



PainterEngine API manual

PainterEngine API manual.....	1
Core.....	9
预编译.....	9
计算数组个数.....	9
日志/调试信息.....	10
输出日志信息.....	10
错误信息.....	10
断言.....	10
取得日志文本.....	10
内存.....	11
内存拷贝.....	11
内存设值.....	11
按 4 字节对齐设置内存值.....	11
内存比对.....	11
内存池.....	12
创建内存池.....	12
返回一块内存指针所指向内存的内存大小.....	12
从内存池分配一块内存.....	12
从内存池释放一块内存.....	13
重置内存池.....	13
设置内存池异常回调.....	14
字符串操作.....	14

转换.....	19
字节类型转十六进制字符串.....	19
十六进制字符串转字节类型.....	19
十六进制字符转数字.....	19
字符转数字.....	20
字符串转 IPv4 地址.....	20
IPv4 地址转字符串.....	20
数字转字符串.....	20
弧度角度转换.....	21
字节序.....	22
主机字节序转网络字节序.....	22
网络字节序转主机字节序.....	22
数学.....	22
素数模乘同余发生器.....	22
高斯随机数发生器.....	23
对数函数.....	23
三角函数.....	24
平方根.....	24
方差.....	25
协方差.....	25
求绝对值.....	25
tanh.....	25
sigmoid.....	26
ReLU.....	26
四舍五入.....	26
幂.....	26
复数.....	27
矩阵.....	28
点/向量.....	30
交叉碰撞.....	33
快速排序(由小到大).....	33
快速排序(由大到小).....	34
矩形.....	34
CRC 校验 (CRC32)	35
CRC 校验 (CRC16)	35
累加和校验 (Sum32)	35
第一类修正贝塞尔函数.....	36
通讯链接器(IO 虚拟层).....	36
初始化链接器.....	36
链接器读.....	37
链接器写.....	37
数据结构.....	37

取得结构体偏移量.....	37
字母.....	37
链表.....	38
容器.....	41
字符串映射表.....	43
自适应内存.....	45
内存自适应字符串.....	48
几何绘制.....	52
线.....	52
边框.....	53
矩形.....	53
实心圆.....	53
圆.....	54
圆笔.....	54
环.....	54
扇形.....	55
圆角矩形.....	55
实心圆角矩形.....	56
描边路径.....	56
三角形.....	56
箭头.....	57
绘制贝塞尔曲线.....	57
图形图像.....	58
渲染表面.....	58
纹理.....	61
图像算子.....	70
轮廓.....	72
位图.....	74
TRaw 纹理格式.....	76
Delaunary 三角划分.....	77
带限制区域的 Delaunary 三角建立.....	77
纹理特效.....	78
字体.....	78
绘制字体.....	78
字模初始化.....	79
加载字模.....	79
释放字模库.....	79
绘制单个字.....	80
计算绘制文本的宽高.....	80
信号处理.....	80
离散傅里叶变换 (DFT)	80
离散余弦变换 (DCT)	81

快速傅里叶变换 (FFT)	81
强制共轭对称.....	82
上采样(插值, UpSampled).....	82
下采样(抽取, SubSampled, DownSampled).....	83
tukey 窗.....	83
triangular 窗.....	83
blackMan 窗.....	83
hamming 窗.....	84
hanning 窗.....	84
kaiser 窗.....	84
应用窗函数.....	84
计算系统幅频响应和相频响应.....	85
计算级联型系统幅频响应和相频响应.....	85
倒谱.....	85
人声基音频率估算.....	86
估算瞬时频率.....	86
预加重.....	86
卡尔曼滤波.....	87
MFCC.....	87
WAV 文件格式.....	88
混音器.....	89
调音台.....	92
反向传播神经网络.....	95
初始化神经网络框架.....	95
添加网络层.....	95
训练网络.....	96
前向传播.....	96
取得输出结果.....	97
重置神经网络.....	97
释放神经网络框架.....	97
导出神经网络框架.....	97
导入神经网络框架.....	97
物理.....	98
四叉树 AABB 碰撞检测.....	98
密码学.....	99
AES.....	99
curve25519.....	100
SHA256.....	101
编码及数据压缩.....	101
BASE64.....	101
ARLE.....	102
Huffman.....	103

