()` function:

```
myICubeX.connectMidiOut(0);
myICubeX.connectMidiIn(0);
myICubeX.setMode(0);
myICubeX.setStream(true, 0);
```

Note that the values passed into the connect functions are the indices of the ICubeX MIDI ports. If you have existing ports in your system, they might not be 0!

```
#include "ofApp.h"

//-----
void ofApp::setup(){

    ofSetLogLevel(OF_LOG_VERBOSE);
    ofxMidiIn::listPorts();
    ofxMidiOut::listPorts();

    myICubeX.connectMidiOut(0);
    myICubeX.connectMidiIn(0);
    myICubeX.setMode(0);
    myICubeX.setStream(true, 0);
}

..
```

### Make the sensor control something!

Now we're ready to use the sensor value to do something. While this is not an of tutorial, we will go over some basic functionality of openFrameworks to get started.

The main operation of an of app is in the update() and draw() functions, which are called repeatedly. Usually, we do behind the scenes updates using the update() and any visual update with the draw().

In our simple example, we will grab the latest sensor value(s) from the I-CubeX digitizer, and then use that value to change the drawing of something on screen.

We add a member variable to ofApp.h for storing the sensor value:

```
int sensorVal;
```

```

15     void update();
16     void draw();
17
18     void keyPressed(int key);
19     void keyReleased(int key);
20     void mouseMoved(int x, int y );
21     void mouseDragged(int x, int y, int button);
22     void mousePressed(int x, int y, int button);
23     void mouseReleased(int x, int y, int button);
24     void windowResized(int w, int h);
25     void dragEvent(ofDragInfo dragInfo);
26     void gotMessage(ofMessage msg);
27
28     ofxICubeX myICubeX;
29     int sensorVal;
30

```

And then the following code to ofApp.cpp (update and draw functions):

```

//-----
void ofApp::update(){
    sensorVal = myICubeX.getSensorData(0);
}

//-----
void ofApp::draw(){
    ofBackground(0, 0, 0); //set background to black
    float x = ofGetWidth()/2;
    float y = ofGetHeight()/2;

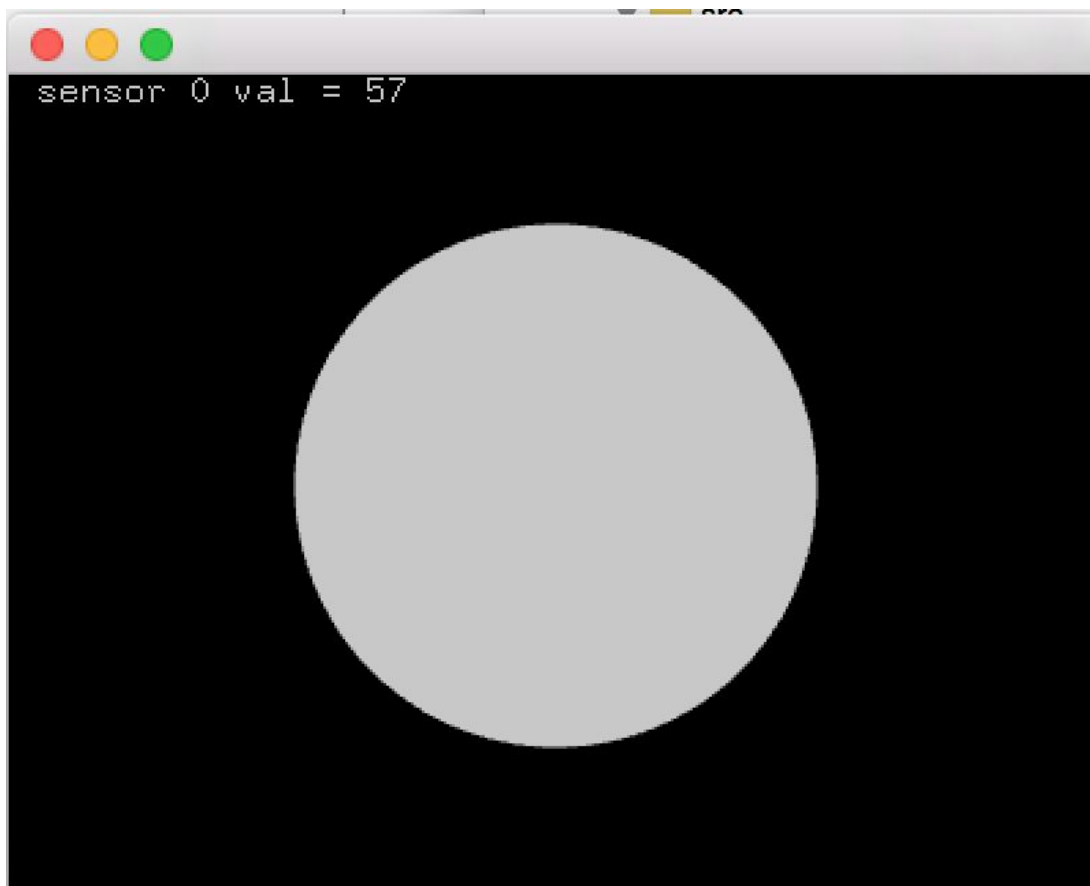
    ofSetCircleResolution(120);
    ofSetColor(200, 200, 200); //set drawing colour to grey-ish white
    ofCircle(x, y, 10+1.5*sensorVal); // draw a circle proportional to sensor value

    string some_text = "sensor 0 val = " + ofToString(sensorVal);
    ofDrawBitmapString(some_text, 10, 10);
}

