Here is some code to demonstrate how to parse sprite descriptor files etc.

sprite.h

```
#ifndef SPRITE_H
#define SPRITE_H

#include "../texture.h"

class Sprite
{
public:

	Sprite();
	~Sprite();
	void free();

	bool on;
	uint32_t difftexi;
	uint32_t teamtexi;
	uint32_t depthtexi;
	float offset[4]; //the pixel texture coordinates centered around a certain point, for use for vertex positions when not cropping (old)
	float crop[4];  //the texture coordinates [0..1] of the cropped sprite, used for the texture coords
	float cropoff[4];	//the pixel texture coordinates of the cropped sprite, used for vertex positions
	LoadedTex* pixels;
	std::string fullpath;
};

//#define SPRITES	4096
#define SPRITES	16096
extern Sprite g_sprite[SPRITES];

class SpriteList
{
public:
	bool on;
	bool inclines;
	bool sides;
	int32_t nsides;
	bool rotations;
	bool frames;
	int32_t nframes;
	std::string fullpath;

	uint32_t* sprites;
```

```
	void free()
	{
		delete [] sprites;
		sprites = NULL;
		on = false;
	}

	SpriteList()
	{
		sprites = NULL;
		on = false;
	}

	~SpriteList()
	{
		free();
	}
};

#define SPRITELISTS	256
extern SpriteList g_splist[SPRITELISTS];

class SpriteToLoad
{
public:
	std::string relative;
	uint32_t* spindex;
	bool loadteam;
	bool loaddepth;
};

extern std::vector<SpriteToLoad> g_spriteload;

extern int32_t g_lastLSp;

bool Load1Sprite();
void FreeSprites();
bool LoadSprite(const char* relative, uint32_t* spindex, bool loadteam, bool loaddepth);
void QueueSprite(const char* relative, uint32_t* spindex, bool loadteam, bool loaddepth);
void ParseSprite(const char* relative, Sprite* s);
bool PlayAnim(float& frame, int32_t first, int32_t last, bool loop, float rate);
bool PlayAnimB(float& frame, int32_t first, int32_t last, bool loop, float rate);    //Play animation
backwards
int32_t SpriteRef(SpriteList* sl, int32_t frame, int32_t incline, int32_t pitch, int32_t yaw, int32_t
roll);
bool LoadSpriteList(const char* relative, uint32_t* splin, bool loadteam, bool loaddepth, bool queue);

#endif
```

sprite.cpp

```
#include "sprite.h"
#include "../utils.h"
#include "../texture.h"
#include "../gui/gui.h"
#include "../debug.h"
#include "tile.h"

std::vector<SpriteToLoad> g_spriteload;
int32_t g_lastLSp = -1;
Sprite g_sprite[SPRITES];
SpriteList g_splist[SPRITELISTS];

Sprite::Sprite()
{
	on = false;
	difftexi = 0;
	teamtexi = 0;
	pixels = NULL;
}

Sprite::~Sprite()
{
   free();
}

void Sprite::free()
{
   if(pixels)
   {
      delete pixels;
      pixels = NULL;
   }

	//Free textures?

	on = false;
}

void FreeSprites()
{
	for(int32_t i=0; i<SPRITES; i++)
	{
		Sprite* s = &g_sprite[i];

		if(!s->on)
```

```
                continue;

            s->free();
        }

        for(int32_t i=0; i<SPRITELISTS; i++)
        {
            SpriteList* sl = &g_splist[i];

            if(!sl->on)
                continue;

            sl->free();
        }
}

bool Load1Sprite()
{
        if(g_lastLSp+1 < g_spriteload.size())
            Status(g_spriteload[g_lastLSp+1].relative.c_str());

        CHECKGLERROR();

        if(g_lastLSp >= 0)
        {
            SpriteToLoad* s = &g_spriteload[g_lastLSp];
            if(!LoadSprite(s->relative.c_str(), s->spindex, s->loadteam, s->loaddepth))
            {
                char m[128];
                sprintf(m, "Failed to load sprite %s", s->relative.c_str());
                ErrMess("Error", m);
            }
        }

        g_lastLSp ++;

        if(g_lastLSp >= g_spriteload.size())
        {
            return false;    // Done loading all
        }

        return true;     // Not finished loading
}

void QueueSprite(const char* relative, uint32_t* spindex, bool loadteam, bool loaddepth)
{
        SpriteToLoad stl;
        stl.relative = relative;
```

```
	stl.spindex = spindex;
	stl.loadteam = loadteam;
	stl.loaddepth = loaddepth;
	g_spriteload.push_back(stl);