3D 渲染管线.....	103
Kernel.....	109
词法分析机.....	109
初始化词法分析机.....	109
注册注释块.....	109
注册包含块.....	109
注册空白符.....	110
注册分隔符.....	110
取得分隔符 ID.....	110
取得字符串包含块类型.....	110
释放词法分析机资源.....	111
加载分析文本.....	111
读取文本.....	111
取得特殊符号.....	111
取得当前词.....	112
取得包含文本.....	112
设置词大小写敏感.....	112
取得词法分析机当前状态.....	112
设置词法分析机当前状态.....	113
判断当前词是否是合法数字.....	113
读取下一个字母.....	113
读取下一个词.....	113
取得当前词性.....	114
PainterScript 语法规则.....	114
脚本语法.....	114
PainterScript 简化编译器.....	126
初始化编译器.....	126
添加源代码.....	126
编译源代码.....	126
释放编译器.....	127
PainterScript Asm 汇编语法.....	127
标志.....	127
汇编指令集.....	127
表达式.....	138
PainterScript Asm 汇编器.....	138
编译汇编代码.....	138
PainterVM 虚拟机.....	139
vm 实例.....	139
host 函数定义.....	139
执行虚拟机函数.....	140

运行虚拟机.....	141
线程上下文切换.....	142
终止线程.....	142
挂起线程.....	142
恢复线程.....	143
注册 host 函数.....	143
释放虚拟机.....	143
类型访问.....	143
命令调试系统.....	145
Obj 静态 3D 模型加载器.....	145
加载器初始化.....	145
解析 Object 数据.....	146
数据转换到 RenderList.....	146
释放加载器.....	146
2dx 动画描述语言.....	146
指令.....	147
编译.....	148
2dx 动画.....	148
2dx library.....	148
2dx animation.....	149
PainterEngine LiveFramework.....	153
导入 LiveFramework.....	153
LiveFramework 动画播放.....	154
LiveFramework 暂停播放.....	154
重置 LiveFramework.....	154
停止 LiveFramework.....	155
渲染 LiveFramework.....	155
PainterEngine Live.....	155
创建 Live.....	155
释放 Live.....	156
Live 动画播放.....	156
Live 暂停播放.....	156
重置 live.....	156
停止 live.....	157
渲染 live.....	157
粒子系统.....	157
粒子发射器参数.....	157
以默认值初始化一个发射器参数.....	158
创建粒子系统.....	159
渲染粒子发射器.....	159
释放粒子发射器.....	159

设置粒子发射器的位置.....	159
设置粒子发射器喷射方向.....	160
MQTT 协议.....	160
初始化协议.....	160
MQTT 连接.....	160
发布数据.....	161
订阅数据.....	161
取消订阅数据.....	162
MQTT Ping.....	162
监听.....	162
释放.....	163
资源管理器.....	163
初始化.....	163
加载一个资源.....	163
添加一个贴图资源.....	164
读取资源.....	164
读取纹理资源.....	164
读取轮廓资源.....	165
读取动画资源.....	165
读取脚本资源.....	165
读取声音资源.....	165
读取数据资源.....	166
删除资源.....	166
释放资源库.....	166
帧同步协议.....	166
同步服务端.....	167
帧客户端.....	169
JSON.....	170
初始化.....	171
读取 JSON Object 的子数据.....	171
读取 JSON 数据.....	171
读取 JSON Array 数据.....	172
解析 JSON 数据.....	172
建立 JSON 数据类型.....	172
将 JSON 结构转换为文本.....	172
删除 JSON 结构中一个数据.....	173
释放 JSON.....	173
JSON 数据建立.....	173
PainterEngine 基本对象.....	178
使用 PainterEngine 绘制对象.....	178
PainterEngine 对象机制.....	178