```

```
myICubeXTest > src > ofApp.cpp > No Selection
29
30 //-----
31 void ofApp::update(){
32     sensorVal = myICubeX.getSensorData(0);
33
34 }
35
36
37 //-----
38 void ofApp::draw(){
39     ofBackground(0, 0, 0); //set background to black
40     float x = ofGetWidth()/2;
41     float y = ofGetHeight()/2;
42
43     ofSetCircleResolution(120);
44     ofSetColor(200, 200, 200); //set drawing colour to grey-ish white
45     ofCircle(x, y, 10+1.5*sensorVal); // draw a circle proportional to sensor value
46
47     string some_text = "sensor 0 val = " + ofToString(sensorVal);
48     ofDrawBitmapString(some_text, 10, 10);
49 }
50
51 //-----
```

If all goes well, you will see something like the following, and the radius of the circle will change as the sensor value is changed.



Now, we are ready to integrate M+M!

## M+M Middleware

### Download + Install

Download the installer package for M+M here:

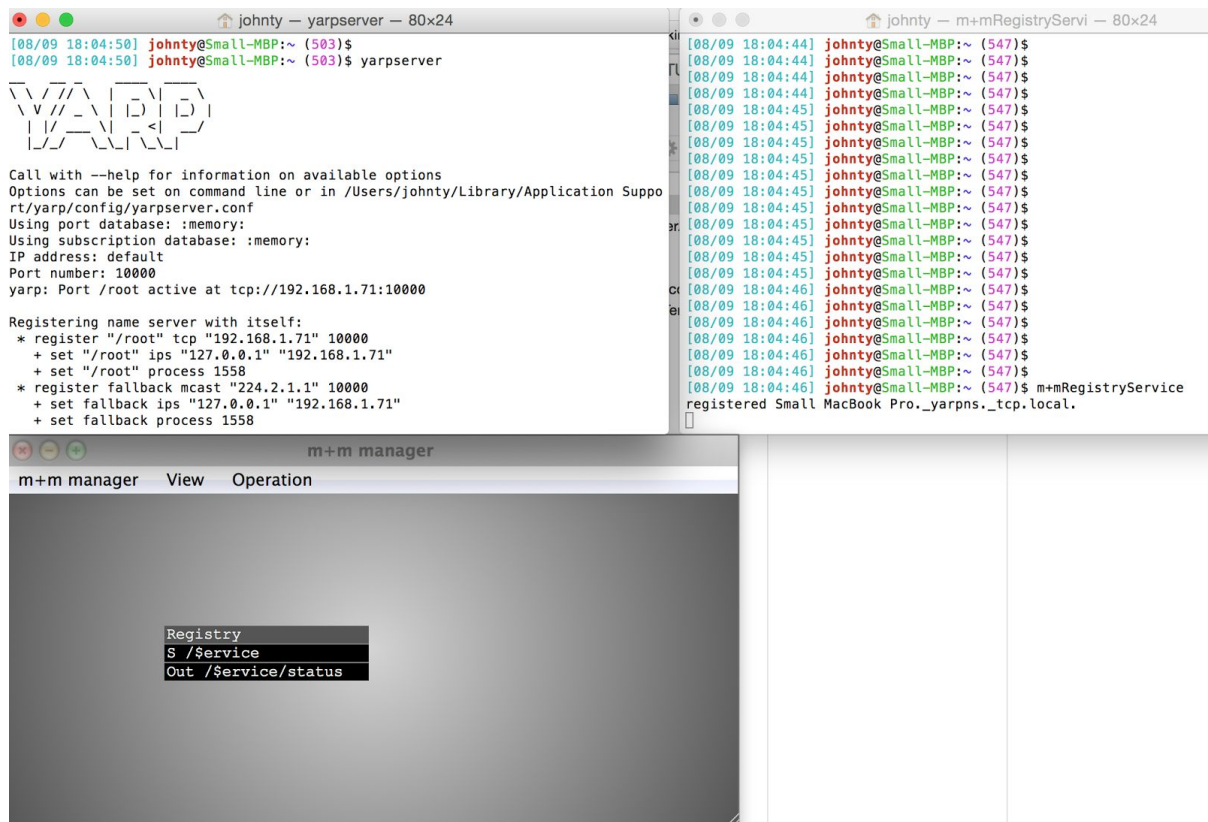
<http://www.mplusm.ca/documentation/downloads/>

Version 1.6.2 is the one we'll be using.

