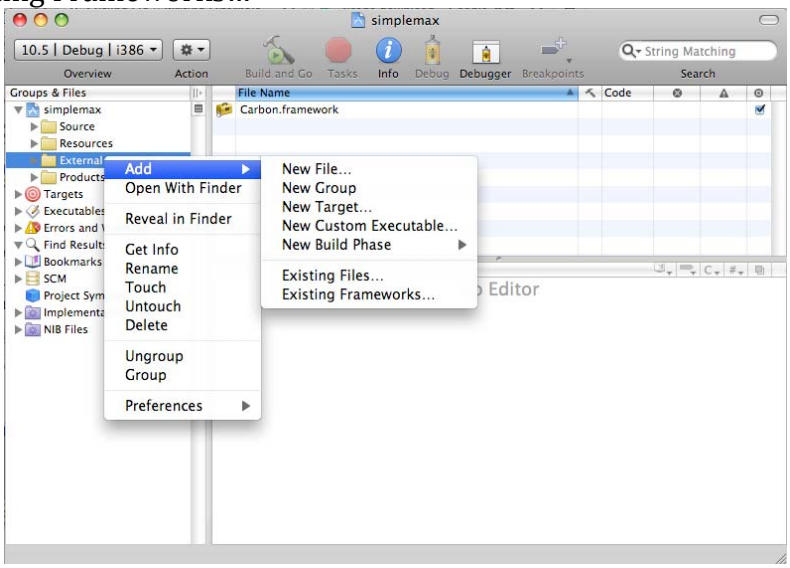
Writing Max/MSP External Tutorial: part II

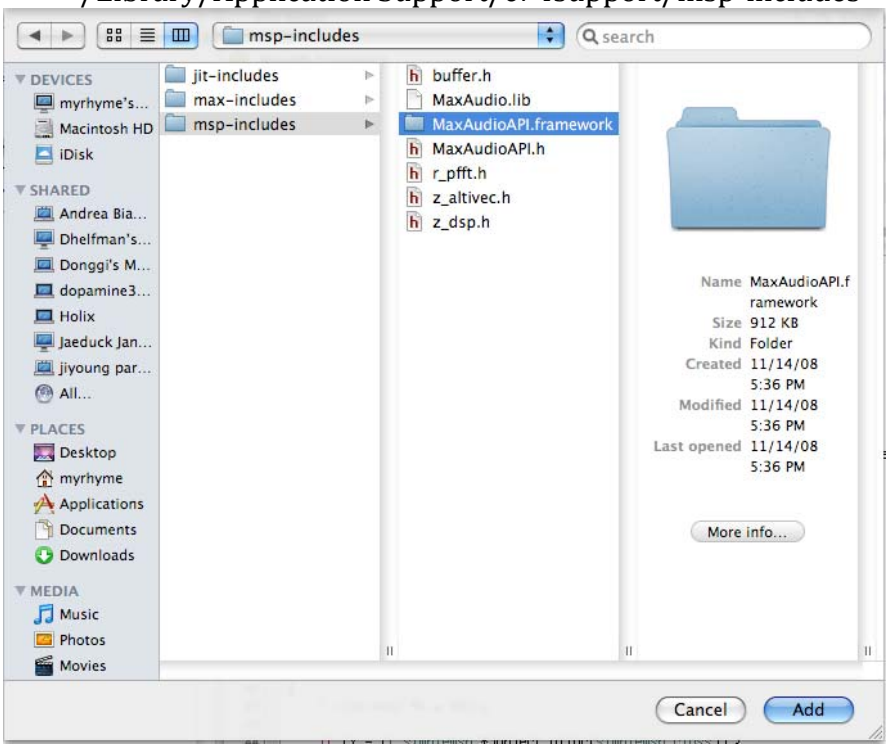
10/12/2009
GCT633 – Fall '09
Kibeom Lee | Sihwa Park | Woon Seung Yeo