创建对象.....	184
设置对象的 usercode.....	185
删除对象.....	185
对象属性设置.....	185
PainterEngine 更新/绘制流水线.....	186
对象事件机制.....	187
事件类型.....	187
响应处理函数.....	188
注册对象事件.....	188
派分事件.....	189
构造事件.....	189
构造字符串事件.....	189
指针事件偏移.....	190
取得指针事件的 X 坐标.....	190
取得指针事件的 Y 坐标.....	190
取得指针事件的 Z 坐标.....	190
设置指针事件的 X 坐标.....	191
设置指针事件的 Y 坐标.....	191
设置指针事件的 X 坐标.....	191
取得键盘事件的输入码.....	191
取得输入法事件的输入字符串.....	192
PainterEngine UI 对象.....	192
静态文本框.....	192
进度条.....	194
图片框.....	196
滑动框.....	197
按钮.....	199
文本编辑框.....	203
列表框.....	207
虚拟键盘.....	210
勾选框.....	213
菜单.....	216
下拉框.....	217
单选框.....	220
文件浏览器.....	223
子窗口.....	226
消息对话框.....	230
PainterEngine UI 设计框架.....	232
基本数据类型.....	232
UI 类型及对应属性.....	233
label 静态文本框.....	233
progressbar 进度条.....	233

image 图像框.....	234
sliderbar 滑动框.....	234
pushbutton/cursorbutton 按钮.....	234
edit 文本编辑框.....	234
scrollarea 滚动区域.....	235
autotext 自动换行静态文本框.....	235
virtualkeyboard 虚拟键盘.....	235
virtualnumberkeyboard 虚拟数字键盘.....	235
checkbox 选择框.....	236
radiobox 单选框.....	236
selectbar 下拉框.....	236
Widget 子窗体.....	237
游戏世界框架.....	237
游戏世界初始化.....	237
取得游戏世界 Object 的个数.....	237
删除 Object.....	238
在世界查找 Object.....	238
添加 Object.....	238
更新游戏世界.....	239
渲染游戏世界.....	239
设置摄像机.....	239
设置游戏对象的碰撞类型.....	239
设置游戏世界辅助线间隔.....	240
设置游戏世界辅助线.....	240
设置游戏世界辅助线的颜色.....	240
世界坐标转换为屏幕坐标.....	241
释放游戏世界.....	241

Core

预编译

计算数组个数

函数名	PX_COUNTOF(x)
说明	计算一个数组的个数
参数	fmt 日志信息

返回值	
-----	--

日志/调试信息

输出日志信息

函数名	void PX_LOG(char fmt[]);
说明	输出一段日志信息
参数	fmt 日志信息
返回值	

错误信息

函数名	void PX_ERROR(char fmt[]);
说明	输出一段错误日志信息
参数	fmt 日志信息
返回值	

断言

函数名	void PX_ASSERT(void);
说明	断言函数,debug 模式下执行该函数将触发一个异常中断
参数	fmt 日志信息
返回值	

取得日志文本

函数名	char *PX_GETLOG(void);
说明	取得日志文本
参数	

返回值	日志文本指针
-----	--------

内存

内存拷贝

函数名	void px_memcpy(void *dst,void *src,px_uint size);
说明	内存拷贝
参数	dst:指向目标内存 src:指向源内存 size:需要拷贝的内存大小

内存设值

函数名	void px_memset(void *dst,px_byte byte,px_uint size)
说明	设置内存值
参数	Dst 目标内存指针 Byte 值 Size 设置内存大小

按 4 字节对齐设置内存值

函数名	void px_memdwordset(void *dst,px_dword dw,px_uint count)
说明	按 4 字节对齐设置内存值
参数	Dst 目标内存指针 dw 值 count 需要设置的四字节个数

内存比对

函数名	px_bool px_memequ(void *dst,void *src,px_uint size)
说明	比较目标内存是否相同
参数	Dst 目标内存指针 src 目标内存指针 Size 比较大小
返回值	如果相同返回 PX_TRUE,否者返回 PX_FALSE