Open the .pkg file, and follow the instructions.

### Check Installation

Once installed, verify the install by opening up two terminal windows, and open in this order: "yarpserver", "m+mRegistryService", and the m+m manager application (see the Applications folder on your computer).



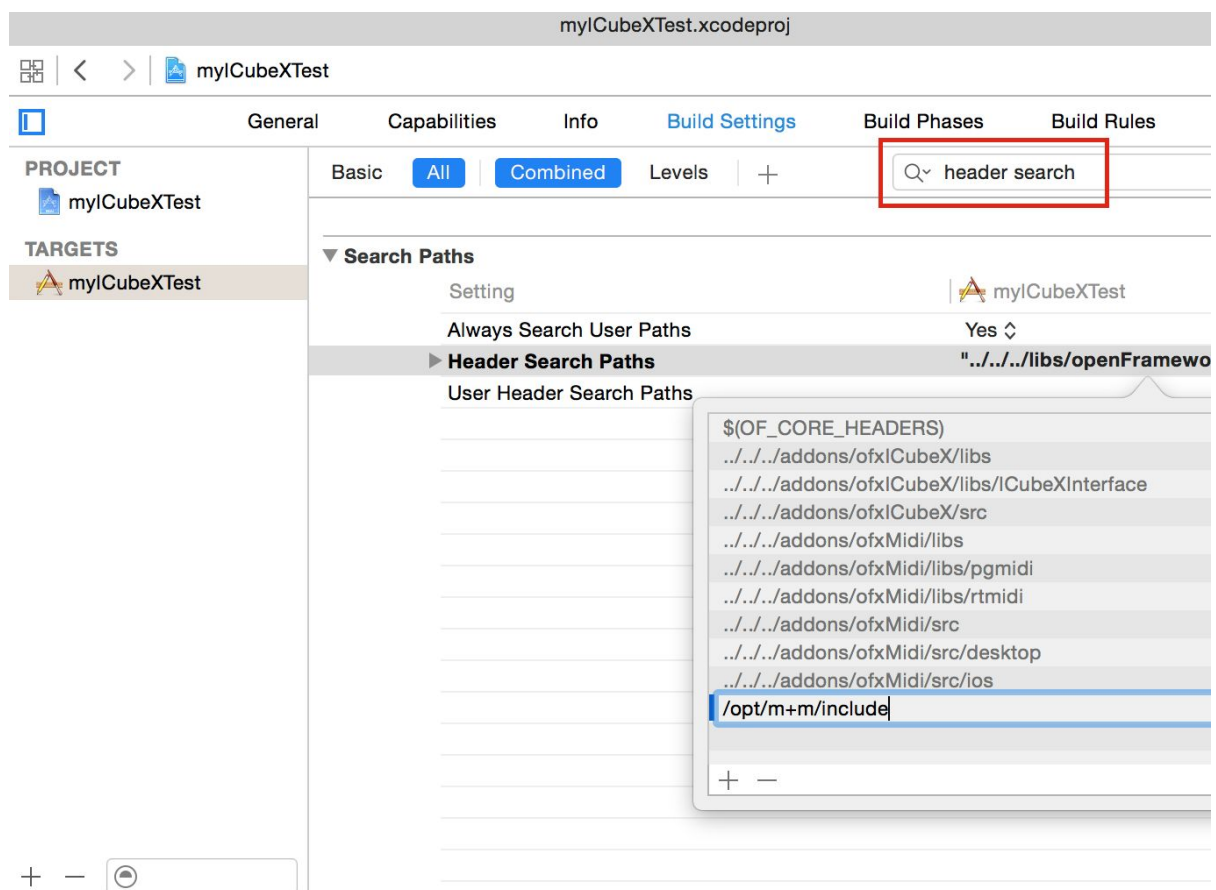
You should see the above. Keep these running in the background.

