
SORA
HOSHIZORA

Hoshizora API文档

Rev 1

Author: Griffin Bu

核心宏, 数据结构和杂项函数

SoraPlatform.h

sora::hash_map

哈希表数据结构, 依赖于不同平台和不同编译器有不同实现, 主要是std::hash_map (vs), __gnu_cxx::hash_map(gcc)和tr1::unordered_map(高版本gcc和vs)

mymalloc
myfree
myrealloc

内存申请/释放函数, 在USE_KMALLOC宏没有定义的情况下是malloc, free和realloc, 否则是kmalloc, kfree, krealloc(依赖于kmalloc)

SoraSharedPtr

带引用计数的智能指针，在USE_TR1_SHARED_PTR宏没有定义的情况下是SoraAutoPtr, 否则是tr1::shared_ptr

lrint(double x)

转换浮点数到最近的整数

int32
uint32
int16
uint16
int8
uint8
float32
long32
ulong32
int64
uint64

数据类型定义, 注意ulong32在不同位数的编译器下长度不同, 用于指针. 需要保持32位长度不变的数据, 例如RGBA颜色, 用int32/uint32.

HSORASPRITE
HSORATEXTURE
HGUIWIDGET
HSORTARGET

handle类型定义, 实际上都是ulong32

SORA_LINE
SORA_TRIANGLES
SORA_TRIANGLES_FAN
SORA_TRIANGLES_STRIP
SORA_QUAD

绘画模式定义, 具体含义参见说明文档的SoraSprite一章的自定义顶点渲染.

SoraString
SoraWString
　　分别代表std::string和std::wstring

SORAEXPORT
　　dllexport前置, 用于windows dll导出. 在SORA_DLL_EXPORT宏被定义的情况下为__declspec(dllexport), 否则为空

SORACALL
　　__stdcall前置, 在SORA_STD_CALL宏被定义的情况下为__stdcall, 否则为空

strcmpnocase(str1, str2)
　　无大小写区别的字符串比较, 参数为两个字符串

OS_OSX
OS_WIN32
OS_LINUX
OS_IOS
OS_ANDROID
　　当前操纵系统平台定义

sora_fopenw(path, mode)
　　wchar_t类型的fopen, 参数为wchar_t*文件路径和开启模式, 和fopen含义等同

msleep(msec)
　　使程序sleep msec毫秒
　　在Windows下是调用WINAPI Sleep, 在其余平台是一个使用nanosleep的特殊实现

MB_OK
MB_OKCANCEL
MB_ICONERROR
MB_ICONSTOP
MB_ICONWARNING
MB_ICONINFORMATION

IDOK
IDCANCEL
IDABORT
IDRETRY
IDYES
IDNO
　　SoraCore::messageBox的code和返回值定义, 具体含义请参见API文档

_DEBUG
　　是否输出调试信息宏

SoraMath.h

F_PI
F_PI_2
F_PI_4
F_PI_6
F_PI_12

D_PI
D_PI_2
D_PI_4
D_PI_8
　　圆周率相关定义, 后边的数字为圆周率的x分之一. F_前缀表示float类型, D_前缀表示double类型

DGR_RAD(dgr)
　　转换角度值到分度值

RAD_DGR(rad)
　　转换分度值到角度值

SoraVector
　　等于hgeVector

SoraRect
　　等于hgeRect

soraswap(t1, t2)
　　一个Swap实现

```
float32 getDistanceSqr(T x, T y, T x1, T y1)
```
　　获取两点之间的距离的平方

```
float32 decompressUnitFloat(uint32 quantized, uint32 nBits)
```
　　有损压缩一个范围在0-1之间的float到nBits长的无符号整数, 返回32位长整数, 但是根据nBits的具体数目可以安全的转换到其他类型, 例如uint16(nBits = 16)

```
uint32 compressFloat(float32 value,
                     float32 min,
                     float32 max,
                     uint32 nBits)
```
恢复一个压缩到了nBits整数的float

```
uint32 compressFloat(float32 value,
                     float32 min,
                     float32 max,
                     uint32 nBits)
```
压缩一个范围在min-max之间的float到nBits长的无符号整数

```
float32 decompressFloat(uint32 quantized,
                        float32 min,
                        float32 max,
                        uint32 nBits)
```
恢复一个范围在min-max之间压缩到了nBits整数的float

SoraKeyInfo.h

SORA_KEY_(0-9)
SORA_KEY_(A-Z)
SORA_KEY_(F1-F12)
SORA_KEY_NUMPAD(0-9)
SORA_KEY_ESCAPE
SORA_LEY_BACKSPACE
SORA_KEY_TAB
SORA_KEY_ENTER
SORA_KEY_SPACE
SORA_KEY_SHIFT
SORA_KEY_ALT
SORA_KEY_SPACE
SORA_KEY_PAUSE
SORA_KEY_CAPSLOCK
SORA_KEY_NUMLOCK
SORA_KEY_SCROLLLOCK
SORA_KEY_PGUP
SORA_KEY_PGDN
SORA_KEY_HOME
SORA_KEY_END
SORA_KEY_INSERT
SORA_KEY_DELETE
SORA_KEY_LEFT
SORA_KEY_RIGHT

SORA_KEY_UP
SORA_KEY_DOWN
SORA_KEY_GRAVE
SORA_KEY_MINUS
SORA_KEY_EQUALS
SORA_KEY_BACKSLASH
SORA_KEY_LBRACKET
SORA_KEY_RBRACKET
SORA_KEY_SEMICOLON
SORA_KEY_APOSTROPHE
SORA_KEY_COMMA
SORA_KEY_PERIOD
SORA_KEY_SLASH
SORA_KEY_LWIN
SORA_KEY_RWIN
SORA_KEY_APPS
    各个键盘按键的按键代码

SORA_KEY_LBUTTON
SORA_KEY_RBUTTON
SORA_KEY_MBUTTON
    鼠标左, 中, 右键的按键代码

SORA_JOYSTICK_(1-16)
    手柄的按键代码

按键状态定义:
SORA_INPUT_PRESSED
SORA_INPUT_KEYDOWN
    按键按下

SORA_INPUT_KEYUP
    按键抬起

SORA_INPUT_MBUTTONDOWN
SORA_INPUT_MBUTTONUP
    鼠标按键按下/抬起(暂时无用)

SORA_INPUT_MOUSEMOVE
    鼠标移动
SORA_INPUT_MOUSEWHEEL
    鼠标滚轮移动

```
bool isKeyPrintable(int key)
```
获取按键是否可以被转换为可以打印的字符.

参数:
key
    目标按键代码

返回值:
    按键是否可以被转换为可打印的字符

```
char toasciiWithFlag(int key, int flag)
```
转换按键到可打印的字符

参数:
key
    目标按键代码
flag
    目标按键状态(用于判断shift键是否被按下)

返回值:
    可打印的字符

SoraSprite.h

```
BLEND_COLORADD
BLEND_COLORMUL
BLEND_ALPHAADD
BLEND_ALPHABLEND
BLEND_ZWRITE
BLEND_NOZWRITE

BLEND_DEFAULT =     (BLEND_COLORMUL |
                     BLEND_ALPHABLEND |
                     BLEND_NOZWRITE)
BLEND_DEFAULT_Z =   (BLEND_COLORMUL |
                     BLEND_ALPHABLEND |
                     BLEND_ZWRITE)
```
    渲染混合模式定义

```
struct SoraVertex {
    float x, y;          顶点位置
    float z;             顶点z深度
    uint32col;           顶点颜色
    float tx, ty;        顶点贴图坐标
};

struct SoraQuad {
    SoraVertex      v[4];
    SoraTexture*    tex;       贴图
    int             blend;     渲染混合模式
};

struct SoraTriple {
    SoraVertex      v[3];
    SoraTexture*    tex;       贴图
    int             blend;     渲染混合模式
};
```

SoraFont.h

FONT_ALIGNMENT_LEFT
    左对齐
FONT_ALIGNMENT_RIGHT
    右对齐
FONT_ALIGNMENT_CENTER
    中对齐

SoraShader.h

VERTEX_SHADER
    顶点着色器
FRAGMENT_SHADER
    片段着色器

SoraColor.h

CARGB(a, r, g, b)
    返回一个RGBA = (r, g, b, a)值的无符号32位长整数表示的RGBA颜色, 参数取值0-1

CGETA(col)
CGETR(col)
CGETG(col)
CGETB(col)
    获取一个无符号32位长整数颜色的RGBA分量

CSETA(col, a)
CSETR(col, r)
CSETG(col, g)
CSETB(col, b)
    设置一个无符号32位长整数颜色的RGBA分量

SoraImageEffect.h

IMAGE_EFFECT_END
IMAGE_EFFECT_NOSTART
IMAGE_EFFECT_PLAYING
IMAGE_EFFECT_PAUSED
    当前ImageEffect播放状态定义, update函数的返回值

IMAGE_EFFECT_ONCE
IMAGE_EFFECT_PINGPONG
IMAGE_EFFECT_REPEAT
    ImageEffect的播放模式定义

SoraEnvValues.h

SET_ENV(name, val)
GET_ENV(name, default)
    设置和获取EnvValue宏, name为stingid, val为要设置的值, default为默认值