}

int32_t NewSprite()
{
	for(int32_t i=0; i<SPRITES; i++)
	{
		Sprite* s = &g_sprite[i];

		if(!s->on)
			return i;
	}

	return -1;
}

int32_t NewSpriteList()
{
	for(int32_t i=0; i<SPRITES; i++)
	{
		SpriteList* sl = &g_splist[i];

		if(!sl->on)
			return i;
	}

	return -1;
}

bool FindSprite(uint32_t &spriteidx, const char* relative)
{
	char corrected[DMD_MAX_PATH+1];
	strcpy(corrected, relative);
	CorrectSlashes(corrected);
	char fullpath[DMD_MAX_PATH+1];
	FullPath(corrected, fullpath);

	for(int32_t i=0; i<SPRITES; i++)
	{
		Sprite* s = &g_sprite[i];

		if(s->on && stricmp(s->fullpath.c_str(), fullpath) == 0)
		{
			//g_texindex = i;
			//texture = t->texname;
```

```
                        spriteidx = i;
                        return true;
                }
        }

        return false;
}

bool LoadSprite(const char* relative, uint32_t* spindex, bool loadteam, bool loaddepth)
{
#ifdef ISOTOP
        loaddepth = false;
#endif

        if(FindSprite(*spindex, relative))
                return true;

        int32_t i = NewSprite();

        if(i < 0)
                return false;

        Sprite* s = &g_sprite[i];
        s->on = true;
        *spindex = i;

        char full[DMD_MAX_PATH+1];
        FullPath(relative, full);
        CorrectSlashes(full);
        s->fullpath = full;

        char reltxt[DMD_MAX_PATH+1];
        char reldiff[DMD_MAX_PATH+1];
        char relteam[DMD_MAX_PATH+1];
        char reldepth[DMD_MAX_PATH+1];
        sprintf(reltxt, "%s.txt", relative);
        sprintf(reldiff, "%s.png", relative);
        sprintf(relteam, "%s_team.png", relative);
        sprintf(reldepth, "%s_depth.png", relative);
        ParseSprite(reltxt, s);

        CreateTex(s->difftexi, reldiff, true, false);
        //if(loadteam)
                CreateTex(s->teamtexi, relteam, true, false);
#ifndef PLATFORM_MOBILE
        if(loaddepth)
                CreateTex(s->depthtexi, reldepth, true, false);
#endif
```

```
	char pixfull[DMD_MAX_PATH+1];
	FullPath(reldiff, pixfull);
	s->pixels = LoadTexture(pixfull);

	if(!s->pixels)
	{
		Log("Failed to load sprite %s\r\n", relative);
		return false;
	}
	else
		Log("%s\r\n", relative);

	return true;
}

bool LoadSpriteList(const char* relative, uint32_t* splin, bool loadteam, bool loaddepth, bool queue)
{
	//if(FindSpriteList(*splin, relative))
	//	return;

	int32_t i = NewSpriteList();

	if(i < 0)
		return false;

	SpriteList* sl = &g_splist[i];
	sl->on = true;
	*splin = i;

	char full[DMD_MAX_PATH+1];
	FullPath(relative, full);
	CorrectSlashes(full);
	sl->fullpath = full;

	char txtpath[DMD_MAX_PATH+1];
	sprintf(txtpath, "%s_list.txt", full);

	int32_t nsides = 1;
	bool dorots = false;	//rotations
	bool dosides = false;	//sides
	bool doincls = false;	//inclines
	bool doframes = false;		//frames
	int32_t nframes = 1;

	FILE* fp = fopen(txtpath, "r");

	if(!fp)
```

```
		return false;

	char line[128];

	do
	{
		fgets(line, 100, fp);

		char* tok = strtok(line, " ");

		if(!tok)
			continue;

		if(strcmp(tok, "inclines") == 0)
		{
			tok = strtok(NULL, " ");

			if(!tok)
				continue;

			doincls = (bool)StrToInt(tok);
		}
		else if(strcmp(tok, "frames") == 0)
		{
			tok = strtok(NULL, " ");

			if(!tok)
				continue;

			doframes = (bool)StrToInt(tok);

			tok = strtok(NULL, " ");

			if(!doframes || !tok)
				continue;

			nframes = StrToInt(tok);
		}
		else if(strcmp(tok, "sides") == 0)
		{
			tok = strtok(NULL, " ");

			if(!tok)
				continue;

			dosides = (bool)StrToInt(tok);

			tok = strtok(NULL, " ");
```

```
                if(!dosides || !tok)
                        continue;

                nsides = StrToInt(tok);
        }
        else if(strcmp(tok, "rotations") == 0)
        {
                tok = strtok(NULL, " ");

                if(!tok)
                        continue;

                dorots = (bool)StrToInt(tok);