Notes:

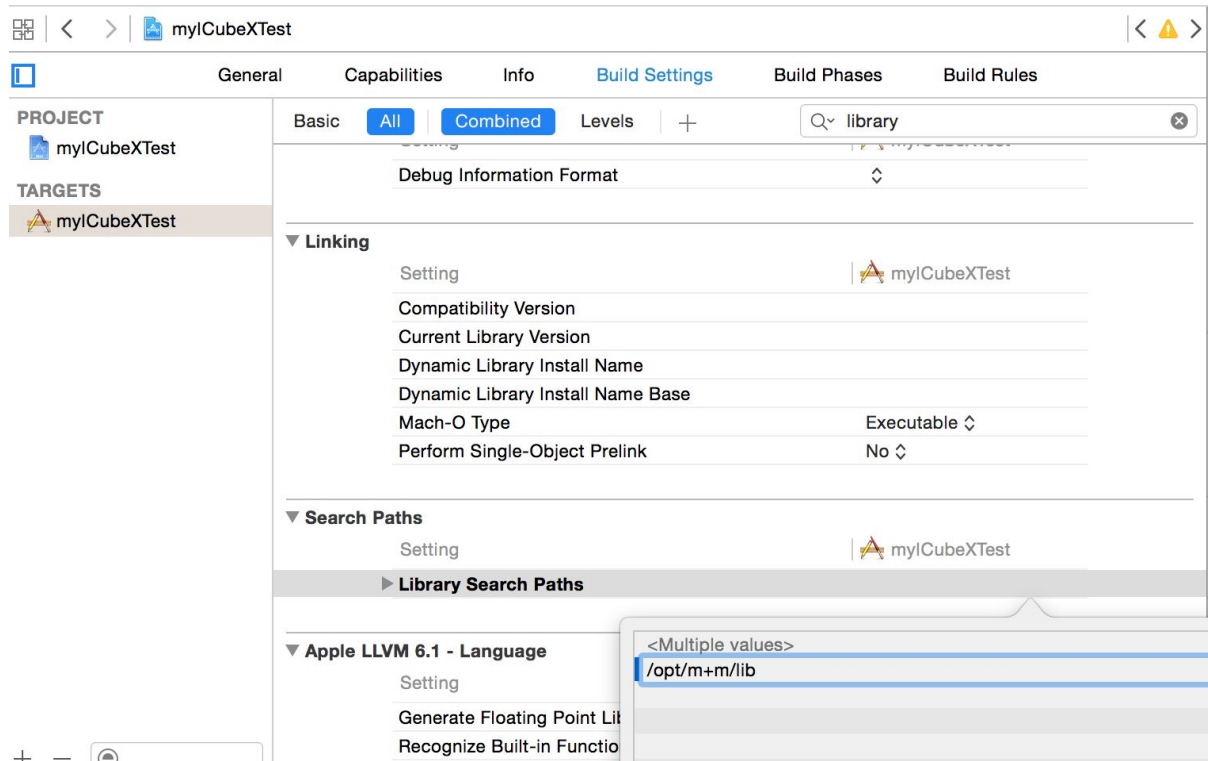
- If there was an existing yarpserver configuration that doesn't match, there will be an error, run "yarp config --clean" to override the mismatched settings, and then launch yarpserver again.
- If for some reason the path to the above binaries aren't set up, go to the /opt/m+m/bin folder ("cd /opt/m+m/bin") before running the two commands.

## Add Additional Dependencies to XCode Project

First, under the project target's "Build Settings"-> "Header Search Paths", add "/opt/m+m/include". Hint: use the search bar to find this field quickly, as shown in the red rectangle in the image below.