SET_ENV_INT(name, val)
SET_ENV_FLOAT(name, val)
SET_ENV_BOOL(name, val)
SET_ENV_STRING(name, val)
SET_ENV_DATA(name, val)

GET_ENV_INT(name, default)
GET_ENV_FLOAT(name, default)
GET_ENV_BOOL(name, default)
GET_ENV_STRING(name, default)
GET_ENV_DATA(name, default)
    设置和获取EnvValue宏, name为字符串名字, val为要设置的值, default为默认值

IEFade
IEShake
IEScale
IETransitions
IEColorTransitions
IERotation
    各类型ImageEffect的简化名字定义

Rect4V.h

```
Rect4V Rect4VByRect(const hgeRect& rect, float rot=0.f)
```
    旋转一个hgeRect rot角度, 返回用Rect4V表示的旋转过后的矩形

common.h

DeleteSTLPtr
DeleteSTLPairPtr
    两个仿函数结构, 用于stl的for_each算法, 删除某种容器内储存的指针

```
T cFileReadT(FILE* file, bool& err)
```
    从一个file文件读取模板类型T的值

参数:
file
    源文件
[ref]err
    将被设置为读取过程中是否发生错误

返回值:
    读取的值

```
bool cFileWriteT(FILE* file, T i)
```
    将一个模板类型T写入file

参数:
file
    要写入的file
T
    要写入的值

返回值:
    是否发生错误

SoraException.h

SORA_EXCEPTION(mess)
    抛出一个SoraException类型的异常, mess为信息, 自动记录函数, 文件和所处代码行数

graphicEffects.h

```
int gaussBlur(unsigned long *data,
              int width,
              int height,
              double sigma,
              int radius)
```
    高斯模糊一个贴图数据

参数:
data
    贴图数据
width
    贴图宽
height
    贴图高
sigma
    模糊sigma
radius
    模糊半径

返回值:
    成功则1, 否则0

```
int gray(unsigned long *data,
         int width,
         int height)
```
    把一个贴图数据表示为灰度图

参数:
data
    贴图数据
width
    贴图宽
height
    贴图高

返回值:
    成功则1, 否则0

```
int reverse(unsigned long* data,
            int width,
            int height);
```
    将一个贴图数据反相

参数:
data
    贴图数据
width
    贴图宽
height
    贴图高

返回值:
    成功则1, 否则0

```
int alphaMix(unsigned long* data1,
             unsigned long* data2,
             int width,
             int height,
             int width2,
             int height2,
             double a,
             int posx=0,
             int posy=0);
```
    Alpha混合两张贴图数据

参数:
data1
    贴图数据1
data2
    贴图数据2
width
    贴图1宽
height
    贴图1高
width2
    贴图2宽
height2
    贴图2高
a
    混合比例
posx
    贴图2写入贴图1的x坐标
posy
    贴图2写入贴图1的y坐标

SoraLocalizer.h

GET_LOCAL_STR(ident)
    获取标识为ident的本地化字符串

GET_LOCAL_RESOURCE(ident)
    获取标识为ident的本地化资源文件名

LOCALIZE_RESOURCE_NAME(ident)
    把当前locale加入ident文件名的后缀之前, 例如test.png而locale=chn则ident会被设置为test_chn.png

SoraStringConv.h

int_to_str(n)
    转换int到字符串

fp_to_str(fp)
    转换float到字符串

ws2s(ws)
    转换宽字符串到字符串

s2ws(s)
    转换字符串到宽字符串

anyToString(t, precision)
    转换任意类型的数据到字符串

anyToWString(t, precision)
    转换任意类型的数据到宽字符串

ws2sfast(ws)
    快速转换只含英文字符的宽字符串到字符串

s2wsfast(s)
    快速转换只含英文字符的字符串到宽字符串

```
void deliStr(std::vector<std::string>& c,
             const std::string& str,
             char deli);
```
    用特定的字符分割一个字符串

参数:
[ref]c
    用于储存分割后的子字符串
str
    要分割的字符串
deli
    分割符

stringid.h

str2id(str)
    转换字符串到字符串id(cache模式)

str2idnc(str)
    转换字符串到字符串id(nocache模式), 将无法转换id回到字符串

id2strw
    转换字符串id到宽字符串

id2str
    转换字符串id到字符串

SoraSoundFile.h

SORASOUND_SUCCESS
SORASOUND_FAILED_OPEN_DEVICE
SORASOUND_FAILED_OPEN_SOUND
SORASOUND_FAILED_OPEN_DISK_FILE
SORASOUND_FAILED_OPEN_MEMORY_FILE
    SoraMusicFile和SoraSoundEffectFile的readFile返回值定义

SoraPlaybackEvent.h

SORAPB_EV_NULL
SORAPB_EV_PLAY_STARTED
SORAPB_EV_PLAY_ENDED
SORAPB_EV_PLAY_RESUMED
SORAPB_EV_PLAY_STOPPED
    SoraPlaybackEvent的eventype定义

SoraRTTI.h

RTTI具体信息请参考Hoshizora说明文档&教程的RTTI一节, 这里不给出具体定义了

RTTI_METHOD
    RTTI描述一个类函数

RTTI_OVERLOAD_METHOD_(1-12)
    RTTI描述一个重载的类函数

RTTI_FIELD
    RTTI描述一个类成员变量
RTTI_DESCRIBED_FIELD
    RTTI描述一个可以get和set的类成员变量(目标必须用DESCRIBE_FIELD(ARRAY)描述过)

RTTI_PTR
    RTTI描述一个指针
RTTI_DESCRIBED_PTR
    RTTI描述一个可以get和set的指针(目标必须用DESCRIBE_FIELD(ARRAY)描述过)

RTTI_ARRAY
    RTTI描述一个数组
RTTI_DESCRIBED_ARRAY
    RTTI描述一个可以get和set的数组(目标必须用DESCRIBE_FIELD(ARRAY)描述过)

RTTI_BASE_CLASS
    RTTI描述基类

RTTI_DEFAULT_CONSTRUCTOR
RTTI_CONSTRUCTOR_(1-8)
    RTTI描述构造函数

RTTI_NO_METHOD
RTTI_NO_FIELD
RTTI_NO_BASE_CLASS
RTTI_NO_CONSTRUCTOR
    RTTI没有特定成员的占位符

DESCRIBE_FIELD
    RTTI描述类成员变量
DESCRIBE_ARRAY
    RTTI描述类数组成员变量

DESCRIBE_CLASS
    RTTI描述类

REGISTER_CLASS
    RTTI注册类

SoraRTTIType.h

RTTI_FLAG_VIRTUAL
RTTI_FLAG_STATIC
RTTI_FLAG_PUBLIC
RTTI_FLAG_PROTECTED
RTTI_FLAG_PRIVATE
    各个类型的RTTI flag, 用于RTTI描述宏

RTTI_FLAG_TO_STR(flag)
    转换RTTI flag到对应的字符串

RTTI_UNKNOWN
RTTI_VOID
RTTI_BOOL
RTTI_INT8
RTTI_UINT8
RTTI_INT16
RTTI_UINT16
RTTI_INT32
RTTI_UINT32
RTTI_LONG32
RTTI_ULONG32
RTTI_INT64
RTTI_UINT64
RTTI_FLOAT
RTTI_DOUBLE
RTTI_CLASS
RTTI_DERIVED
RTTI_PTR
RTTI_ARRAY
RTTI_METHOD
RTTI_SRING
RTTI_WSTRING
RTTI_CONSTRUCTOR
    RTTIType tag定义.

SoraRTTIMethod.h

```
__RT ReturnValToVar(SoraRTTIType* type)
```
转换invoke之后method返回的RTTIType到特定类型

```
SoraRTTIVar<__RT>* ReturnValToRTTIVar(SoraRTTIType* type)
```
转换invoke之后method返回的RTTIType到某个类型的SoraRTTIVar, 需要手动释放内存

SoraRTTIField.h

```
void SoraRTTIFieldSet(SoraRTTIFieldDescriptor* descriptor,
                      void* obj,
                      __T t)
```
快速设置某个描述过的(DESCRIBE_FIELD & RTTI_DESCRIBED_FIELD)RTTI类成员变量的值

```
__T SoraRTTIFieldGet(SoraRTTIFieldDescriptor* descriptor,
                     void* obj)
```
快速获取某个描述过的 (DESCRIBE_FIELD & RTTI_DESCRIBED_FIELD) RTTI类成员变量的值

```
void SoraRTTIFieldVarSet(SoraRTTIFieldDescriptor* descriptor,
                         void* obj,
                         SoraRTTIVar<__T>* t)
```
快速设置某个描述过的(DESCRIBE_FIELD & RTTI_DESCRIBED_FIELD)RTTI类成员变量的值, SoraRTTIVar版

```
SoraRTTIVar<__T>* SoraRTTIFieldVarGet(
    SoraRTTIFieldDescriptor* descriptor,
    void* obj
)
```
快速设置某个描述过的(DESCRIBE_FIELD & RTTI_DESCRIBED_FIELD)RTTI类成员变量的值, SoraRTTIVar版, 需要手动释放内存