On Mac OS X

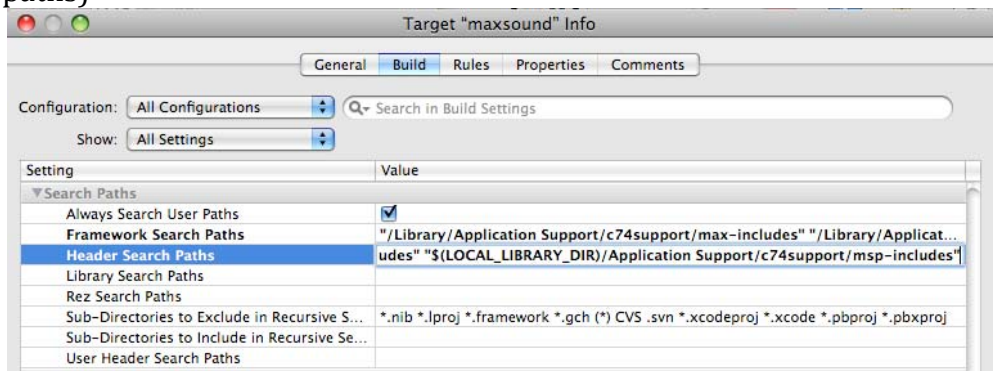
1. In order to make this project work with MSP, we have to add MSP's glue framework. Right-click on the Frameworks folder, and choose Add > Existing Frameworks…



The framework is located at
/Library/Application Support/c74support/msp-includes



2. Go to Project > Edit Active Target in the menu. In the Build tab, scroll around in the list until you find the Header Search Paths. Add it in with "/Library/Application Support/c74support/msp-includes" (it is essential that you DO use the quotes, as Xcode uses whitespace to separate multiple paths)



3. Example Source code

```
#include "ext.h" // standard Max include, always required
#include "ext_obex.h" // required for new style objects
#include "z_dsp.h"   // required for MSP objects

////////////////////////// object struct
typedef struct _simplemsp
 {
   t_pxobject  obj;  // the object itself (t_pxobject in MSP)
   float       gain;
 } t_simplemsp;

////////////////////////// function prototypes
//// standard set
void *simplemsp_new(t_symbol *s, long argc, t_atom *argv);
void simplemsp_free(t_simplemsp *x);
void simplemsp_float(t_simplemsp *x, double f);
void simplemsp_dsp(t_simplemsp *x, t_signal **sp, short *count);
t_int *simplemsp_perform(t_int *w);

////////////////////////// global class pointer variable
void *simplemsp_class;

int main(void)
{
 t_class *c;

 c = class_new("changegain~", (method)simplemsp_new,
               (method)simplemsp_free, (long)sizeof(t_simplemsp),
               0L, A_GIMME, 0);
 class_addmethod(c, (method)simplemsp_float, "float", A_FLOAT, 0);
 class_addmethod(c, (method)simplemsp_dsp, "dsp", A_CANT, 0);

 class_register(CLASS_BOX, c); // register class as a box class
```

```
 class_dspinit(c); // new style object version of dsp_initclass();
 simplemsp_class = c;

 return 0;
}

void *simplemsp_new(t_symbol *s, long argc, t_atom *argv)
{
 t_simplemsp *x = NULL;

 if (x = (t_simplemsp *)object_alloc(simplemsp_class)) {
   // MSP inlets: arg is # of inlets and is REQUIRED!
   // use 0 if you don't need inlets
   dsp_setup((t_pxobject *)x, 1);

   // signal outlet (note "signal" rather than NULL)
   outlet_new(x, "signal");

   x->gain = 1.;
 }
 return (x);
}

void simplemsp_free(t_simplemsp *x)
{
 dsp_free((t_pxobject *)x);;
}

void simplemsp_float(t_simplemsp *x, double f)
{
 x->gain = f;
}

// this function is called when the DAC is enabled, and
// “registers" a function for the signal chain.
// in this case, "simplemsp_perform"

// if msp object has 2 inlets and 3 outlets,
//    sp[0] // left input
//    sp[1] // right input
//    sp[2] // left output
//    sp[3] // middle output
//    sp[4] // right output
void simplemsp_dsp(t_simplemsp *x, t_signal **sp, short *count)
{
 post("my sample rate is: %f", sp[0]->s_sr);

 dsp_add(simplemsp_perform, 4, x, sp[0]->s_vec, sp[1]->s_vec,
                                                  sp[0]->s_n);
}
```

```
t_int *simplemsp_perform(t_int *w)
{
 // DO NOT CALL post IN HERE
 // args are in a vector, sized as specified
 // in simplemsp_dsp method
 // w[0] contains &simplemsp_perform, so we start at w[1]
 t_simplemsp *x = (t_simplemsp *)w[1];
 t_float *in = (t_float *)w[2];
 t_float *out = (t_float *)w[3];
 int n = (int)w[4];
 t_float gain = x->gain;
 t_float val;

 if (!x->obj.z_disabled) {   // check for object being disabled
        while (n--) {
         val = *in++;
         *out++ = val * gain;
   }

 }

 // you have to return the NEXT pointer in the array
 // OR MAX WILL CRASH
 return w + 5;
}
```