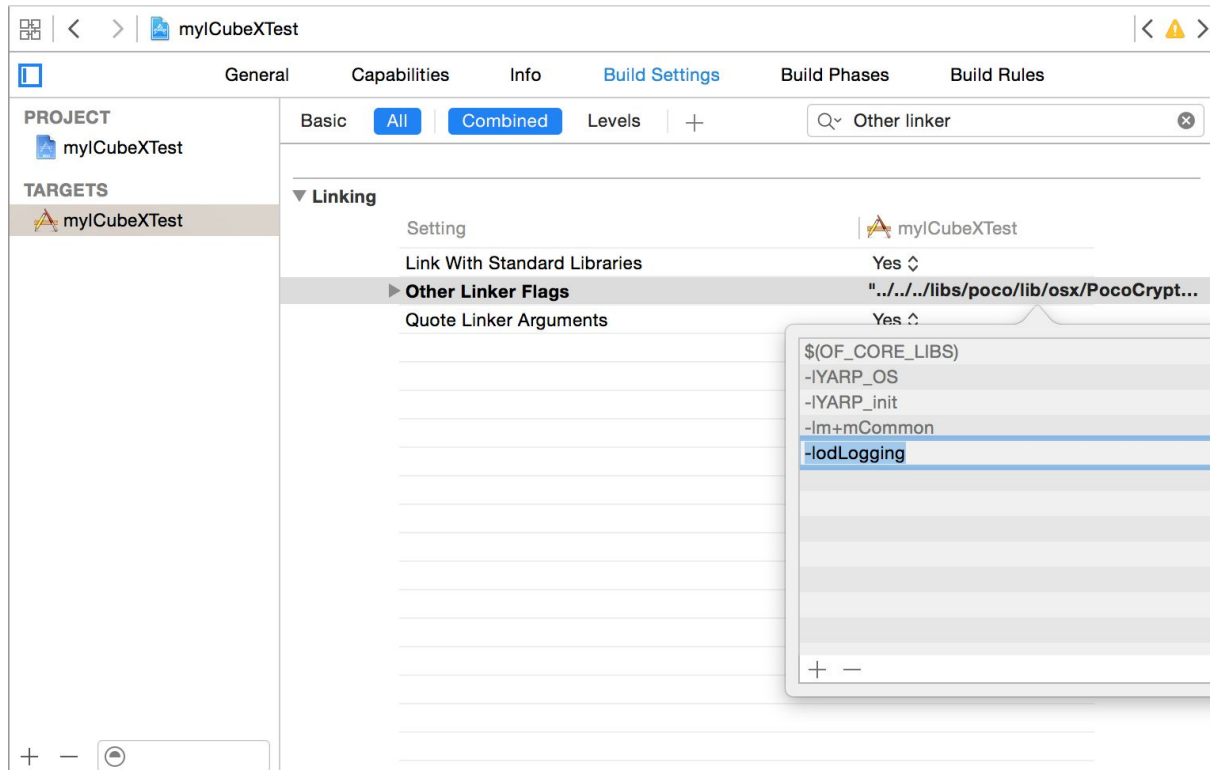
Next, add "/opt/m+m/lib" to Library Search Paths:



Add to "Other Linker Flags" (read "-l" as "minus lowercase L") :