Part 1. 核心组件

SoraCore:
Sora引擎的核心类, 封装了Sora的基础核心功能.

```
void start()
```

启动引擎, 最低要求有RenderSystem已经注册.

```
void shutDown()
```

关闭引擎, 销毁主窗口, 退出程序.

```
void update()
```

引擎主update函数, RenderSystem的实现必须调用此函数. (依赖于RenderSystem的实现, 对于需要注册回调的RenderSystem, SoraInifiniteRendercallback.h内提供了基本的回调函数)

```
bool isActive()
```

当前窗口是否活动. (当前并非所有平台有效)

```
void registerRenderSystem(SoraRenderSystem* pRenderSystem)
```

注册渲染器. 所有的RenderSystem必须继承自SoraRenderSystem基类并实现他的虚接口.

```
void registerResourceManager(
    SoraResourceManager* pResourceManager
)
```

注册一个资源管理器, 重复调用此函数将构造一个ResourceManager列表. 所有的Resource相关函数都将依赖于已经注册的ResourceManager. ResourceManager使用的优先级为注册顺序.

```
void registerInput(SoraInput* pInput)
```

注册输入管理器.

```
void registerFontManager(SoraFontManager* pFontManager)
```

注册字体管理器.

```
void registerSoundSystem(SoraSoundSystem* pSoundSystem)
```

注册声音管理器.

```
void registerMiscTool(SoraMiscTool* pMiscTool)
```

注册系统杂项函数实现类. (Sora核心自带各平台的实现)

```
void registerPluginManager  (SoraPluginManager* pPluginManager)
```

注册插件管理器. (Sora核心自带实现)

```
void registerTimer(SoraTimer* pTimer)
```

注册时间管理器. (Sora核心自带各平台的实现)

```
void registerPlugin(SoraPlugin* pPlugin)
```

加载一个SoraPlugin插件.

参数:
pPlugin
    要加载的插件的指针

```
void unistallPlugin(SoraPlugin* pPlugin)
```

卸载一个SoraPlugin插件.

参数:
pPlugin
    要卸载的插件的指针

```
void unistallPluginS(const SoraString& sPluginName)
```

依据插件名字卸载一个SoraPlugin插件.

参数:
sPluginName
    要卸载的插件的名字

```
SoraPlugin* getPlugin(const SoraString& sPluginName)
```

依据插件名字获取一个SoraPlugin插件.

参数:
sPluginName
    要获取的插件的名字

```
void setFPS(int32 fps)
```

设置FPS. FPS恒定的维持依赖于Timer实现.

参数:
fps
    需要保持的fps.

```
float32 getFPS()
```

获取当前运行的FPS.

```
float32 getDelta()
```

获取当前运行的帧间隔.

see also:
SoraCore::setTimeScale
soraCore::getTimeScale

```
float32 getTime()
```

获取引擎已经运行的时间.

```
int32 getFrameCount()
```

获取引擎已经运行的帧数.

```
void setTimeScale(float32 scale)
```

设置引擎的时间流逝比例.
* 影响getDelta的结果. 所有依赖于delta的组件都将收到影响.

参数:
scale
    时间流逝比例, 默认为1.0.

```
float32 getTimeScale()
```

获取当前引擎的时间流逝比例.

```
s_int64 getCurrentSystemTime()
```

获取当前的系统时间.
* 结果依赖于Timer实现, 在不同平台下单位可能不同. 最低精度保证到毫秒(基础Timer).

```
void beginScene(ulong32 c=0, ulong32 h=0)
```

开始一个Scene, 在任何渲染指令执行之前必须调用此函数.
参数:
c
    屏幕清空背景颜色
h
    渲染target, 如果为0则是屏幕.

```
void endScene()
```

结束一个Scene, 刷新帧缓冲到屏幕或者Target.

```
HSORATARGET createTarget(int width,
                         int height,
                         bool zbuffer=true)
```

创建一个target, 用于渲染

参数:
width
    target宽
height
    target高
zbuffer
    是否允许zbuffer

返回值:
    创建的target, 如果为0则创建失败

```
void freeTarget(ulong32 t)
```

释放一个target

参数:
t
    target的handle, createTarget的返回值

```
HSORATEXTURE getTargetTexture(ulong32 t)
```

获取Target的贴图

参数:
t
    target的handle, createTarget的返回值

```
HSORATEXTURE createTexture(const SoraWString& sTexturePath,
                           bool bCache=true,
                           bool bMipmap=false)
```

创建一张贴图, 支持jpeg, png, bmp等. (依赖于各RenderSystem的实现, 在不同平台不同RenderSystem的情况下支持格式可能不同)

参数:
sTexturePath
    贴图的路径. 如果硬盘上不存在, 则会从已经加载的资源包内寻找
bCache
    是否缓冲贴图, 如果缓冲, 则会使用引用计数技术, 加快需要加载同样的贴图时的效率.
bMipmap
    是否为mipmap

返回值:
    创建的贴图的handle, 如果为0则创建失败

```
HSORATEXTURE createTextureWH(int32 w, int32 h)
```

依据宽高创建一张空的贴图.

参数:
w
    贴图宽
h
    贴图高

返回值:
    创建的贴图, 如果为0则创建失败

```
HSORATEXTURE createTextureFromRawData(uint32* data,
                                      int32 w,
                                      int32 h)
```

从RGBA原始数据创建一张贴图, 数据必须为32位长整数表示的RGBA贴图数据.

参数:
data
    贴图数据
w
    贴图宽
h
    贴图高

返回值:
    创建的贴图, 如果为0则创建失败

```
HSORATEXTURE createTextureFromMem(void* data, ulong32 size)
```
从内存中图像数据创建一张贴图, 可以是支持的贴图格式的数据. 例如使用fopen打开文件然后读取的数据.

参数
data
    贴图数据
size
    数据长度

返回值:
    创建的贴图, 如果为0则创建失败

```
uint32* textureLock(HSORATEXTURE h)
```

锁定一张贴图, 获取他的原始颜色数据.

参数
h
    贴图的handle, createTexture*的返回值

返回值:
    贴图的颜色数据, 以32位长整数表示的RGBA值, 每个值占8位, 取值0-255.

see also:
SoraCore::textureUnlock

```
void textureUnlock(HSORATEXTURE h)
```

解除对一张贴图的锁定, 必须先调用textureLock. 把对于textureLock返回的颜色数据的更改写入贴图.

参数
h

    贴图的handle, createTexture*的返回值.
int32 getTextureWidth(HSORATEXTURE h, bool origin=false)

获取一张贴图的宽
参数:
h
    贴图的handle, createTexture*的返回值
origin
    是否为贴图的原始宽度. 如果为false, 则会离贴图原始宽度最近的2的n次方值.

返回值:
    贴图的宽

int32 getTextureHeight(HSORATEXTURE, bool origin=false)

获取一张贴图的高度

参数:
h
    贴图的handle, createTexture*的返回值
origin
    是否为贴图的原始高度. 如果为false, 则会离贴图原始高度最近的2的n次方值.

返回值:
    贴图的高

ulong32 getTextureId(HSORATEXTURE h)

获取贴图id, 依赖于平台和RenderSystem的实现有不同含义.
例如在Windows+DX环境下, 这个函数应该返回(ulong32)(LPDIRECT3DTEXTURE9)
而在OGL环境下, 则返回OpenGL的texture id

参数:
h
    贴图的handle, createTexture*的返回值

返回值:
    贴图id

void releaseTexture(HSORATEXTURE h)

释放一张贴图

参数:
h
    贴图的handle, createTexture*的返回值

```
void clearTextureMap()
```

强制清空贴图的缓存数据, 释放占用的内存. 但是将导致所有的贴图都需要重新加载.

```
SoraShaderContext* createShaderContext()
```

创建一个ShaderContext, ShaderContext是在渲染时要使用的Shader的管理器.

返回值:
    创建的ShaderContext, 如果创建失败则返回NULL

```
void attachShaderContext(SoraShaderContext* context)
```

使用一个ShaderContext. 这个函数必须在具体的渲染函数之前调用, 以使下一个渲染函数使用这个ShaderContext.

参数:
context
    要使用的ShaderContext

```
void detachShaderContext()
```

分离正在使用的ShaderContext. 这个函数将使正在使用的ShaderContext失效. 一般用于渲染函数之后.

```
SoraShader* createShader(const SoraWString& file,
                         const SoraString& entry,
                         SORA_SHADER_TYPE type)
```

创建一个Shader.

参数:
file
    Shader文件
entry
    Shader入口函数
type
    Shader类型, 有FRAGMENT_SHADER和VERTEX_SHADER两种

返回值:
    创建的Shader, 如果失败则返回NULL

```
SoraSprite* createSprite (const SoraWString& sPath)
```

创建一个精灵.

参数:
sPath
    要创建的精灵的位置.

返回值:
    创建的精灵, 如果创建失败则返回NULL.

```
void renderQuad(SoraQuad& quad)
```

渲染一个四边形.

参数:
quad
    要渲染的四边形

```
void renderTriple(SoraTriple& trip)
```

渲染一个三角形.

