

# 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 (<u>http://icubex.com/connect</u>) 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 (<u>http://icubex.com/connect</u>) 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!

•	
0 B	0 S
ofx3DModelLoader	∕ofxICubeX
ofxAssimpModelLoader	🗙 ofxMidi
ofxGui	ofxUI
ofxKinect	
ofxNetwork	
ofxOpenCv	
ofxOsc	
ofxSvg	
ofxThreadedImageLoader	
ofxVectorGraphics	
ofxXmlSettings	

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

PROJECT GENERATOR	
Name: myICubeX_test	<< CLICK TO CHANGE THE NAME
<pre>Path: /Users/johnty/Downloads/ of_v0.8.4_osx_release_icube/apps/myApps</pre>	<< CLICK TO CHANGE THE DIRECTORY
Platforms: osx (xcode)	
Addons: ofxICubeX, ofxMidi	<< CLICK TO SELECT ADDONS
	GENERATE PROJECT
- Click generate, and go into the genera	ited location to open the XCode Project.

Coding for I-CubeX+openFrameworks

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

			i mylCube	eXTest			
		~	<b>*</b> < ≜ ⊂	>_<	]		Q Search
	.DS_Store addons apps CHANGELOG.md docs	•	DS_Store myApps	•	DS_Store emptyExample myICubeXTest	•	DS_Store addons.make bin config.make Makefile
	examples export INSTALL.md libs LICENSE.md other projectator_osx README.md scripts						<ul> <li>mylCubeXTest.xcodeproj</li> <li>openFrameks-Info.plist</li> <li>Project.xcconfig</li> <li>src</li> </ul>

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

			mylCube	eXTest.xcodeproj			
	멾 🛛 🔇 💛 📓 mylCubeXTes	st					< 🔺
1 target, OS X SDK 10.10	Genera	Capabilities	Info	Build Settings	Build Phases	Build Rules	
<pre>mylCubeXTest 1 target. OS X SDK 10.10     openFrameworks-Info.plist     ofApp.cpp     ofApp.n     openFrameworks     addons     ofXICubeX     ofXICubeX     ofXIMidi     frameworks     mylCubeXTestDebug.app</pre>	Image: Contract of the second sec	Capabilities Target Depende Compile Source Link Binary Wit	Info encies (1 item es (14 items) h Libraries (1 Name Accelera a QTKit.fr GUT.fr Applicat Applicat Applicat CoreFou CoreFou CoreFou QuickTir AppKit.fr Cocoa.fr	Build Settings Build Settings  16 items) 16 items	Build Phases	Build Rules Q. Search Status Required \$ Required \$	×
			iOKit.fra	Required 🗘			
			+ -	Drag	to reorder framewor	ks	

Add the CoreMIDI framework to the project :

	myrcubex rest.xcodeproj		
Choose f	rameworks and libraries to add:		
Q midi		8	
<b>v 📒</b> OS	S X 10.10		
-	CoreMIDI.framework		

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 of App.h:

#include "ofxICubeX.h"

and create the following member variables:

ofxICubeX myICubeX;

	ofApp.h
器	$\langle \rangle \mid h$ mylCubeXTest $\rangle$ is src $\rangle$ h ofApp.h $\rangle$ C class ofApp
1	#pragma once
2	
3	#include "ofMain.h"
4	#include "otxicubex.n"
5	class of App , public of PaceApp(
7	class or App : public or baseAppl
8	public:
9	void setup():
10	<pre>void update();</pre>
11	<pre>void draw();</pre>
12	
13	<pre>void keyPressed(int key);</pre>
14	<pre>void keyReleased(int key);</pre>
15	<pre>void mouseMoved(int x, int y );</pre>
16	<pre>void mouseDragged(int x, int y, int button);</pre>
17	void mousePressed(int x, int y, int button);
18	<pre>void mouseReleased(int x, int y, int button);</pre>
19	void windowResized(int w, int n);
20	void actMessage(ofMessage msg);
22	voru gotnessage(ornessage msg/,
23	ofxICubeX mvICubeX:
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();
```

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

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

t	View	Find	Navigate	E	ditor	Produ	uct	Debug	g Sou	rce
<b>X</b>	Star Assi Vers	idard E stant E ion Edi	ditor ditor tor		Runni	ng mylC	ubeXTe	est : myl	CubeXTe	st Re
	Navi	igators		►	ubeX	Test $ angle$ 🚞	src $ angle$	c++ ofA	pp.cpp $ angle$	M
	Deb	ug Area	1		A	ctivate	Cons	sole	企業C	;
	Utili	ties		•	S	how D	ehua	Area	ሰ <mark></mark>	

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 (<u>http://icubex.com/connect</u>) 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!



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;