- IYARP\_OS
- IYARP\_init
- lm+mCommon
- lodLogging

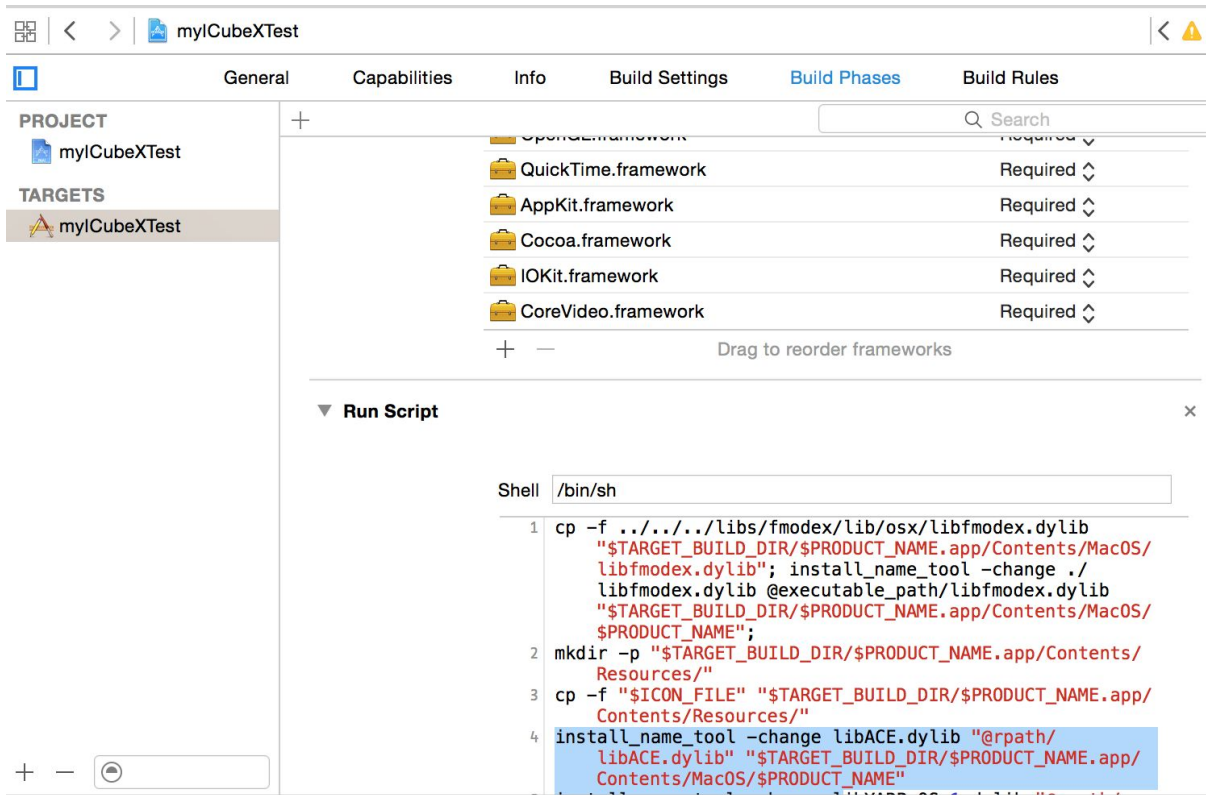




Finally, to Build Phases->Run script, append the following to the bottom:

```
install_name_tool -change libACE.dylib "@rpath/libACE.dylib"
"$TARGET_BUILD_DIR/$PRODUCT_NAME.app/Contents/MacOS/$PRODUCT_NAME"
install_name_tool -change libYARP_OS.1.dylib "@rpath/libYARP_OS.1.dylib"
"$TARGET_BUILD_DIR/$PRODUCT_NAME.app/Contents/MacOS/$PRODUCT_NAME"
install_name_tool -change libYARP_init.1.dylib "@rpath/libYARP_init.1.dylib"
"$TARGET_BUILD_DIR/$PRODUCT_NAME.app/Contents/MacOS/$PRODUCT_NAME"
install_name_tool -change libYARP_sig.1.dylib "@rpath/libYARP_sig.1.dylib"
"$TARGET_BUILD_DIR/$PRODUCT_NAME.app/Contents/MacOS/$PRODUCT_NAME"
install_name_tool -change libYARP_dev.1.dylib "@rpath/libYARP_dev.1.dylib"
"$TARGET_BUILD_DIR/$PRODUCT_NAME.app/Contents/MacOS/$PRODUCT_NAME"
install_name_tool -change libYARP_name.1.dylib "@rpath/libYARP_name.1.dylib"
"$TARGET_BUILD_DIR/$PRODUCT_NAME.app/Contents/MacOS/$PRODUCT_NAME"
install_name_tool -change libm+mCommon.dylib "@rpath/libm+mCommon.dylib"
"$TARGET_BUILD_DIR/$PRODUCT_NAME.app/Contents/MacOS/$PRODUCT_NAME"
install_name_tool -change libodLogging.dylib "@rpath/libodLogging.dylib"
"$TARGET_BUILD_DIR/$PRODUCT_NAME.app/Contents/MacOS/$PRODUCT_NAME"
```





It's a good idea to compile at this point just to make sure everything still works...

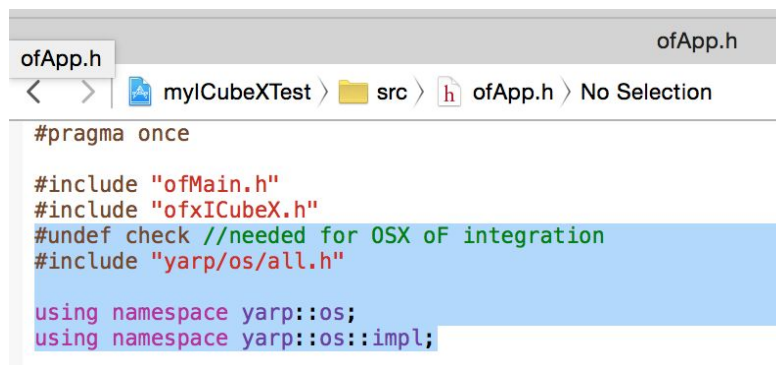
Now, we are ready to add the Yarp/M+M bindings.

## Yarp Bindings to of Application

Add the following two lines to the top of ofApp.h:

```
#undef check //needed for OSX of integration  
#include "yarp/os/all.h"
```