参数:
trip
    要渲染的三角形

```
void renderWithVertices(HSORATEXTURE tex,
                        int32 blendMode,
                        SoraVertex* vertices,
                        uint32 vsize,
                        int32 mode=SORA_TRIANGLES)
```

以指定的模式和顶点渲染一张贴图.

参数:
tex
    要渲染的贴图
blendMode
    渲染模式
vertices
    要渲染的顶点数组
vsize
    顶点数组的大小

mode
    描绘模式, 可用SORA_LINE, SORA_TRIANGLES, SORA_TRIANGLES_FAN, SORA_TRIANGLES_STRIP, SORA_QUAD.

```
void renderRect(float32 x1,
                float32 y1,
                float32 x2,
                float32 y2,
                float32 fWidth=1.f,
                ulong32 color=0xFFFFFFFF,
                float32 z=0.0f)
```

渲染一个矩形区域

参数:
x1
    区域左上角x
y1
    区域左上角y
x2
    区域右下角x
y2
    区域右下角y
fWidth
    区域宽, 如果等于1则会视区域为(x1, y1) - (x2, y2)的直线
color
    渲染颜色
z
    渲染z

```
void renderBox(float32 x1,
               float32 y1,
               float32 x2,
               float32 y2,
               ulong32 color,
               float32 z=0.f)
```

渲染一个矩形区域的边框

参数:
x1
    区域左上角x
y1
    区域左上角y

x2
    区域右下角x
y2
    区域右下角y
color
    渲染颜色
z
    渲染z

```
void setClipping(int32 x=0, int32 y=0, int32 w=0, int32 h=0)
```

设置裁剪区域, 会转换当前坐标系到目标区域. 所有之后的渲染指令都会被影响.

参数:
x
    区域x坐标
y
    区域y坐标
w
    区域宽
h
    区域高

```
void setTransform(float32 x=0.f,
                  float32 y=0.f,
                  float32 dx=0.f,
                  float32 dy=0.f,
                  float32 rot=0.f,
                  float32 hscale=0.f,
                  float32 vscale=0.f)
```

设置屏幕举证的变换, 所有之后的渲染指令都会被影响

参数:
x
    中心x坐标
y
    中心y坐标
dx
    中心x坐标的位移
dy
    中心y坐标的位移
rot
    旋转角度

hscale
    横向放大倍数

vscale
    纵向放大倍数

```
void beginZBufferSort()
```

开始针对z值的渲染排序. 所有之后的渲染指令都会依照z值从大到小进行排序然后渲染. 当你的贴图需要启用zbuffer但是没有透明像素的时候你并不需要使用此函数.

```
void endZBufferSort()
```

结束针对z值的渲染排序.

```
ulong32 getMainWindowHandle()
```

返回主窗口的Handle, 依据平台和RenderSystem的实现结果不同.
例如在Windows下应当返回(ulong32)(HWND)
而OpenGL下则单纯返回(ulong32)(MainWindow)

返回值:
    主窗口的Handle

```
SoraWindowInfoBase* getMainWindow()
```

获取主窗口指针

返回值:
    主窗口的指针.

```
void enableMessageBoxErrorPost(bool bFlag)
```

启用MessageBox式的错误报告. 默认为false. 当启用时, 所有SoraCore内部的_postError错误信息都将以MessageBox的方式呈现

参数:
bFlag
    是否启用

```
void setRandomSeed(int32 seed)
```

设置随机数种子.

参数:

seed
    要设置的随机数种子

int32 getRandomSeed()

获取当前的随机数种子.

返回值:
    随机数种子

int32 randomInt(int32 min, int32 max)

生成一个范围在(min, max)之类的整数随机数.

参数:
min
    范围最小值
max
    范围最大值

返回值:
    生成的随机数

float32 randomFloat(float32 min, float32 max)

生成一个范围在(min, max)之间的浮点随机数.

参数:
min
    范围最小值
max
    范围最大值

返回值:
    生成的随机数

int32 randomIntNoRange()

生成一个范围在(0, maxint)之间的整数随机数

返回值:
    生成的随机数

```
float32 randomFloatNoRange()
```

生成一个范围在(0, 1)之间的浮点随机数

返回值:
    生成的随机数

```
int32 getScreenWidth()
```

获取主窗口的宽

返回值:
    主窗口宽

```
int32 getScreenHeight()
```

获取主窗口的高

返回值:
    主窗口高

```
HSORARESOURCE loadResourcePack(const SoraWString& file)
```

加载一个资源包

参数:
file
    资源包的路径

返回值:
    资源包的Handle

```
void attachResourcePack(HSORARESOURCE h)
```

使用一个资源包

参数:
h
    资源包的Handle

```
void detachResourcePack(HSORARESOURCE h)
```

停止使用一个资源包

参数:

h
    资源包的Handle

```
void* getResourceFile(const SoraWString& file, ulong32& size)
```

读取一个资源文件的全部内容, 依赖于ResourceManager, 不光支持硬盘文件.

参数:
file
    要读取的资源文件的名字或者路径
[ref]size
    会被设置为被读取的资源数据的大小, 如果为0则读取失败

返回值:
    被读取的数据指针, 如果失败则为NULL.

```
void* readResourceFile(const SoraWString& file, ulong32 size)
```

读取一个资源文件的部分内容, 依赖于ResourceManager, 不光支持硬盘文件.

参数:
file
    要读取的资源文件的名字或者路径
size
    要读取的数据大小

返回值:
    被读取的数据指针, 如果失败则为NULL.

```
ulong32 getResourceFileSize(const SoraWString& file)
```

获取一个资源文件的大小, 依赖于ResourceManager, 不光支持硬盘文件.

参数:
file
    要获取的资源文件的名字或者路径

返回值:
    资源文件的大小, 如果为0则获取失败

```
void freeResourceFile(void* p)
```

释放一个资源文件的数据指针. 所有读取到的资源文件的指针必须由此函数释放. 不然将会导致内存泄露.

参数:
p
    读取的资源文件的指针.

```
void enumFilesInFolder(std::vector<SoraWString>& cont,
                       const SoraWString& folder)
```

获取一个文件夹内的所有文件. 依赖于ResourceManager的实现.

参数:
[ref]cont
    将被文件夹内的文件路径填充
folder
    目标文件夹路径

```
bool isMainWindowSet()
```

是否已经创建主窗口.

返回值:
    是否已经创建主窗口

```
ulong32 createWindow(SoraWindowInfoBase* info)
```

创建主窗口. 在调用start运行引擎之前必须先创建主窗口.

参数:
info
    主窗口实现类.

```
void setWindowSize(int32 w, int32 h)
```

设置主窗口大小.

参数:
w
    窗口宽
h
    窗口高

```
void setWindowTitle(const SoraWString& title)
```

设置主窗口的标题

参数:
title
    主窗口标题

```
void setWindowPos(int32 px, int32 py)
```

设置主窗口的位置

参数:
px
    窗口x坐标
py
    窗口y坐标

```
void setFullscreen(bool flag)
```

设置是否全屏运行.

参数:
flag
    是否全屏.

```
bool isFullscreen()
```

获取是否是全屏模式.

返回值:
    是否是全屏模式

```
void getMousePos(float32 *x, float32 *y)
```

获取鼠标当前的位置.

参数
[ptr]x
    将被设置为鼠标x坐标
[ptr]y
    将被设置为鼠标y坐标

```
float32 getMousePosX()
```

获取鼠标当前的x坐标

返回值:
    鼠标x坐标

```
float32 getMousePosY()
```

获取鼠标当前的y坐标.

返回值:
    鼠标y坐标

```
void setMousePos(float32 x, float32 y)
```

设置鼠标的位置.

参数:
x
    鼠标x坐标
y
    鼠标y坐标

```
int getMouseWheel()
```

获取鼠标滚轮的位置.

返回值:
    鼠标滚轮的位置

```
bool isMouseOver()
```

获取鼠标当前是否在主窗口之上.

返回值:
    鼠标是否在主窗口之上

```
bool keyDown(int32 key)
```

测试一个按键是否已经被按下, 移动平台不可用.

参数:
key
    要测试的键位

返回值:
    按键是否被按下

```
bool keyUp(int32 key)
```

测试一个按键是否为抬起状态, 移动平台不可用.

参数:
key
    要测试的键位

返回值:
    按键是否为抬起状态

```
int32 getKeyState(int32 key)
```

获取一个按键的当前状态, 移动平台不可用.

参数:
key
    要获取的键位

返回值:
    按键状态, 可能为SORA_INPUT_KEYDOWN或者SORA_INPUT_KEYUP

```
char* getKeyName(int32 key)
```

获取一个按键的描述, 移动平台不可用.

参数:
    要获取的键位

返回值:
    键位的描述, 例如”shift”

```
bool getKeyEvent(SoraKeyEvent& ev)
```

获取按键事件, 将弹出按键事件的队尾, 当没有按键事件时返回false. 移动平台不可用.

参数:
[ref]ev
    将被填充为按键事件队列的队尾事件的信息.

返回值:
    事件队列是否为空

```
bool joyKeyPressed(int32 key)
```

测试一个手柄按键是否已经被按下, 不一定所有平台可用.