                tok = strtok(NULL, " ");

                if(!dorots || !tok)
                        continue;

                nsides = StrToInt(tok);
        }
}while(!feof(fp));

fclose(fp);

int32_t ncombos = 1;
int32_t sidechunk = 1;
int32_t inclchunk = 1;
int32_t frameschunk = 1;

if(dorots)
{
        ncombos *= nsides;
        ncombos *= nsides;
        ncombos *= nsides;
        sidechunk = nsides * nsides * nsides;
}
else if(dosides)
{
        ncombos *= nsides;
        sidechunk = nsides;
}

inclchunk = sidechunk;

if(doincls)
{
```

```
		ncombos *= INCLINES;
		inclchunk = sidechunk * INCLINES;
	}

	frameschunk = inclchunk;

	if(doframes)
	{
		ncombos *= nframes;
		frameschunk = inclchunk * nframes;
	}

	sl->frames = doframes;
	sl->nframes = nframes;
	sl->inclines = doincls;
	sl->nsides = nsides;
	sl->sides = dosides;
	sl->rotations = dorots;
	sl->sprites = new uint32_t [ ncombos ];

	int32_t incli = 0;
	int32_t pitchi = 0;
	int32_t yawi = 0;
	int32_t rolli = 0;
	int32_t sidei = 0;
	int32_t framei = 0;

	for(int32_t ci=0; ci<ncombos; ci++)
	{
		if(dorots)
		{
			yawi = ci % nsides;
			pitchi = (ci / nsides) % nsides;
			rolli = (ci / nsides / nsides) % nsides;
		}
		else if(dosides)
		{
			sidei = ci % nsides;
		}

		if(doincls)
		{
			incli = (ci / sidechunk) % INCLINES;
		}

		if(doframes)
		{
			framei = (ci / inclchunk) % nframes;
```

```
		}

		char combo[DMD_MAX_PATH+1];

		char frame[32];
		char side[32];
		strcpy(frame, "");
		strcpy(side, "");

		if(doframes)
			sprintf(frame, "_fr%03d", framei);

		if(dosides && !dorots)
			sprintf(side, "_si%d", sidei);
		else if(dorots)
			sprintf(side, "_y%dp%dr%d", yawi, pitchi, rolli);

		//std::string incline = "";
		char incline[32] = "";

		//TODO use INCLINENAME
		if(doincls)
		{
#if 0
			if(incli == IN_0000)  incline = "_inc0000";
			else if(incli == IN_0001)	incline = "_inc0001";
			else if(incli == IN_0010)	incline = "_inc0010";
			else if(incli == IN_0011)	incline = "_inc0011";
			else if(incli == IN_0100)	incline = "_inc0100";
			else if(incli == IN_0101)	incline = "_inc0101";
			else if(incli == IN_0110)	incline = "_inc0110";
			else if(incli == IN_0111)	incline = "_inc0111";
			else if(incli == IN_1000)	incline = "_inc1000";
			else if(incli == IN_1001)	incline = "_inc1001";
			else if(incli == IN_1010)	incline = "_inc1010";
			else if(incli == IN_1011)	incline = "_inc1011";
			else if(incli == IN_1100)	incline = "_inc1100";
			else if(incli == IN_1101)	incline = "_inc1101";
			else if(incli == IN_1110)	incline = "_inc1110";
#else
			sprintf(incline, "_inc%s", INCLINENAME[incli]);
#endif
		}

		std::string stage = "";
#if 0
		if(rendstage == RENDSTAGE_TEAM)
			stage = "_team";
```

```
                else if(rendstage == RENDSTAGE_DEPTH)
                        stage = "_depth";
#endif

                sprintf(combo, "%s%s%s%s%s", relative, side, frame, incline, stage.c_str());

                if(queue)
                        QueueSprite(combo, &sl->sprites[ci], loadteam, loaddepth);
                else
                        LoadSprite(combo, &sl->sprites[ci], loadteam, loaddepth);
        }

        return true;
}

int32_t SpriteRef(SpriteList* sl, int32_t frame, int32_t incline, int32_t pitch, int32_t yaw, int32_t roll)
{
        //int32_t ncombos = 1;
        int32_t sidechunk = 1;
        int32_t inclchunk = 1;
        int32_t frameschunk = 1;

        //pitch = roll = 0;
        //roll=0;
        //pitch = 1;

        if(sl->rotations)
        {
                //ncombos *= sl->nsides;
                //ncombos *= sl->nsides;
                //ncombos *= sl->nsides;
                sidechunk = sl->nsides * sl->nsides * sl->nsides;
        }
        else if(sl->sides)
        {
                //ncombos *= sl->nsides;
                sidechunk = sl->nsides;
        }

        inclchunk = sidechunk;