内存池

创建内存池

```
//Create a memory pool & return a MemoryPool structure
//MemoryAddr :Start address of memory
//MemorySize :Size of memory pool
```

函数名	MemoryPool MP_Create (void *MemoryAddr,unsigned int MemorySize);
说明	创建一个内存池
参数	MemoryAddr :内存池的开始地址 MemorySize :需要创建的内存池的大小 返回一个 MemoryPool 结构体
返回值	内存池结构

返回一块内存指针所指向内存的内存大小

```
//Get memory size of Ptr
//Pool: Pool MemoryPool structure pointer
//Ptr: memory pointer
//Return - if succeeded return the size of Ptr,else return zero
```

函数名	px_uint MP_Size(px_memorypool *Pool,px_void *Ptr);
-----	--

说明	返回一块内存指针所指向内存的内存大小
参数	Pool 内存池 Ptr 指针
返回值	该指针所指向内存块大小

从内存池分配一块内存

```
//Alloc a memory from memory pool
//Pool: Pool MemoryPool structure pointer
//Size: Size of alloc
//Return - if succeeded return the begin address of memories
//      if faith return null
```

函数名	void *MP_Malloc (MemoryPool *Pool,unsigned int Size);
说明	从内存池分配一块内存 该内存将会优先从已回收内存节点中分配,如果回收节点无法分配再从空余空间分配
参数	Pool : 指向 MemoryPool 的结构体, 这个结构体使用 MP_Create 进行创建 Size : 需要分配的大小 如果成功返回这个内存的指针, 如果失败返回 null/0
返回值	分配的内存指针,如果分配失败将会返回 PX_NULL

从内存池释放一块内存

```
//Free the memory from memory pool
//Pool: Pool MemoryPool structure pointer
//pAddress: Pointer memory need to be free
```

Pool : 指向 MemoryPool 的结构体, 这个结构体使用 MP_Create 进行创建
pAddress:内存首地址指针

```
void MP_Free (MemoryPool *Pool,void *pAddress);
void MP_Release (MemoryPool *Pool);
```

函数名	void MP_Free (MemoryPool *Pool,void *pAddress);
说明	释放一个内存节点

参数	Pool：指向 MemoryPool 的结构体，这个结构体使用 MP_Create 进行创建 pAddress:内存首地址指针
返回值	-

重置内存池

void MP_Reset (MemoryPool *Pool);

函数名	void MP_Reset (MemoryPool *Pool);
说明	重置内存池, 重置内存池将该内存池所有内存分配节点快速释放,对于某类归类或计算算法的加速尤为有用
参数	Pool：指向 MemoryPool 的结构体，这个结构体使用 MP_Create 进行创建
返回值	-

设置内存池异常回调

函数名	px_void MP_ErrorCatch(px_memorypool *Pool,PX_MP_ErrorCall ErrorCall);
说明	设置内存池的错误回调函数
参数	Pool：指向 MemoryPool 的结构体，这个结构体使用 MP_Create 进行创建 ErrorCall：异常回调函数
返回值	-

typedef px_void (*PX_MP_ErrorCall)(PX_MEMORYPOOL_ERROR);

当内存池触发异常时将会执行该异常回调函数,如果没有设置异常回调将会以 PX_ERROR 进行处理

异常回调消息包括

PX_MEMORYPOOL_ERROR_OUTOFMEMORY 内存池内存不足

PX_MEMORYPOOL_ERROR_INVALID_ACCESS 内存池指向内存无法读/写访问

PX_MEMORYPOOL_ERROR_INVALID_ADDRESS 释放无效的内存地址

字符串操作

字符串拷贝函数

dst:指向目标内存
src:指向源字符串
size:目标内存的大小
void px_strcpy(px_char *dst,px_char *src,px_uint size);

宽字符串拷贝函数

dst:指向目标内存
src:指向源字符串
size:目标内存的大小
void px_wstrcpy(px_word *dst,px_word *src,px_uint size);