参数:
key
    要测试的键位

返回值:
    按键是否被按下

```
bool joyKeyDown(int32 key)
```

测试一个手柄按键在当前帧是否被按下, 不一定所有平台可用.

参数:
key
    要测试的键位

返回值:
    按键是否在当前帧被按下

```
bool joyKeyUp(int32 key)
```

测试一个手柄按键在当前帧是否抬起, 不一定所有平台可用.

参数:
key
    要测试的键位

返回值:
    按键是否在当前帧抬起

```
bool joyKeyState(int32 key, unsigned char state)
```

测试一个手柄按键是否处于某个状态, 不一定所有平台可用.

参数:
key
    要测试的键位

state
    要测试的状态, 可用状态有SORA_JOYSTICK_PRESSED, SORA_JOYSTICK_DOWN, SORA_JOYSTICK_UP

返回值:
    按键是否处于指定的状态

```
bool setJoyKey(int32 key)
```

不可用函数

```
bool hasJoy()
```

是否手柄是否存在.

返回值:
    手柄是否存在.

```
void simulateKey(int32 key, int32 state)
```

模拟一个按键输入, 只在当前帧有效.

参数:
key
    要模拟的按键
state
    要模拟的状态, SORA_INPUT_KEYDOWN或者SORA_INPUT_KEYUP

```
int32 messageBox(const SoraString& sMssg,
                 const SoraString& sTitle,
                 int32 iCode)
```

唤出一个系统级别的MessageBox, 暂停当前任务. 不同平台有不同实现. 表现效果可能不同, 不一定所有平台可用.

参数
sMssg
    MessageBox要显示的信息
sTitle
    MessageBox的标题
iCode
    MessageBox的表现形式, 有

MB_OK	显示ok按钮
MB_OKCANCLE	显示ok和cancel按钮
MB_ICONERROR	显示错误图标, 并非所有平台可用
MB_ICONSTOP	显示停止图标, 并非所有平台可用
MB_ICONWARNING	显示警告图标, 并非所有平台可用
MB_ICONINFORMATION	显示信息图标, 并非所有平台可用

返回值
    被按下的按钮, 有
          IDOK   ok按钮被按下
          IDCANCEL   cancle按钮被按下

```
int32 messageBoxW(const SoraWString& sMssg,
                  const SoraWString& sTitle,
                  int32 iCode)
```

MessageBox的wstring版, 参见MessageBox

```
void log(const SoraString& sMssg,
         int32 level=LOG_LEVEL_NORMAL)
```

输出一条log信息, 你可以从Console看到.

参数:
sMssg
    要输出的信息
level
    输出信息的级别, 不同的级别在console里会有不用的颜色表示, 可用的值有
          LOG_LEVEL_NORMAL   普通级别, 白色
          LOG_LEVEL_WARNING   警告级别, 黄色
          LOG_LEVEL_NOTICE   提示级别, 蓝色
          LOG_LEVEL_ERROR   错误级别, 红色

```
void logw(const SoraWString& sMssg,
          int32 level=LOG_LEVEL_NORMAL)
```

log的wstring版, 参见log

```
void logf(const char* str, ...)
```

log一个指定format的字符串, 级别总是为LOG_LEVEL_NORMAL. 外部有vamssg函数可以以string的形式获取一个指定格式的字符串, 所以这个函数通常不用.

参数:
str
    字符串format

```
SoraWString fileOpenDialog(const char* filter = NULL,
                           const char* defaultPath = NULL)
```

唤出一个系统级别的打开文件对话框, 暂停当前任务. 不同平台有不同实现, 表现形式可能不用. 移动平台不可用.

参数:
filter
    文件扩展名过滤器, 不同平台有不同写法
    windows下是 “描述\0扩展名1;扩展名2;...\0\0”, 例如”pngfiles\0*.png\0\0”
    os x下则是 “扩展名1;扩展名2;...”, 例如”txt;doc”
defaultPath
    打开文件对话框的初始位置. 你可以通过SoraFileUtility::getApplicationPath()函数来获取当前程序的位置.

返回值:
    被打开的文件的路径. 如果对话框被取消则为一个空字符串(size=0)

```
SoraWString fileSaveDialog(const char* filter = NULL,
                           const char* defaultPath = NULL,
                           const char* defaultExt = NULL)
```

唤出一个系统级别的保存文件对话框, 暂停当前任务. 不同平台有不同实现, 表现形式可能不用. 移动平台不可用.

参数:
filter
    文件扩展名过滤器, 写法参见fileOpenDialog
defaultPath
    保存文件对话框的初始位置
defaultExt
    用户没有输入扩展名时的默认扩展名

返回值:
    被保存的文件的路径, 如果对话框被取消则为一个空字符串(size=0)

```
SoraFont* createFont(const SoraWString& fontName,
                     int size)
```

创建一个字体, 如果没有FontManager被注册, 则总是返回NULL. 注意有些FontManager实现带有缓存机制, Font最好重用而不要手动删除.

参数:
fontName
    字体名字或者路径. 引擎会自动试图寻找系统字体文件夹内的习题.
size
    字体大小

返回值:
    创建的字体, 如果FontManager不存在或者字体不存在则返回NULL

```
SoraMusicFile* createMusicFile(const SoraWString& musicName,
                               bool bStream=true)
```

创建一个可以播放的音乐文件, 如果没有SoundManager被注册, 则总是返回NULL. 支持的格式依赖于SoundManager实现.

参数:
musicName:
    要创建的音乐文件的路径
bStream
    是否以流模式创建音乐文件(不是所有SoundManager实现皆可正常运作)

返回值:
    创建的音乐文件, 如果SoundManager不存在或者文件创建失败则返回NULL.

```
SoraSoundEffectFile* createSoundEffectFile(
    const SoraWString& se
)
```

创建一个可以播放的音效文件, 如果没有SoundManager被注册, 则总是返回NULL. 支持的格式依赖于SoundManager实现.

参数:
se
    要创建的音效文件的路径

返回值:
    创建的音效文件, 如果SoundManager不存在或者文件创建失败则返回NULL.

```
SoraMusicFile* createMusicFile(bool bStream=false)
```

创建一个空的音乐文件, 创建成功后可以通过readFile来读取具体的音乐文件. 如果没有SoundManager被注册, 则总是返回NULL. 支持的格式依赖于SoundManager实现.

参数:
bStream
    是否以流模式创建

返回值:
    创建的音乐文件

```
SoraSoundEffectFile* createSoundEffectFile()
```

创建一个空的音效文件, 创建成功后可以通过readFile来读取具体的音效文件. 如果没有SoundManager被注册, 则总是返回NULL. 支持的格式依赖于SoundManager实现.

返回值:
    创建的音效文件

```
void setViewPoint(float32 x=0.f, float32 y=0.f, float32 z=0.f)
```

设置视点, 暂时不可用.

参数:
x
    视点x坐标
y
    视点y坐标
z
    视点z坐标

```
void execute(const SoraString& appPath, const SoraString& args)
```

执行系统命令行命令, 依赖于命令行实现. 或者打开一个程序.

参数:
appPath
    要执行的命令或者要打开的程序
args
    参数

```
void snapshot(const SoraString& path)
```

保存一张当前主窗口的截图

参数:
path
    截图的保存位置

```
ulong32 getVideoDeviceHandle()
```

获取当前视频设备的Handle, 依赖于RenderSystem, 对于不同渲染器返回值的含义不同.
DirectX下应当返回(ulong32)IDirect3DDevice9
而OpenGL下则单纯的返回(ulong32)RenderSystem

返回值:
    当前视频设备的Handle

```
SoraWString videoInfo()
```

获取当前视频设备和驱动的描述.

返回值:
    当前视频设备的描述

```
void flush()
```

强制清空渲染缓冲区到屏幕或者target. 通常情况下你不应手动调用这个函数, 除非你知道你在干什么.

```
void postError(const SoraString& sMssg)
```

发布一条错误信息, 和log的区别是在MessageBoxErrorPost开启的情况下会以MessageBox的形式展示这条信息, 否则将以LOG_LEVEL_ERROR级别log这条信息.

参数:
    要发布的错误信息

```
void setFrameSync(bool flag)
```

设置是否帧同步, 如果开启, 则getDelta函数将总是返回1.0. 默认为关.

参数:
flag
    是否开启

```
void addFrameListener(SoraFrameListener* listener)
```

添加一个FrameListener. FrameListener继承自SoraFrameListener, 每帧的开始和结束将会被调用回调函数.

参数:
listener

要添加的FrameListener指针

```
void delFrameListener(SoraFrameListener* listener)
```

移除一个FrameListener.

参数:
listener
要移除的FrameListener指针

```
s_int64 getEngineMemoryUsage()
```

获取当前引擎使用的内存大小, 以kb表示.

返回值:
引擎使用的内存大小, 0表示功能不可用.

```
SoraConsole* getConsole()
```

获取引擎内置的Console, Console的部分信息可以配置.

返回值:
获取的Console

```
void setSystemFont(const wchar_t* font, int32 fontSize)
```

设置引擎要使用的字体, 要求FontManager已经被注册. 如果这个函数没有被调用, 则Console无法渲染文字.