        if(sl->inclines)
        {
                //ncombos *= INCLINES;
                inclchunk = sidechunk * INCLINES;
        }

        frameschunk = inclchunk;
```

```
	if(sl->frames)
	{
		//ncombos *= sl->nframes;
		frameschunk = inclchunk * sl->nframes;
	}

	int32_t ci = 0;

	if(sl->rotations)
	{
		ci += yaw + pitch * sl->nsides + roll * sl->nsides * sl->nsides;
	}
	else if(sl->sides)
	{
		ci += yaw;
	}

	if(sl->inclines)
	{
		ci += sidechunk * incline;
	}

	if(sl->frames)
	{
		ci += inclchunk * frame;
	}

	return ci;
}

//TODO size up to power of 2 for mobile etc
void ParseSprite(const char* relative, Sprite* s)
{
	char fullpath[DMD_MAX_PATH+1];
	FullPath(relative, fullpath);

	FILE* fp = fopen(fullpath, "r");
	if(!fp) return;

	float centerx;
	float centery;
	float width;
	float height;
	float clipszx, clipszy;
	float clipminx, clipminy, clipmaxx, clipmaxy;

	fscanf(fp, "%f %f", &centerx, &centery);
```

```
	fscanf(fp, "%f %f", &width, &height);
	fscanf(fp, "%f %f", &clipszx, &clipszy);
	fscanf(fp, "%f %f %f %f", &clipminx, &clipminy, &clipmaxx, &clipmaxy);

	s->offset[0] = -centerx;
	s->offset[1] = -centery;
	s->offset[2] = s->offset[0] + width;
	s->offset[3] = s->offset[1] + height;

	//s->crop[0] = clipminx / width;
	//s->crop[1] = clipminy / height;
	//s->crop[2] = clipmaxx / width;
	//s->crop[3] = clipmaxy / height;
	//s->crop[2] = (clipminx + clipszx) / width;
	//s->crop[3] = (clipminy + clipszy) / height;
	s->crop[0] = 0;
	s->crop[1] = 0;
	s->crop[2] = clipszx / width;
	s->crop[3] = clipszy / height;

	//s->cropoff[0] = clipminx - centerx;
	//s->cropoff[1] = clipminy - centery;
	//s->cropoff[2] = clipmaxx - centerx;
	//s->cropoff[3] = clipmaxy - centery;
	//s->cropoff[2] = clipminx + clipszx - centerx;
	//s->cropoff[3] = clipminy + clipszy - centery;
	s->cropoff[0] = -centerx;
	s->cropoff[1] = -centery;
	s->cropoff[2] = clipszx - centerx;
	s->cropoff[3] = clipszy - centery;

	fclose(fp);

#if 0
	char fullpath[DMD_MAX_PATH+1];

	char frame[32];
	char side[32];
	strcpy(frame, "");
	strcpy(side, "");

	if(g_rendertype == RENDER_UNIT || g_rendertype == RENDER_BUILDING)
		sprintf(frame, "_fr%03d", g_renderframe);

	if(g_rendertype == RENDER_UNIT)
		sprintf(side, "_si%d", g_rendside);

	std::string incline = "";
```

```
	if(g_rendertype == RENDER_TERRTILE || g_rendertype == RENDER_ROAD)
	{
		if(g_currincline == INC_0000)	incline = "_inc0000";
		else if(g_currincline == INC_0001)	incline = "_inc0001";
		else if(g_currincline == INC_0010)	incline = "_inc0010";
		else if(g_currincline == INC_0011)	incline = "_inc0011";
		else if(g_currincline == INC_0100)	incline = "_inc0100";
		else if(g_currincline == INC_0101)	incline = "_inc0101";
		else if(g_currincline == INC_0110)	incline = "_inc0110";
		else if(g_currincline == INC_0111)	incline = "_inc0111";
		else if(g_currincline == INC_1000)	incline = "_inc1000";
		else if(g_currincline == INC_1001)	incline = "_inc1001";
		else if(g_currincline == INC_1010)	incline = "_inc1010";
		else if(g_currincline == INC_1011)	incline = "_inc1011";
		else if(g_currincline == INC_1100)	incline = "_inc1100";
		else if(g_currincline == INC_1101)	incline = "_inc1101";
		else if(g_currincline == INC_1110)	incline = "_inc1110";
	}

	std::string stage = "";

	if(rendstage == RENDSTAGE_TEAM)
		stage = "_team";

	sprintf(fullpath, "%s%s%s%s%s.png", g_renderbasename, side, frame, incline.c_str(),
stage.c_str());
	SavePNG(fullpath, &finalsprite);
	//sprite.channels = 3;
	//sprintf(fullpath, "%s_si%d_fr%03d-rgb.png", g_renderbasename, g_rendside,
g_renderframe);
	//SavePNG(fullpath, &sprite);

	sprintf(fullpath, "%s%s%s%s.txt", g_renderbasename, side, frame, incline.c_str());
	std::ofstream ofs(fullpath, std::ios_base::out);
	ofs<<finalcenter.x<<" "<<finalcenter.y);
	ofs<<finalimagew<<" "<<finalimageh);
	ofs<<finalclipsz.x<<" "<<finalclipsz.y);