字符串长度

函数名	px_int px_strlen(px_char *dst)
说明	字符串长度
参数	字符串指针
返回值	返回长度

宽字符字符串长度

函数名	px_int px_wstrlen(px_word *dst)
说明	宽字符字符串长度
参数	字符串指针
返回值	返回长度

字符串设置

函数名	px_void px_strset(px_char *dst, const px_char *src);
说明	将目标字符串拷贝为源字符串
参数	Dst 目标字符串原指针 src 源字符串
返回值	-

字符串拼接

函数名	px_void px_strcat(px_char *src, const px_char *cat);
说明	字符串拼接
参数	src 字符串原指针 cat 需要拼接到 src 的字符串
返回值	-

宽字符串拼接

函数名	px_void px_wstrcat(px_word *src, const px_word *cat);
说明	宽字符串拼接
参数	src 字符串原指针 cat 需要拼接到 src 的字符串
返回值	-

字符串转大写

函数名	px_void px_strupr(px_char *src);
说明	将 src 指向字符串转换为大写
参数	src 字符串原指针
返回值	-

字符串转小写

函数名	px_void px_strlwr(px_char *src);
说明	将 src 指向字符串转换为小写
参数	src 字符串原指针
返回值	-

字符串全等

函数名	px_bool px_strequ(px_char *src,px_char *dst);
说明	字符串全等
参数	src 字符串指针 dst 字符串指针
返回值	若两个字符串全等,返回 PX_TRUE,否者返回 PX_FALSE

函数名	px_bool px_strequ2(px_char *src,px_char *dst);
说明	字符串全等(忽略大小写)
参数	src 字符串指针 dst 字符串指针
返回值	若两个字符串全等(忽略大小写),返回 PX_TRUE,否者返回 PX_FALSE

字符串是否是数字

函数名	px_bool px_strIsNumeric(px_char *src);
说明	字符串是否是数字
参数	src 字符串指针
返回值	若字符串是数字,返回 PX_TRUE,否者返回 PX_FALSE

字符串是否是小数

函数名	px_bool px_strIsFloat(px_char *src);
说明	字符串是否是小数
参数	src 字符串指针
返回值	若字符串是小数,返回 PX_TRUE,否者返回 PX_FALSE

字符串是否是整数

函数名	px_bool px_strIsInt(px_char *src);
说明	字符串是否是整数
参数	src 字符串指针
返回值	若字符串是整数,返回 PX_TRUE,否者返回 PX_FALSE

字符串查找字符

函数名	px_char *PX_strchr(const char *s,int ch);
说明	字符串查找字符
参数	s 查找字符串,ch 字符
返回值	如果找到返回首指针,否者返回 PX_NULL

字符串查找字符串

函数名	char* PX_strstr(const char* dest, const char* src);
说明	字符串查找字符串
参数	dest 待比较字符串 src 比对字符串
返回值	如果找到返回首指针,否者返回 PX_NULL

字符串格式化

函数名	px_int px_sprintf1(px_char *_out_str,px_int str_size,px_char fmt[], px_stringformat _1) px_int px_sprintf2(px_char *_out_str,px_int str_size,px_char fmt[], px_stringformat _1, px_stringformat _2) px_int px_sprintf2(px_char *_out_str,px_int str_size,px_char fmt[], px_stringformat _1, px_stringformat _2, px_stringformat _3)
说明	字符串格式化函数

参数	_out_str 输出字符串指针 Fmt 格式化输出字符串 使用%1 %2 %3....来表示需要格式化的字符串 当需要指定浮点精度时,可以使用%1.x 的格式来表述,其中 x 为拓展精度 _1,_2,_3.....参数,数量和 px_sprintf 后的数字对应,用于对应%1 %2...对应格 式化字符串 px_stringformat 可以用 PX_STRING_FORMAT_INT PX_STRING_FORMAT_FLOAT PX_STRING_FORMAT_STRING 函数,进行构造分别用于表述一个整数,浮点数,字符串
返回值	-

*取得文件路径的文件名

函数名	px_void PX_FileGetName(const px_char filefullName[],px_char _out[],px_int outSize);
说明	取得文件路径的文件名
参数	filefullName 路径 out 输出文件名 outsize 输出数组长度
返回值	