```
VOID UPULLC(),
1.1
             void draw();
16
17
            void keyPressed(int key);
18
19
             void keyReleased(int key);
             void mouseMoved(int x, int y);
20
            void mouseDragged(int x, int y, int button);
21
             void mousePressed(int x, int y, int button);
22
            void mouseReleased(int x, int y, int button);
23
             void windowResized(int w, int h);
24
             void dragEvent(ofDragInfo dragInfo);
25
             void gotMessage(ofMessage msg);
26
27
             ofxICubeX myICubeX;
28
             int sensorVal;
29
30
```

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

//-----void ofApp::update(){
 sensorVal = mylCubeX.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);

}

```
名(く
       > a mylCubeXTest > src > a ofApp.cpp > No Selection
29
30
     11-
31
     void ofApp::update(){
32
33
         sensorVal = myICubeX.getSensorData(0);
34
35
     }
36
37
     11--
     void ofApp::draw(){
38
         ofBackground(0, 0, 0); //set background to black
float x = ofGetWidth()/2;
39
40
41
         float y = ofGetHeight()/2;
42
43
          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
44
45
46
47
          string some_text = "sensor 0 val = " + ofToString(sensorVal);
48
          ofDrawBitmapString(some_text, 10, 10);
49 }
50
51
    11-
```

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.

			mylCul	peXTest.xcodeproj			
踞 🛛 < 🛛 > 🗋 myl	ICubeXTest						
	General	Capabilities	Info	Build Settings	Build Phases	Build Rules	
PROJECT	Ba	sic All C	ombined	Levels +	Q~ header	search	
TARGETS		earch Daths					
A mylCubeXTest		Setting			A m	ylCubeXTest	
		Always S	Search Use	r Paths	Yes (	;	
		► Header	Search Pa	ths	"//	/libs/openFramewo	
		User He	ader Searc	h Paths			
				\$(OF_CORE	E_HEADERS)		
		//addons/ofxlCubeX/libs					
				//addo	ns/ofxICubeX/libs/IC	CubeXInterface	
		//addons/ofxlCubeX/src					
			//addons/ofxMidi/libs/pgmidi				
				//addo	ns/ofxMidi/libs/rtmi	i	
			top				
				//addo	ns/orxivilal/src/los		
				7000/1111/1	neidde		
$+$ – $\bigcirc$							

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

⊞ < > ≧ mylCubeX	(Test			< 🔺 >
Gene	eral Capabilities Info	Build Settings	Build Phases Bu	ild Rules
PROJECT	Basic All Combined	Levels +	Q~ library	8
mylCubeXTest			1 × 2 ····	
TARGETS	Debug Information	Format	\$	
Myloubexiest	▼ Linking			
	Setting		🕂 mylCube	XTest
	Compatibility Versi	on		
	Current Library Ver	rsion		
	Dynamic Library In	stall Name		
	Dynamic Library In	stall Name Base		
	Mach-O Type		Executable	\$
	Perform Single-Ob	ject Prelink	No 🗘	
	▼ Search Paths			
	Setting		🕂 mylCube	XTest
	Library Search Pa	iths		
	▼ Apple LLVM 6.1 - Language	<multiple td="" v<=""><td>alues&gt;</td><td></td></multiple>	alues>	
	Setting	/opt/m+m/l	ib	
	Generate Floating	Point Lit		
+ - (	Recognize Built-in	Functio		

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

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

出   く >   칠 myl	ICubeXTest						< 🔺 >
	General	Capabilities	Info	Build Settings	Build Phases	Build Rules	
PROJECT	Ba	isic All	Combined	Levels +	Q~ Other lin	lker	8
TARGETS	T	inking					
À mylCubeXTest		Setting			A my	CubeXTest	
		Link Wi	th Standard	Libraries	Yes 🗘		
		► Other L	inker Flags		"//	/libs/poco/lib/osx/F	ocoCrypt
		Quote L	inker Argum	ients	Yes û		
					\$(OF_CORE_L	.IBS)	
					-IYARP_OS		
					-IYARP_init		
					-Im+mCommo	n	
					-lodLogging		
$+ - \bigcirc$							

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;



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
< > a mylCubeXTest > src > o 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
          🛕 mylCubeXTest > 🚞 src > み ofApp.cpp > 🔟 ofApp::setup()
 <
      \geq
 #include "ofApp.h"
 11--
 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 = mylCubeX.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...)

💌 🕞 💮 m+m manager	s measured
m+m manager View Operation	n n n n n n n n n n n n n n n n n n n
Registry S /Service Out /Service/status I/O /I-CubeX	192.168.1.71:11140           192.168.1.71:11140           192.168.1.71:11140

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:

192.168.1.71:10586	
I/O /I-CueX	192.168.1.71:11140
	In /sensorReader

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.