	ofs<<finalclipmin.x<<" "<<finalclipmin.y<<" "<<finalclipmax.x<<" "<<finalclipmax.y;
#endif

#if 0
	char infopath[DMD_MAX_PATH+1];
	strcpy(infopath, texpath);
	StripExt(infopath);
	strcat(infopath, ".txt");

	std::ifstream infos(infopath);
```

```
        if(!infos)
                return;

        int32_t centeroff[2];
        int32_t imagesz[2];
        int32_t clipsz[2];

        infos>>centeroff[0]>>centeroff[1];
        infos>>imagesz[0]>>imagesz[1];
        infos>>clipsz[0]>>clipsz[1];

        t->sprite.offset[0] = -centeroff[0];
        t->sprite.offset[1] = -centeroff[1];
        t->sprite.offset[2] = t->sprite.offset[0] + imagesz[0];
        t->sprite.offset[3] = t->sprite.offset[1] + imagesz[1];
#endif
}

bool PlayAnim(float& frame, int32_t first, int32_t last, bool loop, float rate)
{
   if(frame < first || frame > last+1)
   {
      frame = first;
      return false;
   }

   frame += rate;

   if(frame > last)
   {
      if(loop)
         frame = first;
                else
                        frame = last;

      return true;
   }

   return false;
}

//Play animation backwards
bool PlayAnimB(float& frame, int32_t first, int32_t last, bool loop, float rate)
{
   if(frame < first-1 || frame > last)
   {
      frame = last;
```

```
        return false;
    }

    frame -= rate;

    if(frame < first)
    {
        if(loop)
            frame = last;
                else
                        frame = first;

        return true;
    }

    return false;
}
```

utils.h

```
#ifndef UTILS_H
#define UTILS_H

#include "platform.h"

#define CORRECT_SLASH '/'

#define ARRSZ(a)      (sizeof(a)/sizeof(a[0]))

extern std::ofstream g_applog;

const std::string DateTime();
const std::string DateTime(uint64_t milli);
const std::string Time();
const std::string FileDateTime();
void FullPath(const char* filename, char* full);
void FullWritePath(const char* filename, char* full);
std::string MakeRelative(const char* full);
void ExePath(char* exepath);
std::string StripFile(std::string filepath);
void StripPathExt(const char* n, char* o);
void StripExt(char* filepath);
void StripPath(char* filepath);
void OpenLog(const char* filename, uint64_t version);
void Log(const char* format, ...);
float StrToFloat(const char *s);
int32_t HexToInt(const char* s);
int32_t StrToInt(const char *s);
```

```
void CorrectSlashes(char* corrected);
void BackSlashes(char* corrected);
void ErrMess(const char* title, const char* message);
void InfoMess(const char* title, const char* message);
void WarnMess(const char* title, const char* message);
void OutOfMem(const char* file, int32_t line);
uint64_t Add64(uint64_t a, uint64_t b);

#define OUTOFMEM()              OutOfMem(__FILE__,__LINE__)

inline float fmax(const float a, const float b)
{
        return (((a)>(b))?(a):(b));
}

inline float fmin(const float a, const float b)
{
        return (((a)<(b))?(a):(b));
}

inline int32_t imax(const int32_t x, const int32_t y)
{
#if 0
        return (((x)>(y))?(x):(y));
#else   //probably wrong on mac
        // https://graphics.stanford.edu/~seander/bithacks.html#IntegerMinOrMax
        //int32_t x;  // we want to find the minimum of x and y
        //int32_t y;
        //int32_t r;  // the result goes here

        return x ^ ((x ^ y) & -(x < y)); // max(x, y)
#endif
}

inline int32_t imin(const int32_t x, const int32_t y)
{
#if 0
        return (((x)<(y))?(x):(y));
#else   //probably wrong on mac
        // https://graphics.stanford.edu/~seander/bithacks.html#IntegerMinOrMax
        //int32_t x;  // we want to find the minimum of x and y
        //int32_t y;
        //int32_t r;  // the result goes here

        return y ^ ((x ^ y) & -(x < y)); // min(x, y)
#endif
}
```

```
#define enmax(a,b) (((a)>(b))?(a):(b))
#define enmin(a,b) (((a)<(b))?(a):(b))

inline int32_t iabs(int32_t v)
{
#if 1	//gives wrong result on mac