*取得文件路径的路径

函数名	px_void PX_FileGetPath(const px_char filefullName[],px_char _out[],px_int outSize);
说明	取得文件路径的路径
参数	filefullName 路径 out 输出路径 outsize 输出数组长度
返回值	

*取得文件路径的后缀名

函数名	px_void PX_FileGetExt(const px_char filefullName[],px_char _out[],px_int
-----	--

	outSize);
说明	取得文件路径的后缀名
参数	filefullName 路径 out 输出路径 outsize 输出数组长度
返回值	

转换

字节类型转十六进制字符串

函数名	px_void PX_BufferToHexString(px_byte data[],px_int size,px_char hex_str[]);
说明	数据类型转十六进制字符串类型
参数	data 数据指针,size 为数据长度,hex_str 字符串指针长度为 size*2+1,
返回值	

十六进制字符串转字节类型

函数名	px_bool PX_HexStringToBuffer(const px_char hex_str[],px_byte data[])
说明	十六进制字符串转数据类型
参数	hex_str 字符串指针,data 数据指针,长度是 hex_str 字符串长度的一半
返回值	成功返回 PX_TRUE,失败返回 PX_FALSE;

十六进制字符转数字

函数名	px_uint PX_htoi(char*)
说明	十六进制字符串转 px_uint
参数	字符串指针
返回值	返回转换值

字符转数字

函数名	px_int PX_atoi(char*)
说明	字符串转 px_int
参数	字符串指针
返回值	返回转换值

函数名	px_float PX_atof(char*)
说明	字符串转 px_float
参数	字符串指针
返回值	返回转换值

字符串转 IPv4 地址

函数名	px_dword PX_inet_addr(px_char i[]);
说明	字符串转 IPv4 地址
参数	l 字符串 ip 地址
返回值	转换后的双字

IPv4 地址转字符串

函数名	px_char* PX_inet_ntoa(px_dword ipv4);
说明	IPv4 转字符串地址
参数	Ipv4 双字 ip 地址
返回值	转换后的字符串

数字转字符串

整数

函数名	px_int *PX_itoa(int num,char *str,px_int size,int radix)
说明	数字转字符串
参数	num 数字

	str 结果字符串指针 size 缓存大小 radix 进制
返回值	返回转换长度

函数名	PX_RETRUN_STRING PX_itos(px_int num,px_int radix);
说明	数字转字符串
参数	num 数字 radix 进制
返回值	返回一个字符串结构体,访问其 data 成员可直接取得转换字符串

小数

函数名	px_int *PX_ftoa(float num,char *str,px_int size,px_int precision)
说明	数字转字符串
参数	num 数字 str 结果字符串指针 size 缓存大小 precision 精度长度
返回值	返回转换长度

函数名	PX_RETRUN_STRING PX_ftos(float f, int precision);
说明	数字转字符串
参数	num 数字 precision 精度长度
返回值	返回一个字符串结构体,访问其 data 成员可直接取得转换字符串

弧度角度转换

函数名	PX_RadianToAngle(angle)
说明	弧度转换为角度
参数	angle 角度
返回值	角度

函数名	PX_AngleToRadian(radian)
说明	角度转弧度
参数	radian 弧度
返回值	弧度

字节序

主机字节序转网络字节序

函数名	px_dword PX_htonl(px_dword h); px_word PX_htons(px_word h);
说明	主机字节序转网络字节序
参数	h:dword 或 word 类型值
返回值	转换后的字节序

网络字节序转主机字节序

函数名	px_dword PX_ntohl(px_dword n); px_word PX_ntohs(px_word n);
说明	网络字节序转主机字节序
参数	h:dword 或 word 类型值
返回值	转换后的字节序