参数:
font
字体名字或者路径
fontSize
字体大小

SoraEnvValue.h

SoraEnvValues:
SoraEnvValues提供了一个全局的数据储存器, 方便在不同类, 在程序和脚本之间交换数据

[parent] SoraEnvValues* Instance()

获取SoraEnvValues的单实例

[parent] void Destroy()

删除SoraEnvValues的单实例

bool getValue(stringId name, bool defaultVal)

获取一个bool值

参数:
name
    以字符串id表示的要获取的值的名字
defaultVal
    在指定值不存在时返回的默认值

返回值:
    获取的值

int32 getValue(stringId name, int32 defaultVal)

获取一个int值

参数:
name
    以字符串id表示的要获取的值的名字
defaultVal
    在指定值不存在时返回的默认值

返回值:
    获取的值

float32 getValue(stringId name, float32 defaultVal)

获取一个float值

参数:
name

以字符串id表示的要获取的值的名字

defaultVal
在指定值不存在时返回的默认值

返回值:
获取的值

```
std::string getValue(stringId name,
                     const std::string& defaultVal)
```

获取一个string值

参数:
name
以字符串id表示的要获取的值的名字
defaultVal
在指定值不存在时返回的默认值

返回值:
获取的值

```
std::wstring getValue(stringId name,
                      const std::wstring& defaultVal)
```

获取一个wstring值

参数:
name
以字符串id表示的要获取的值的名字
defaultVal
在指定值不存在时返回的默认值

返回值:
获取的值

```
void* getValue(stringId name)
```

获取一个userdata值

参数:
name
以字符串id表示的要获取的值的名字
defaultVal
在指定值不存在时返回的默认值

返回值:
    获取的值
```
void setValue(stringId name, bool val)
```

设置一个bool值

参数:
name
    以字符串id表示的要获取的值的名字
val
    要设置的值

```
void setValue(stringId name, int32 val)
```

设置一个int值

参数:
name
    以字符串id表示的要获取的值的名字
val
    要设置的值

```
void setValue(stringId name, float32 val)
```

设置一个float值

参数:
name
    以字符串id表示的要获取的值的名字
val
    要设置的值

```
void setValue  (stringId name, const std::string& val)
```

设置一个string值

参数:
name
    以字符串id表示的要获取的值的名字
val
    要设置的值

```
void setValue  (stringId name, const std::wstring& val)
```

设置一个wstring值

参数:

name
    以字符串id表示的要获取的值的名字
val
    要设置的值

```
void setValue  (stringId name, void* data)
```

设置一个userdata值

参数:
name
    以字符串id表示的要获取的值的名字
val
    要设置的值

```
bool getBool(const std::string& name, bool defaultVal)
int32 getInt(const std::string& name, int32 defaultVal)
float32 getFloat(const std::string& name, float32 defaultVal)
std::string getString(const std::string& name,
                      const std::string& defaultVal)
std::wstring getWString(const std::string& name,
                        const std::wstring& defaultVal)
void* getData(const std::string& name)

void setBool(const std::string& name, bool val)
void setInt(const std::string& name, int32 val)
void setFloat(const std::string& name, float32 val)
void setString(const std::string& name, const std::string& val)
void setWString(const std::string& name,
                const std::wstring& val)
void setData(const std::string& name, void* data)
```

含义和setValue/getValue的重载相同, 为了脚本导出而存在.

```
void removeData(const std::string& name)
```

移除一个名字为name的值

```
void removeData(stringId name)
```

移除一个名字为name的值, 名字以字符串id表示

SoraCanvas.h

SoraCanvas:
SoraCanvs提供了针对RenderTarget的封装, 方便使用target进行渲染.

```
[constructor] SoraBaseCanvas(int32 width,
                             int32 height,
                             bool bDepthBuffer=true);
```

构造函数

参数:
width
    图层高
height
    图层宽
bDepthBuffer
    是否使用深度缓冲

```
void render()
```

渲染图层本身

```
uint32 update(float32 dt)
```

update图层本身

```
void beginRender()
```

开始图层内容的渲染, 相当于SoraCore::beginScene(color, CanvasTarget)

```
void finishRender()
```

结束图层内容的渲染, 相当于SoraCore::endScene()

```
void attachShader(SoraShader* shader)
```

在图层上附着一个Shader

参数:
shader
    要附着的shader

```
void detachShader(SoraShader* shader)
```

删除图层上已附着的一个Shader

参数:
shader
    要删除的shader

```
SoraShader* attachShader(const SoraWString& shaderPath,
                         const SoraString& entry,
                         SORA_SHADER_TYPE type)
```

在图层上附着一个从文件读取的Shader

参数:
shaderPath
    shader文件路径
entry
    shader入口函数
type
    shader类型, 有FRAGMENT_SHADER, VERTEX_SHADER

```
bool hasShader() const
```

获取图层上是否已经附着了shader

```
void clearShader()
```

清除图层上附着的所有shader, 会删除Shader指针

```
void addEffect(SoraImageEffect* effect)
```

给图层添加一个ImageEffect

参数:
effect
    要添加的ImageEffect

```
void stopEffect(SoraImageEffect* effect)
```

删除图层的一个ImageEffect

参数:
effect
    要删除的ImageEffect

```
void clearEffects()
```

清除图层上所有的ImageEffect

```
bool hasEffect() const
```

获取图层是否含有ImageEffect

```
SoraSprite* getCanvasSprite() const
```

获取图层的精灵

返回值:
    图层的精灵

SoraImageEffect.h

SoraImageEffect:
SoraImageEffect是所有图像变换特效的基类, 图像特效皆基于此实现

[constructors]
SoraImageEffect()

默认构造函数

SoraImageEffect(CoreTransformer<CoreTransform>* transformer)

以一个自定义的Transformer构造ImageEffect

SoraImageEffect(IMAGE_EFFECT_MODE _mode)

以特定模式构造ImageEffect

void stop()

停止特效播放, 将导致update返回IMAGE_EFFECT_END

void pause()

暂停特效播放

void pauseForTime(float32 t)

暂停特定的时间

void resume()

恢复特效的播放

bool finished() const

特效播放是否已经完成

float32 getTime()

获取特效已经播放的时间

uint8 getState() const

获取当前特效播放状态, 可能是IMAGE_EFFECT_END, IMAGE_EFFECT_NOTSTART, IMAGE_EFFECT_PLAYING, IMAGE_EFFECT_END

```
uint16 getType() const
```

获取特效的类型, 可能是IMAGE_EFFECT_FADEIN/OUT,
IMAGE_EFFECT_TRANSITIONS等

```
IMAGE_EFFECT_MODE getMode() const
```

获取特效的模式, 可能是. 模式播放次数可以同RepeatTimes来限制, 对于
IMAGE_EFFECT_PINGPONG和IMAGE_EFFECT_REPEAT默认为永久播放

```
    IMAGE_EFFECT_ONCE               播放一次
    IMAGE_EFFECT_PINGPONG           往返播放
    IMAGE_EFFECT_REPEAT             重复播放
```

```
virtual void start(IMAGE_EFFECT_MODE mode, float32 time)
```

开始特效的播放

参数:
mode
    特效播放模式
time
    特效播放时间

```
virtual uint32 update(float32 delta)
```

update特效

```
[interface]
virtual void effect(SoraSprite* sprite) = 0
```

变换精灵的属性, 由SoraSprite自动回调

```
void restart()
```

重新开始特效

```
void setTransformer(
    CoreTransformer<CoreTransform>* transformer
)
```

设置特效的Transformer

```
float32 get1st()
float32 get2nd()
float32 get3rd()
float32 get4th()
```

获取特效具体变换的值, 不同类型的特效使用数量不同.

```
float32 getEffectTime() const
```

获取特效要播放的时间

```
void setRepeatTimes(int32 times)
```

设置特效重复播放次数, 对于IMAGE_EFFECT_PINGPONG模式一个轮回为一次

```
uint32 getRepeatTimes() const
```

获取特效重复播放的次数

```
void swap()
```

交换特效变换的开始值和结束值

SoraImageEffectList:
SoraImageEffectList定义了一个特效执行链条, 用于配接特效的播放, 例如先放大然后移动.

```
SoraImageEffectList* add(SoraImageEffect* effect)
```

添加一个特效到队尾.