	//return x & INT_MAX;
	//return x & 0x7FFFFFFF;
	//return v < 0 ? -v : v;

	// https://graphics.stanford.edu/~seander/bithacks.html
	//int32_t v;		// we want to find the absolute value of v
	//uint32_t r;  // the result goes here
	int32_t const mask = v >> ( (sizeof(int32_t) * CHAR_BIT) - 1 );
	return (v + mask) ^ mask;
#else
	return (v < 0) ? -v : v;
#endif
}

inline int32_t isign(int32_t x)
{
	//return x < 0 ? -1 : 1;

	//https://graphics.stanford.edu/~seander/bithacks.html#CopyIntegerSign

	return +1 | (x >> (sizeof(int32_t) * CHAR_BIT - 1));
}

inline float fsign(float x)
{
	return x / fabs(x);
}

//deterministic ceil
inline int32_t iceil(const int32_t num, const int32_t denom)
{
	if(denom  == 0)
		return 0;

	int32_t div = num / denom;
	const int32_t mul = div * denom;
	const int32_t rem = num - mul;

	if(rem > 0)
		div += 1;

	return div;
```

```
}

//https://github.com/preshing/CompareIntegerMaps/blob/master/util.h

//typedef unsigned int uint32_t;
//typedef unsigned long long uint64_t;

inline uint32_t upper_power_of_two(uint32_t v)
{
   v--;
   v |= v >> 1;
   v |= v >> 2;
   v |= v >> 4;
   v |= v >> 8;
   v |= v >> 16;
   v++;
   return v;
}

inline uint64_t upper_power_of_two(uint64_t v)
{
   v--;
   v |= v >> 1;
   v |= v >> 2;
   v |= v >> 4;
   v |= v >> 8;
   v |= v >> 16;
   v |= v >> 32;
   v++;
   return v;
}

// from code.google.com/p/smhasher/wiki/MurmurHash3
inline uint32_t integerHash(uint32_t h)
{
	h ^= h >> 16;
	h *= 0x85ebca6b;
	h ^= h >> 13;
	h *= 0xc2b2ae35;
	h ^= h >> 16;
	return h;
}

// from code.google.com/p/smhasher/wiki/MurmurHash3
inline uint64_t integerHash64(uint64_t k)
{
	k ^= k >> 33;
	k *= 0xff51afd7ed558ccd;
```

```
	k ^= k >> 33;
	k *= 0xc4ceb9fe1a85ec53;
	k ^= k >> 33;
	return k;
}

std::string iform(int32_t n);
std::string ullform(uint64_t n);
uint64_t GetTicks();

std::string LoadTextFile(char* strFile);
void ListFiles(const char* fullpath, std::list<std::string>& files);
void ListDirs(const char* fullpath, std::list<std::string>& dirs);

#endif //UTILS_H
```

utils.cpp

```
#include "utils.h"
#include "platform.h"
#include "window.h"
#include "version.h"

std::ofstream g_applog;

const std::string DateTime()
{
	time_t     now = time(0);
	struct tm  tstruct;
	char       buf[80];
	tstruct = *localtime(&now);
	// Visit http://www.cplusplus.com/reference/clibrary/ctime/strftime/
	// for more information about date/time format
	strftime(buf, sizeof(buf), "%Y-%m-%d %X", &tstruct);

	return buf;
}

const std::string DateTime(uint64_t milli)
{
	time_t     now = milli;
	struct tm  tstruct;
	char       buf[80];
	tstruct = *localtime(&now);
	// Visit http://www.cplusplus.com/reference/clibrary/ctime/strftime/
	// for more information about date/time format
	strftime(buf, sizeof(buf), "%Y/%m/%d %X", &tstruct);
```

```
	return buf;
}

const std::string Time()
{
	time_t     now = time(0);
	struct tm  tstruct;
	char       buf[80];
	tstruct = *localtime(&now);
	// Visit http://www.cplusplus.com/reference/clibrary/ctime/strftime/
	// for more information about date/time format
	strftime(buf, sizeof(buf), "%X", &tstruct);

	return buf;
}

const std::string FileDateTime()
{
	time_t     now = time(0);
	struct tm  tstruct;
	char       buf[80];
	tstruct = *localtime(&now);
	// Visit http://www.cplusplus.com/reference/clibrary/ctime/strftime/
	// for more information about date/time format
	strftime(buf, sizeof(buf), "%Y-%m-%d %X", &tstruct);

	for(int32_t i=0; i<strlen(buf); i++)
		if(buf[i] == ':')
			buf[i] = '-';

	return buf;
}


void OpenLog(const char* filename, uint64_t version)
{
	char fullpath[DMD_MAX_PATH+1];
	FullWritePath(filename, fullpath);
	g_applog.close();
	g_applog.open(fullpath, std::ios_base::out);
	Log("%s\r\n", DateTime().c_str());
	char verstr[64];
	VerStr(version, verstr);
	Log("Version %s\r\n", verstr);

}

void Log(const char* format, ...)