数学

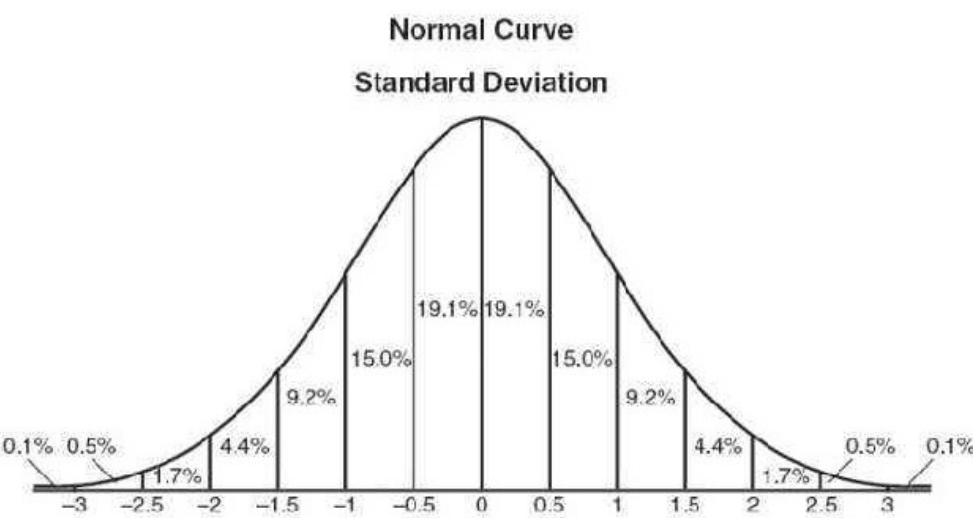
素数模乘同余发生器

函数名	px_void PX_srand(px_uint64 seed);
说明	初始化一个素数模乘同余发生器的随机数种子
参数	seed 种子,如果未调用该函数,将会以默认值对其初始化
返回值	-

函数名	px_int PX_rand();
说明	返回一个 0~PX RAND_MAX-1 之间的随机数
参数	
返回值	返回随机数,范围为 0~PX RAND_MAX-1

函数名	px_uint32 PX_randEx(px_uint64 seed)
说明	返回一个 0~PX RAND_MAX-1 之间的随机数以 seed 作为运算种子
参数	
返回值	返回随机数,范围为 0~PX RAND_MAX-1

高斯随机数发生器



函数名	px_int PX_GaussRand();
说明	返回一个符合高斯分布的随机数
参数	
返回值	返回高斯随机数,见上图分布

对数函数

函数名	px_double PX_ln(px_double __x) px_double PX_log(px_double __x)
说明	求一个数 e 为底对数函数
参数	__x 这个数的值
返回值	

函数名	px_double PX_lg(px_double __x) px_double PX_log10(px_double __x)
-----	---

说明	求一个数 10 为底对数函数
参数	__x 这个数的值
返回值	

三角函数

函数名	px_double PX_sind(px_double radian); px_double PX_cosd(px_double radian);
说明	px_double PX_sind(px_double radian); 弧度制求正弦 px_double PX_cosd(px_double radian); 弧度制求余弦 该函数使用泰勒展开

函数名	px_float PX_sin_radian(px_float radian); px_float PX_cos_radian(px_float radian); px_float PX_tan_radian(px_float radian); px_float PX_sin_angle(px_float angle); px_float PX_cos_angle(px_float angle); px_float PX_tan_angle(px_float angle); px_double PX_atan(px_double x); px_double PX_atan2(px_double y, px_double x); px_double PX_asin(px_double x); px_double PX_acos(px_double x);
说明	px_float PX_sin_radian(px_float radian); 弧度制求正弦 px_float PX_cos_radian(px_float radian); 弧度制求余弦 px_float PX_tan_radian(px_float radian); 弧度制求正切 px_float PX_sin_angle(px_float angle); 角度制求正弦 px_float PX_cos_angle(px_float angle); 角度制求余弦 px_float PX_tan_angle(px_float angle); 角度制求正切 px_double PX_atan(px_double x); 弧度制求反正切 px_double PX_atan2(px_double y, px_double x); 弧度制 atan2 px_double PX_asin(px_double x); 弧度制 arcsin px_double PX_acos(px_double x); 弧度制 arccos