参数:
effect
    要添加的特效

返回值:
    EffectList自身, 用于连续添加的代码简化

```
SoraImageEffect* getListHead() const
```

获取特效链表的头

```
SoraImageEffect* getListTail() const;
```

获取特效链表的尾

```
void setListMode(IMAGE_EFFECT_MODE mode)
```

设置特效链表播放模式, 可用模式和ImageEffect相同.

```
IMAGE_EFFECT_MODE getListMode() const
```

获取特效链表的播放模式

```
virtual uint32 update(float32 delta)
```

update特效链表

```
virtual void start(IMAGE_EFFECT_MODE mode, float32 time)
```

开始特效链表的播放

```
virtual void effect(SoraSprite* spr)
```

实施对精灵变换, 自动由SoraSprite回调

以下是具体可用的ImageEffect实现, addEffect时你需要添加这些里面的一个

```
SoraImageEffectFade(
    float32 src,
    float32 dst,
    float32 time,
    IMAGE_EFFECT_MODE mode=IMAGE_EFFECT_ONCE
    CoreTransformer<CoreTransform>* transformer=0
)
```

淡入淡出特效

参数:

src
    起始值

dst
    结束值

time
    特效播放时间

mode
    特效播放模式

transformer
    自定义变化器

```
SoraImageEffectScale(
    float32 src,
    float32 dst,
    float32 time,
    IMAGE_EFFECT_MODE mode=IMAGE_EFFECT_ONCE,
    CoreTransformer<CoreTransform>* transformer=0
)
```

放大缩小特效

参数:
src
    起始值
dst
    结束值
time
    特效播放时间
mode
    特效播放模式
tranformer
    自定义变换器

```
SoraImageEffectScale(
    float32 srcV,
    float32 dstV,
    float32 srcH,
    float32 dstH,
    float32 time,
    IMAGE_EFFECT_MODE mode=IMAGE_EFFECT_ONCE,
    CoreTransformer<CoreTransform>* transformer=0
)
```
参数:
srcV
    横向缩放起始值
dstV
    横向缩放结束值
srcH
    纵向缩放起始值
dstH
    纵向缩放结束值
time
    特效播放时间
mode
    特效播放模式
tranformer
    自定义变换器

```
SoraImageEffectTransitions(
    float32 sx,
    float32 sy,
    float32 sz,
    float32 dx,
    float32 dy,
    float32 dz,
    float32 time,
    IMAGE_EFFECT_MODE mode=IMAGE_EFFECT_ONCE,
    CoreTransformer<CoreTransform>* transformer=0
)
```

坐标变换特效

参数:

sx
    起始x坐标
sy
    起始y坐标
sz
    起始z坐标
dx
    结束x坐标
dy
    结束y坐标
dz
    结束z坐标
time
    特效播放时间
mode
    特效播放模式
tranformer
    自定义变换器

```
SoraImageEffectTransitions(
    float32 sx,
    float32 sy,
    float32 dx,
    float32 dy,
    float32 time,
    IMAGE_EFFECT_MODE mode=IMAGE_EFFECT_ONCE,
    CoreTransformer<CoreTransform>* transformer=0
)
```

参数:

sx
    起始x坐标
sy
    起始y坐标
dx
    结束x坐标
dy
    结束y坐标
time
    特效播放时间
mode
    特效播放模式
tranformer
    自定义变换器

```
SoraImageEffectColorTransitions(
    const SoraColorRGBA& start,
    const SoraColorRGBA& end,
    float32 time,
    IMAGE_EFFECT_MODE mode=IMAGE_EFFECT_ONCE,
    CoreTransformer<CoreTransform>* transformer=0
)
```

颜色变换特效

参数:
start
    起始颜色
end
    结束颜色
time
    特效播放时间
mode
    特效播放模式
tranformer
    自定义变换器

```
SoraImageEffectColorTransitions(
    ulong32 start,
    ulong32 end,
    float32 time,
    IMAGE_EFFECT_MODE mode=IMAGE_EFFECT_ONCE,
    CoreTransformer<CoreTransform>* transformer=0
)
```

参数:
start
    起始颜色
end
    结束颜色
time
    特效播放时间
mode
    特效播放模式
tranformer
    自定义变换器

```
SoraImageEffectRotation(
    float32 start,
    float32 end,
    float32 time,
    IMAGE_EFFECT_MODE mode=IMAGE_EFFECT_ONCE,
    CoreTransformer<CoreTransform>* transformer=0
)
```

旋转特效

参数:
start
    起始值
end
    结束值
time
    特效播放时间
mode
    特效播放模式
tranformer
    自定义变换器

```
SoraImageEffectRotation(
    float32 start,
    float32 startz,
    float32 end,
    float32 endz,
    float32 time,
    IMAGE_EFFECT_MODE mode=IMAGE_EFFECT_ONCE,
    CoreTransformer<CoreTransform>* transformer=0
)
```

参数:

start
    起始值
startz
    y轴旋转起始值
end
    结束值
endz
    y轴旋转结束值
time
    特效播放时间
mode
    特效播放模式
tranformer
    自定义变换器

SoraEvent.h

SoraEvent:
Sora核心事件类的基类.

```
void setSource(SoraEventHandler* source)
```

设置Event的发送者

```
SoraEventHandler* getSource() const
```

获取Event的发送者

```
void setName(stringId name)
```

设置Event的名字

```
stringId getName() const
```

获取Event的名字的字符串id

```
void consume()
```

申明这个Event已经被使用, 可能影响事件响应链

```
bool isConsumed() const
```

获取Event是否已经被使用

SoraFont.h

SoraFont
所有字体实现的基类, 掩盖字体的具体实现, 统一接口.

```
void render(float32 x,
            float32 y,
            const wchar_t* text,
            bool hcenter = false,
            bool vcenter = false)
```

渲染一个字符串

参数:
x
    起始x坐标
y
    起始y坐标
text
    要渲染的字符串
hcenter
    是否纵向中对齐
vcenter
    是否横向中对齐

```
void print(float32 x,
           float32 y,
           int32 align,
           const wchar_t *format, ...)
```

渲染一个指定格式的字符串

参数:
x
    起始x坐标
y
    起始y坐标
align
    对齐方式, 有FONT_ALIGNMENT_LEFT, FONT_ALIGNMENT_RIGHT,
FONT_ALIGNMENT_CENTER
format
    格式化字符串

```
void render(float32 x,
            float32 y,
            int32 align,
            const wchar_t* text)
```

渲染一个字符串

参数:
x
    起始x坐标
y
    起始y坐标
align
    对齐方式
text
    要渲染的字符串

```
void setColor(ulong32 dwColor, int32 i = -1)
```

设置字体的颜色, -1代表所有顶点, 顶点取值0-3

参数:
dwColor
    要设置的颜色
i
    顶点index

```
ulong32 getColor(int32 i=0)
```

获取字体的颜色

参数:
i
    指定的顶点

返回值:
    顶点的颜色

```
void setKerningWidth(float32 kerning)
```

设置横向字符间距

参数:
kerning
    字符间距

```
void setKerningHeight(float32 kerning)
```

设置纵向字符间距

参数:
kerning
    字符间距

```
float32 getKerningWidth()
```

获取横向字符间距

```
float32 getKerningHeight()
```

获取纵向字符间距

```
float32 getStringWidth(const wchar_t* text)
```

获取一个字符串的长度

参数:
text
    字符串

返回值:
    字符串的长度

```
float32 getStringHeight(const wchar_t* text)
```

获取一个字符串的高度

参数:
text
    字符串

返回值:
    字符串的高度

```
float32 getHeight()
```

获取字体高度

```
uint32 getFontSize()
```

获取字体大小

```
float32 getWidthFromCharacter(wchar_t c, bool original = false)
```

获取一个字符的宽度

参数:
c
    字符
origin
    是否原始宽度

返回值:
    字符宽度

```
void setCharRotation(float32 rot)
```

设置字体旋转角, 渲染时对每个字符单独旋转

```
void setScale(float32 scale)
```

设置字体放大倍数

SoraKeyInfo.h

SoraKeyEvent
按键事件的事件类

int type
事件类型

int key
按键

int flags
特殊键标记

int chr
ASCII码

int wheel
鼠标滚轮位置

float x
鼠标x坐标

float y
鼠标y坐标

bool isKeyDown() const

是否为按键按下事件

bool isKeyUp() const

是否为按键抬起事件

int getKey() const

获取按下/抬起的键

bool isKeyPressed(int k) const

是否为某个键被按下

bool isKeyReleased(int k) const

是否为某个键抬起

```
bool isShiftFlag() const
```

shift键是否同时被按下/抬起

```
bool isCtrlFlag() const
```

ctrl键是否同时被按下/抬起

```
bool isAltFlag() const
```

alt键是否同时被按下/抬起

SoraNamedObject.h

SoraNamedObject
所有带有名字的Object的基类, 包括SoraObject

```
void setName(stringId n)
```

设置名字

```
stringId getName() const
```

获取名字

```
virtual void serialize(SoraMemoryBuffer& bufferStream)
```

序列化名字到bufferStream

```
virtual void unserialize(SoraMemoryBuffer& bufferStream)
```