```

```
{
	if(!g_applog.is_open())
		return;

	//return;

	char dest[1024 * 16];
	va_list argptr;
	va_start(argptr, format);
	vsprintf(dest, format, argptr);
	va_end(argptr);
	g_applog<<dest<<std::endl;
	g_applog. flush();
#ifdef PLATFORM_IOS
	NSLog(@"%s", dest);
#endif
}

std::string MakeRelative(const char* full)
{
	char full2c[DMD_MAX_PATH+1];
	strcpy(full2c, full);
	CorrectSlashes(full2c);
	std::string full2(full2c);
	char exepath[DMD_MAX_PATH+1];
	ExePath(exepath);
	CorrectSlashes(exepath);

	//Log("exepath: "<<exepath);
	//Log("fulpath: "<<full);

	std::string::size_type pos = full2.find(exepath, 0);

	if(pos == std::string::npos)
	{
		return full2;
	}

	//Log("posposp: "<<pos);

	std::string sub = std::string( full2 ).substr(strlen(exepath), strlen(full)-strlen(exepath));

	//Log("subpath: "<<sub);

	return sub;
}

std::string StripFile(std::string filepath)
```

```
{
	int32_t lastof = filepath.find_last_of("/\\");
	if(lastof < 0)
		lastof = strlen(filepath.c_str());
	else
		lastof += 1;

	std::string stripped = filepath.substr(0, lastof);
	return stripped;
}

void StripPath(char* filepath)
{
	std::string s0(filepath);
	size_t sep = s0.find_last_of("\\/");
	std::string s1;

	if (sep != std::string::npos)
		s1 = s0.substr(sep + 1, s0.size() - sep - 1);
	else
		s1 = s0;

	strcpy(filepath, s1.c_str());
}

void StripExt(char* filepath)
{
	std::string s1(filepath);

	size_t dot = s1.find_last_of(".");
	std::string s2;

	if (dot != std::string::npos)
		s2 = s1.substr(0, dot);
	else
		s2 = s1;

	strcpy(filepath, s2.c_str());
}

void StripPathExt(const char* n, char* o)
{
	std::string s0(n);
	size_t sep = s0.find_last_of("\\/");
	std::string s1;

	if (sep != std::string::npos)
		s1 = s0.substr(sep + 1, s0.size() - sep - 1);
```

```
	else
		s1 = s0;

	size_t dot = s1.find_last_of(".");
	std::string s2;

	if (dot != std::string::npos)
		s2 = s1.substr(0, dot);
	else
		s2 = s1;

	strcpy(o, s2.c_str());
}

void ExePath(char* exepath)
{
#ifndef PLATFORM_IOS
#if 0
#ifdef PLATFORM_WIN
	//char buffer[DMD_MAX_PATH+1];
	GetModuleFileName(NULL, exepath, DMD_MAX_PATH+1);
	//std::string::size_type pos = std::string( buffer ).find_last_of( "\\/" );
	//std::string strexepath = std::string( buffer ).substr( 0, pos);
	//strcpy(exepath, strexepath.c_str());
#else
	char szTmp[32];
	//char buffer[DMD_MAX_PATH+1];
	sprintf(szTmp, "/proc/%d/exe", getpid());
	int32_t bytes = std::min((int32_t)readlink(szTmp, exepath, DMD_MAX_PATH+1),
DMD_MAX_PATH);
	if(bytes >= 0)
		exepath[bytes] = '\0';
	//std::string strexepath = StripFile(std::string(buffer));
	//strcpy(exepath, strexepath.c_str());
#endif
#else
	char *base_path = SDL_GetBasePath();
#if 0
	if (base_path) {
      data_path = SDL_strdup(base_path);
      SDL_free(base_path);
   } else {
      data_path = SDL_strdup("./");
   }
#endif
	if(base_path)
	{
		strcpy(exepath, base_path);
```

```
                SDL_free(base_path);
        }
#endif
#endif

#ifdef PLATFORM_IOS
        char *base_path = SDL_GetBasePath();
        if(base_path)
        {
                strcpy(exepath, base_path);
                SDL_free(base_path);
        }
        strcat(exepath, "testfolder/");
#endif
}

void FullWritePath(const char* filename, char* full)
{
#ifdef PLATFORM_IOS
        /*
         char exepath[DMD_MAX_PATH+1];
         GetModuleFileName(NULL, exepath, DMD_MAX_PATH);
         string path = StripFile(exepath);

         //char full[DMD_MAX_PATH+1];
         sprintf(full, "%s", path.c_str());

         char c = full[ strlen(full)-1 ];
         if(c != '\\' && c != '/')
         strcat(full, "\\");

         strcat(full, filename);*/

        //NSString *path = [NSHomeDirectory()
stringByAppendingPathComponent:@"Documents/"];
        //NSArray *paths = NSSearchPathForDirectoriesInDomains(NSLibraryDirectory,
NSUserDomainMask, YES);
        //NSString *path = [paths objectAtIndex:0];
        NSArray *paths = NSSearchPathForDirectoriesInDomains(NSDocumentDirectory,
NSUserDomainMask, YES);
        NSString *path = [paths objectAtIndex:0];
        sprintf(full, "%s/%s", [path cStringUsingEncoding:NSUTF8StringEncoding], filename);

        //NSLog(@"full write %s", full);
#else

        FullPath(filename, full);
#endif
```

```
}

void FullPath(const char* filename, char* full)
{
	char exepath[DMD_MAX_PATH+1];
	ExePath(exepath);
	std::string path = StripFile(exepath);

	//char full[DMD_MAX_PATH+1];
