Realtime Audio Programming 

CODING GUIDELINES IN C

Name Schemes

Find a good name scheme for source files:

Start with an abbreviation for the whole project, for example “rtap” for Realtime Audio Programming, followed by a (short) description of what the source file does; if it’s a low pass filter class, name header and source files rtap_filter_lowpass.c and rtap_filter_lowpass.h or rtap_iir_lowpass.c and rtap_iir_lowpass.h

Decide, where (and if) you want to use capitals in source file names and stick to.

Find a good name scheme for structs and functions; the struct (the class) should have the same name as your source file, function names should reflect the purpose of the function.

If the function is working directly with the class, the name should contain the class name:

rtap_filter_lowpass_new() or rtap_filter_lowpass_perform(..)

If the function is of a more general kind and could be used for other classes, find other fitting name schemes and name source files and functions accordingly.

rtap_util_copyVector(float *in, float *out, int length);

Always give meaningful names to functions and variables, do not use abbreviations only you will understand.

Duplicate Code

If you have written similar code twice in two different functions, there is something wrong:

```
void rtap_doSomething(..)
{
        ..
        int n = 100;
        int i = 0;
        float in[n];
        float out[n];
        while(i < n)
        {
                out[i] = in[i];
                i++;
        }
        ..
}
```

```
void rtap_doSomethingElse(..)
{
        ..
        ..
        ..
        ..
        int n = 200;
        int i = 0;
        float in[n];
        float out[n];

        while(i < n)
        {
                out[i] = in[i];
                i++;
        }
}
```

Instead outsource the overlapping code:

```
rtap_utility_copyVector(float *in, float *out, int length)
{
        int i = 0;
        while(i < length)
        {
                out[i] = in[i];
                i++;
        }
}

void rtap_doSomething(..)
{
        ..
        int n = 100;
        float in[n];
        float out[n];
        rtap_utility_copyVector(in, out, n);
        ..
}
```

Separate your code from the stk. For example, do not write your algorithm directly into the Pure Data object structure. Instead use the introduced plugin structure in a separate source file and call the corresponding functions from the pd object.

Create header and source rtap_iir_lowpass.c and rtap_iir_lowpass.h

```
typedef struct rtap_iir_lowpass
{
        …
} rtap_iir_lowpass;



rtap_iir_lowpass *rtap_iir_lowpass_new()
{
        ..
        return *x;
}

..
void rtap_iir_lowpass_perform(..)
{
        ..
}
```

Create header and source rtap_iir_lowpass_puredata.c and rtap_iir_lowpass_puredata.h:

```
typedef struct rtap_iir_lowpass_tilde
{
	t_object  x_obj;
	t_sample f;
	t_outlet*x_out;
	rtap_iir_lowpass_*filter;  // integrate the filter into the pure data struct..

} rtap_iir_lowpass_tilde;

void *rtap_gain_tilde_new(t_floatarg f)
{
	rtap_gain_tilde *x =
	(rtap_gain_tilde*)pd_new(rtap_gain_tilde_class);

	x->x_out = outlet_new(&x->x_obj, &s_signal);
	x->filter = rtap_iir_lowpass_new(); // create your filter inside the pure data object new method..
	return (void *)x;
}
```