从bufferStream内读取名字

SoraNamedObjectList

支持迭代器(Iterator)的针对NamedObject特化的链表

SoraObject.h

SoraObject
Sora核心一个物件的代表

```
virtual uint32 update(float32 dt)
```

update物件

```
virtual void render();
```

渲染物件

```
void add(AP_OBJECT pobj)
```

添加子物件

```
void del(AP_OBJECT pobj)
```

删除子物件

```
void setPosition(float32 _x, float32 _y)
```

设置物件位置

```
float32 getPositionX()
```

获取物件x坐标

```
float32 getPositionY()
```

获取物件y坐标

```
void getPosition(float32& _x, float32& _y);
```

获取物件位置

```
SUB_OBJECT_LIST getObjList() const
```

获取子物件链表

```
AP_OBJECT getParent() const;
```

获取父物件

```
void setParent(AP_OBJECT obj)
```

设置父物件

```
AP_OBJECT getObjByName(const SoraString& n)
```

在子物件中查找名字为n的子物件, 没有则返回NULL

```
uint32 getType() const
```

获取物件类型

```
void setType(uint32 t)
```

设置物件类型

SoraMemoryBuffer.h

SoraMemoryBuffer
内存数据缓冲

```
void set(void* pData, ulong32 _length)
```

设置数据

参数
pData
    指向内存数据块的指针

_length
    数据块长度

```
bool alloc(ulong32 size)
```

申请一定大小的内存

参数:
size
    要申请的内存大小

返回值:
    是否申请成功

```
void resize()
```

尝试减小buffer的长度到实际数据的长度

```
bool push(void* pdata, ulong32 size)
```

向内存块添加部分数据, 这个函数在当前buffer空间不足的时候会动态扩大buffer的大小

参数:
pdata
    要添加的数据
size
    要添加的数据的长度

返回值:
    是否添加成功

```
template<typename T>
bool push(T t)
```

添加某一个类型的数据, 等同于push(&t, sizeof(t)). 注意Little-Endian和Big-Endian表示

返回值:
    是否添加成功

```
bool writeToFile(const SoraWString& path)
```

把缓冲数据写入文件, 在写入前会先试图resize

参数:
path
    要写入的文件

返回值:
    是否写入成功

```
template<typename T>
T read()
```

从缓冲中读取某一类型的数据

返回值:
    读取的数据

```
bool read(void* pv, ulong32& size)
```

从缓冲中读取指定大小的数据

参数:
pv
    读取内存缓冲, 大小必须大于或者等于size
[ref]size
    要读取的大小, 如果buffer大小小于size, 则size会被设置为实际读取的大小

```
bool read(ulong32 offset, void* pv, ulong32& size);
```

从指定偏移处读取指定大小的数据, 不影响当前的偏移位置

参数:
offset
    偏移量

pv
    读取内存缓冲，大小必须大于或者等于size
[ref]size
    要读取的大小，如果buffer大小小于size，则size会被设置为实际读取的大小

```
uint8* get()
```

获取指向buffer的指针，注意偏移量为当前的偏移．要获取内存数据块的头请先seek或者用get(offset)

返回值:
    指向内存块的指针，偏移量为当前偏移量

```
uint8* get(ulong32 offset)
```

获取指定偏移量的指向buffer内存的指针

参数:
Offset
    偏移量

返回值:
    获取的指针

```
template<typename datatype>
datatype& get(ulong32 offset) const
```

从指定偏移的位置读取一个类型为datetype的数据，不会影响当前偏移量

参数:
offset
    偏移量

返回值:
    读取的值

```
bool seek(ulong32 pos)
```

设置当前偏移量为pos

返回值:
    是否设置成功

```
ulong32 size() const
```

获取当前buffer的size

```
ulong32 realsize() const
```

获取当前实际数据的size(在alloc + push下这个值总是小于等于size, 在set + read/get下这个值总是等于sie)

```
ulong32 offset() const
```

获取当前偏移量

```
bool valid() const
```

获取buffer是否可用

SoraSprite.h

SoraSprite
精灵类, 继承自SoraObject

```
void render()
```

渲染精灵, 位置为SoraObject的位置

```
void render(float32 x, float32 y)
```

在指定位置渲染精灵

```
void render4V(float32 x1,
              float32 y1,
              float32 x2,
              float32 y2,
              float32 x3,
              float32 y3,
              float32 x4,
              float32 y4)
```

渲染精灵到指定的矩形内, 参数为矩形4个顶点的坐标

```
void renderWithVertices(SoraVertex* vertices,
                        uint32 size,
                        int32 mode);
```

以指定的模式渲染到自定义顶点内

参数:
vertices
    要渲染的顶点数组
size
    顶点数组大小
mode
    渲染模式, 有SORA_LINE, SORA_TRIANGLES, SORA_TRIANGLES_FAN, SORA_TRIANGLES_STRIP, SORA_QUAD

```
void setTexture(HSORATEXTURE tex)
```

设置精灵的贴图

```
void setTextureRect(float32 x,
                    float32 y,
                    float32 width,
                    float32 height)
```

设置精灵贴图区域

```
hgeRect getTextureRect() const
```

获取精灵贴图区域

```
void setColor(uint32 c, int32 i=-1)
```

设置顶点颜色, -1代表所有顶点, 顶点取值0-3

```
uint32 getColor(int32 i=0) const
```

获取指定顶点的颜色

```
void setZ(float32 z, int32 i=-1)
```

设置顶点的z深度值, -1代表所有顶点

```
float32 getZ(int32 i=0) const
```

获取指定顶点的z深度值

```
void setCenter(float32 x, float32 y)
```

设置精灵的中心坐标

```
void getCenter(float32& x, float32& y)
float32 getCenterX() const
float32 getCenterY() const
```

获取精灵的中心坐标

```
void setFlip(bool hflag, bool vflag, bool bFlipCenter=true)
```

设置精灵的翻转

参数:
hflag
    横向翻转

vflag
    纵向翻转
bFlipCenter
    是否翻转中心坐标

```
bool getHFlip() const
```

获取精灵是否横向翻转过

```
bool getVFlip() const
```

获取精灵是否纵向翻转过

```
void setBlendMode(int32 mode)
```

设置渲染混合模式, 有必须为各个分量的结合值, 默认有BLEND_DEFAULT和BLEND_DEFAULT_Z

```
int32 getBlendMode() const
```

获取渲染混合模式

```
int32 getTextureWidth(bool bOriginal=true) const
```

获取精灵贴图宽度

参数:
bOriginal
    是否为原始宽度, 如果是则返回贴图的实际宽度, 否则返回里宽度最近的2的N次方的值

```
int32 getTextureHeight(bool bOriginal=true) const;
```

获取精灵贴图高度

参数:
bOriginal
    是否为原始高度, 如果是则返回贴图的实际高度, 否则返回里宽度最近的2的N次方的值

```
int32 getSpriteWidth() const
```

获取精灵的宽度

```
int32 getSpriteHeight() const
```

获取精灵的高度

```
int32 getSpritePosX() const
```

获取精灵贴图区域的x坐标

```
int32 getSpritePosY() const
```

获取精灵贴图区域的y坐标

```
void setScale(float32 h, float32 v)
```

设置缩放比例

参数:
h
    横向缩放比例
v
    纵向缩放比例

```
float32 getVScale() const
float32 getHScale() const
```

获取精灵的横向和纵向缩放比例

```
void setRotation(float32 r)
```

设置精灵的旋转角度

```
float32 getRotation() const
```

获取精灵的旋转角度

```
void setRotationZ(float32 rz)
```

设置精灵绕y轴旋转的角度

```
float32 getRotationZ() const
```

获取精灵绕y轴旋转的角度

```
uint32* getPixelData() const
```

获取精灵贴图的RGBA图像数据

```
void unlockPixelData()
```

把更改过的贴图数据写入到贴图数据

```
HSORATEXTURE getTexture() const
```

获取精灵的贴图

```
uint32 update(float32 dt)
```

update精灵, 在有effect存在时必须调用此函数以更新精灵

```
void addEffect(SoraImageEffect* effect)
void stopEffect(SoraImageEffect* effect);
void clearEffects();
bool hasEffect() const;

void attachShader(SoraShader*);
void detachShader(SoraShader*);
SoraShader* attachShader(const SoraWString& shaderPath,
                         const SoraString& entry,
                         SORA_SHADER_TYPE type);
bool hasShader() const;
void clearShader();
```

ImageEffect和Shader相关函数, 含义和SoraCanvas内同名函数相同, 参见SoraCanvans的定义

SoraLocalizer.h

SoraLocalizer
本地化字符串和资源文件名

```
bool addLocaleConf(const SoraWString& confPath);
```

加载一个本地化配置文件,
写法和ini类似, 有两种格式
文件必须以@LOCALE开头, 例如
@chn
test = “测试”
“billboard.png” = “billboard_chn.png”

每个文件只能包含一种locale定义
其中test = “测试”定义了本地化字符串, 在项目中你可以把所有需要本地化的字符串都用GET_LOCAL_STR(ident)宏替代, 这样依据SoraLocalizer当前的locale不同, 字符串会自动转换为各个本地化后的字符串
而”billboard.png” = “billboard_chn.png”则定义了本地化资源文件名, 用于资源中本身带有本地化内容的情况, 可以用GET_LOCAL_RESOURCE(name)获取

参数:
confPath
    要加载的配置文件路径

返回值:
    是否加载成功

```
SoraWString getStr(const SoraString& ident)
```

获取一个本地化字符串, 依赖于当前locale

参数:
ident
    字符串的标识

返回值:
    获取的本地化字符串

```
void setCurrentLocale(const SoraString& localeIdent)
```

设置当前的locale, 例如”chn”

参数:
localeIdent
    locale的标识, 必须和配置文件@之后的字符串一样

```
SoraString getCurrentLocale() const
```

获取当前locale

```
SoraWString localizeResourceName(SoraWString& resourceName)
```

添加当前locale到资源文件路径中, 例如当前locale为chn而resourceName为”particle.png”, 则resourceName会被设置为”particle_chn.png”