	sprintf(full, "%s", path.c_str());

	char c = full[ strlen(full)-1 ];
	if(c != '\\' && c != '/')
		strcat(full, "\\");
	//strcat(full, "/");

	strcat(full, filename);
	CorrectSlashes(full);
}

float StrToFloat(const char *s)
{
	if(s[0] == '\0')
		return 1.0f;

	float x;
	std::istringstream iss(s);
	iss >> x;

	if(_isnan(x))
		x = 1.0f;

	return x;
}

int32_t HexToInt(const char* s)
{
	int32_t x;
	std::stringstream ss;
	ss << std::hex << s;
	ss >> x;
	return x;
}

int32_t StrToInt(const char *s)
{
	int32_t x;
	std::istringstream iss(s);
```

```
	iss >> x;
	return x;
}

void CorrectSlashes(char* corrected)
{
	int32_t strl = strlen(corrected);
	for(int32_t i=0; i<strl; i++)
		if(corrected[i] == '\\')
			corrected[i] = '/';
}

void BackSlashes(char* corrected)
{
	int32_t strl = strlen(corrected);
	for(int32_t i=0; i<strl; i++)
		if(corrected[i] == '/')
			corrected[i] = '\\';
}

void ErrMess(const char* title, const char* message)
{
	SDL_ShowCursor(true);
	SDL_ShowSimpleMessageBox(SDL_MESSAGEBOX_ERROR, title, message, NULL);
	SDL_ShowCursor(false);
}

void InfoMess(const char* title, const char* message)
{
	SDL_ShowCursor(true);
	SDL_ShowSimpleMessageBox(SDL_MESSAGEBOX_INFORMATION, title, message,
NULL);
	SDL_ShowCursor(false);
}

void WarnMess(const char* title, const char* message)
{
	SDL_ShowCursor(true);
	SDL_ShowSimpleMessageBox(SDL_MESSAGEBOX_WARNING, title, message, NULL);
	SDL_ShowCursor(false);
}

void OutOfMem(const char* file, int32_t line)
{
	char msg[2048];
	sprintf(msg, "Failed to allocate memory in %s on line %d.", file, line);
	ErrMess("Out of memory", msg);
	//g_quit = true;
```

```
}

std::string LoadTextFile(char* strFile)
{
	std::ifstream fin(strFile);

	if(!fin)
	{
		Log("Failed to load file %s\r\n", strFile);
		return "";
	}

	std::string strLine = "";
	std::string strText = "";

	getline(fin, strText);   //corpd fix

	while(getline(fin, strLine))
		strText = strText + "\n" + strLine;

	fin.close();

	return strText;
}

void ListFiles(const char* fullpath, std::list<std::string>& files)
{
	DIR *dp;
	struct dirent *dirp;

	if((dp  = opendir(fullpath)) == NULL)
	{
		g_applog << "Error opening " << fullpath << std::endl;
		return;
	}

	while ((dirp = readdir(dp)) != NULL)
	{
		if(dirp->d_type != DT_REG)
			continue;

		files.push_back(std::string(dirp->d_name));
	}

	closedir(dp);
	return;
}
```

```
void ListDirs(const char* fullpath, std::list<std::string>& dirs)
{
	DIR *dp;
	struct dirent *dirp;

	if((dp  = opendir(fullpath)) == NULL)
	{
		g_applog << "Error opening " << fullpath << std::endl;
		return;
	}

	while ((dirp = readdir(dp)) != NULL)
	{
		if(dirp->d_type != DT_DIR)
			continue;

		if(strcmp(dirp->d_name, ".") == 0)
			continue;
		if(strcmp(dirp->d_name, "..") == 0)
			continue;

		dirs.push_back(std::string(dirp->d_name));
	}

	closedir(dp);
	return;
}

//add thousands separators
std::string iform(int32_t n)
{
	std::string s;

	char first[32];
	sprintf(first, "%d", n);

	for(int32_t i=strlen(first)-3; i>=0; i-=3)
	{
		if(first[i] == '-' || i == 0 || first[i-1] == '-')
			break;

		int32_t newl = strlen(first)+1;

		for(int32_t j=newl; j>=i; j--)
		{
			first[j+1] = first[j];
		}
```

```
                first[i] = ',';
        }

        s = first;

        return s;
}

std::string ullform(uint64_t n)
{
        std::string s;

        char first[32];
        sprintf(first, "%llu", n);

        for(int32_t i=strlen(first)-3; i>=0; i-=3)
        {
                if(first[i] == '-' || i == 0 || first[i-1] == '-')
                        break;

                int32_t newl = strlen(first)+1;

                for(int32_t j=newl; j>=i; j--)
                {
                        first[j+1] = first[j];
                }

                first[i] = ',';
        }

        s = first;

        return s;
}

uint64_t GetTicks()
{
        //return time(0);
#ifdef PLATFORM_WIN
        //return GetTickCount64();
        SYSTEMTIME st;
        GetSystemTime (&st);
        _FILETIME ft;
        SystemTimeToFileTime(&st, &ft);
        //LARGE_INTEGER lint;
        //lint.HighPart = ft.dwHighDateTime;
   //lint.LowPart = ft.dwLowDateTime;
        //convert from 100-nanosecond intervals to milliseconds
```

```
        return (*(uint64_t*)&ft)/(10*1000);
#else
        struct timeval tv;

        gettimeofday(&tv, NULL);

        return
   (uint64_t)(tv.tv_sec) * 1000 +
   (uint64_t)(tv.tv_usec) / 1000;
#endif
}
```