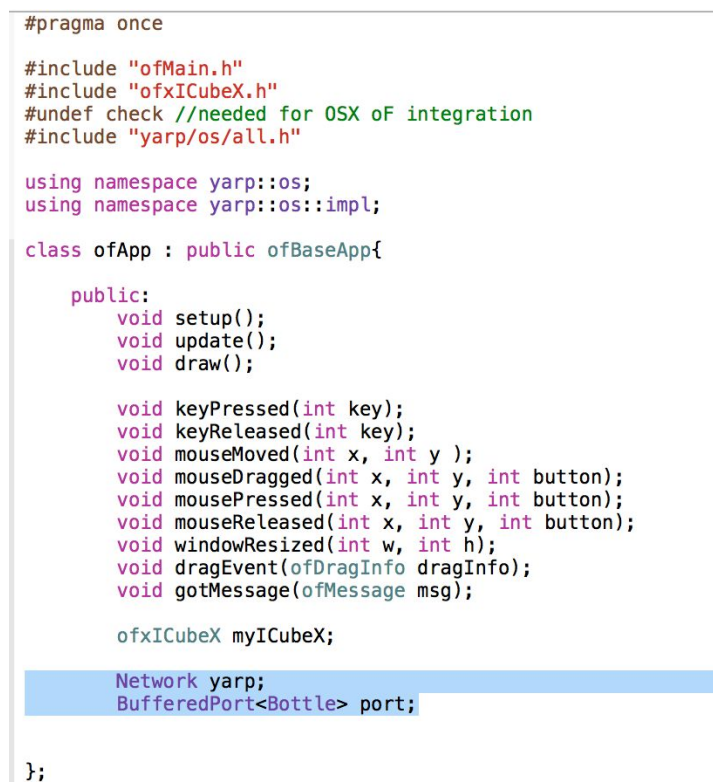
```
using namespace yarp::os;  
using namespace yarp::os::impl;
```



```
ofApp.h  
ofApp.h  
< > myICubeXTest > src > ofApp.h > No Selection  
#pragma once  
  
#include "ofMain.h"  
#include "ofxICubeX.h"  
#undef check //needed for OSX of integration  
#include "yarp/os/all.h"  
  
using namespace yarp::os;  
using namespace yarp::os::impl;
```

and the following member variables:

```
Network yarp;  
BufferedPort<Bottle> port;
```



```
#pragma once  
  
#include "ofMain.h"  
#include "ofxICubeX.h"  
#undef check //needed for OSX of integration  
#include "yarp/os/all.h"  
  
using namespace yarp::os;  
using namespace yarp::os::impl;  
  
class ofApp : public ofBaseApp{  
public:  
    void setup();  
    void update();  
    void draw();  
  
    void keyPressed(int key);  
    void keyReleased(int key);  
    void mouseMoved(int x, int y );  
    void mouseDragged(int x, int y, int button);  
    void mousePressed(int x, int y, int button);  
    void mouseReleased(int x, int y, int button);  
    void windowResized(int w, int h);  
    void dragEvent(ofDragInfo dragInfo);  
    void gotMessage(ofMessage msg);  
  
    ofxICubeX myICubeX;  
  
    Network yarp;  
    BufferedPort<Bottle> port;  
  
};
```

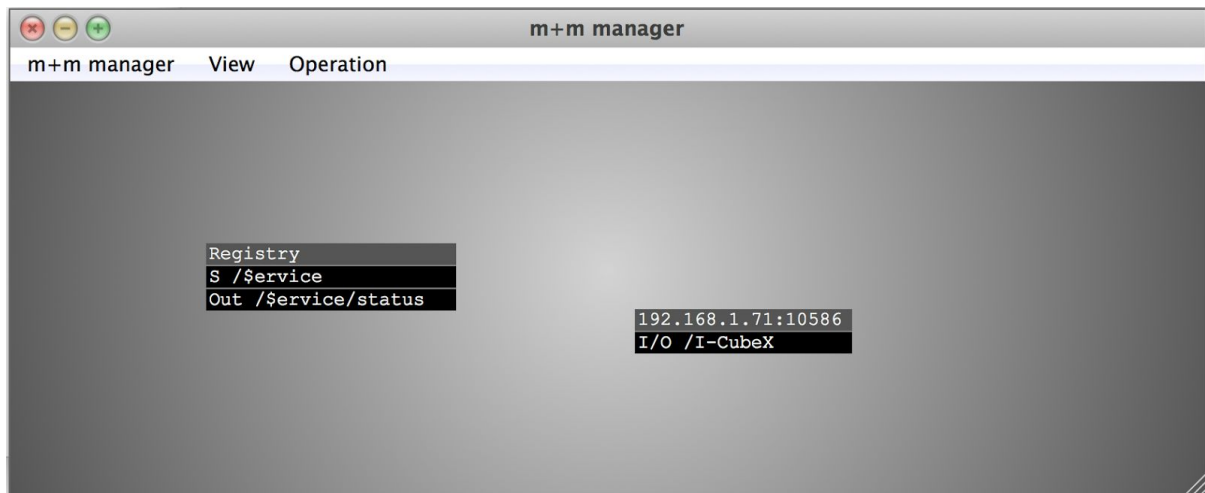
Now add the following to ofApp.cpp's setup() function:

```
yarp.init();  
port.open("/I-CubeX");
```

```
ofApp.cpp  
< > | myICubeXTest > src > ofApp.cpp > M ofApp::setup()  
#include "ofApp.h"  
  
//-----  
void ofApp::setup(){  
  
    ofSetLogLevel(OF_LOG_VERBOSE);  
    ofxMidiIn::listPorts();  
    ofxMidiOut::listPorts();  
  
    myICubeX.connectMidiOut(0);  
    myICubeX.connectMidiIn(0);  
    myICubeX.setMode(0);  
    myICubeX.setStream(true, 0);  
  
    yarp.init();  
    port.open("/I-CubeX");  
}
```

```
ofApp.cpp  
< > | myICubeXTest > src > ofApp.cpp > M ofApp::setup()  
#include "ofApp.h"  
  
//-----  
void ofApp::setup(){  
  
    ofSetLogLevel(OF_LOG_VERBOSE);  
    ofxMidiIn::listPorts();  
    ofxMidiOut::listPorts();  
  
    myICubeX.connectMidiOut(0);  
    myICubeX.connectMidiIn(0);  
    myICubeX.setMode(0);  
    myICubeX.setStream(true, 0);  
  
    yarp.init();  
    port.open("/I-CubeX");  
}
```

Compile and run the application, and you should see the following show up in the M+M manager application (make sure yarpserver and m+mRegistryService is still running!) :



Now we have a yarp port from which we can send data from the openFrameworks application and the network!

### Sending Data from I-CubeX Sensor to network

Add the following code to the update() function of ofApp.cpp:

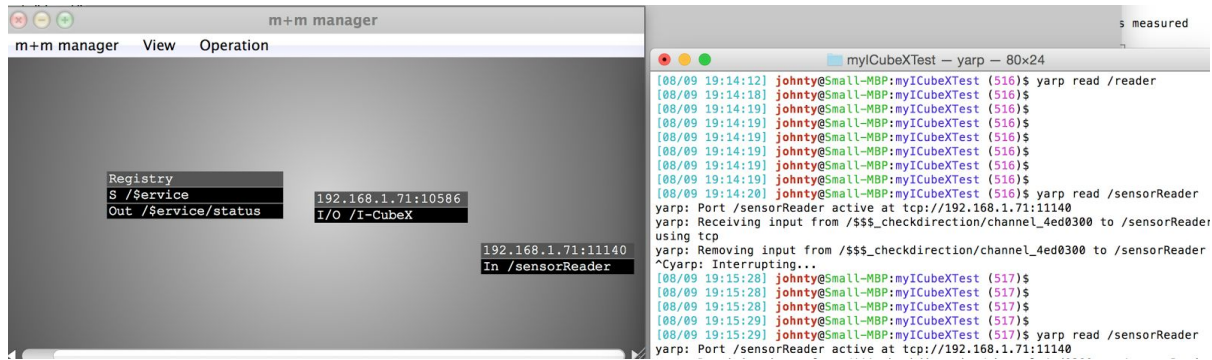
```
int currVal = myICubeX.getSensorData(0);
yarp::os::Bottle* bot;
bot = &port.prepare();
bot->clear();
bot->add(currVal);
port.write();
ofSleepMillis(5);

//-----
void ofApp::update(){
    int currVal = myICubeX.getSensorData(0);
    yarp::os::Bottle* bot;
    bot = &port.prepare();
    bot->clear();
    bot->add(currVal);
    port.write();
    ofSleepMillis(5);
}
```

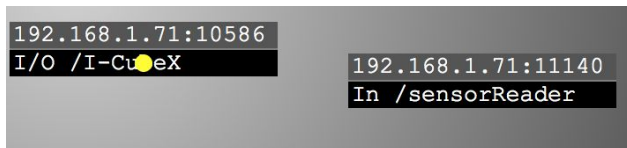
This gets the data from the first sensor, and puts it in the port.

To test this, open a terminal and start a reader using:

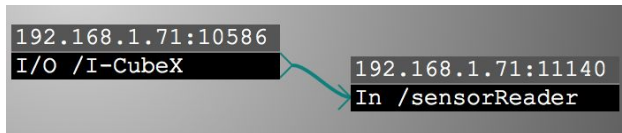
yarp read /sensorReader (once again, if it doesn't find "yarp" cd into opt/m+m/bin first...)



Now we can connect the output to the "sensorReader". in the m+m manager application, alt+left click on the I-CubeX block, which should bring up a dot:



and then left click on the /sensorReader block to make the connection:



Now go to the terminal where the sensorReader was opened, and you should see values streaming in the window. This data is now available on the M+M network, and can be used by any services and devices!

To disconnect cmd+left click on the source block, and then left click on the destination block.