平方根

函数名	float PX_Sqrt(float number)
-----	-------------------------------

说明	求一个数的平方根
参数	Number 需要求平方根的数
返回值	返回这个数的平方根 (这个数必须是正数)

函数名	px_double PX_SqrtD(px_double number)
说明	求一个数更高精度的平方根
参数	Number 需要求平方根的数
返回值	返回这个数的平方根 (这个数必须是正数)

函数名	float PX_SqrtRec(float number)
说明	求一个数平方根的倒数
参数	Number 需要求平方根的倒数
返回值	返回这个数的平方根的倒数 (这个数必须是正数)

方差

函数名	px_double PX_Variance(px_double x[],px_int n)
说明	计算一个序列的方差
参数	x 序列数据 n 元素个数
返回值	返回这个矩阵计算方差

协方差

函数名	px_double PX_Covariance(px_double x[],px_double y[],px_int n);
说明	计算一个矩阵的协方差
参数	x, y 两数据 n 数据长度
返回值	返回这两个数据协计算方差

求绝对值

函数名	PX_ABS(x)
说明	宏函数求绝对值

参数	值
返回值	返回绝对值

tanh

函数名	px_double PX_tanh(px_double x)
说明	tanh 函数
参数	x
返回值	tanh 计算结果

sigmoid

函数名	px_double PX_sigmoid(px_double x)
说明	sigmoid 函数
参数	x
返回值	sigmoid 计算结果

ReLU

函数名	px_double PX_sigmoid(px_double x)
说明	ReLU 函数
参数	x
返回值	ReLU 计算结果

四舍五入

函数名	PX_APO(x)
说明	宏函数小数点后四舍五入求整数
参数	小数
返回值	返回四舍五入的整数

幂

函数名	px_int PX_pow_ii(px_int i,px_int n);
说明	整数整幂
参数	i 底数 n 次幂
返回值	i 的 n 幂

函数名	px_double PX_pow_ff(double num,double m);
说明	幂
参数	num 底数 m 次幂
返回值	num 的 m 幂

复数

函数名	px_complex PX_complexBuild(px_float re,px_float im);
说明	构造一个复数
参数	re 为实部,im 为虚部
返回值	构造的复数结构

函数名	px_complex PX_complexAdd(px_complex a,px_complex b);
说明	复数相加
参数	a,b 需要相加的两复数
返回值	结果

函数名	px_complex PX_complexMult(px_complex a,px_complex b);
说明	复数相乘
参数	a,b 需要相乘的两复数
返回值	结果

函数名	px_double PX_complexMod(px_complex a)
说明	复数的模
参数	需要求解的复数
返回值	结果

函数名	px_complex PX_complexLog(px_complex a);
-----	---

说明	复数的 e 的对数
参数	需要求解的复数
返回值	结果

函数名	px_complex PX_complexExp(px_complex a);
说明	复数的 e 的次数
参数	需要求解的复数
返回值	结果
函数名	px_complex PX_complexSin(px_complex a);
说明	复数的 sin 值
参数	需要求解的复数
返回值	结果

矩阵

Px_matrix 结构体，声明为

```

union {
    struct {
        px_float    _11, _12, _13, _14;
        px_float    _21, _22, _23, _24;
        px_float    _31, _32, _33, _34;
        px_float    _41, _42, _43, _44;
    };
    px_float m[4][4];
}

```

是一个 4x4 的矩阵

函数名	void PX_MatrixZero(px_matrix *Mat);
说明	构造一个 0 矩阵
参数	Mat 指向该矩阵的指针

函数名	void PX_MatrixIdentity(px_matrix *Mat);
说明	构造一个单位矩阵
参数	Mat 指向该矩阵的指针

函数名	px_matrix PX_MatrixMultiply(px_matrix Mat1,px_matrix Mat2);
说明	矩阵相乘
参数	Mat1 ， Mat2 需要相乘的两个矩阵
返回值	返回矩阵相乘结果矩阵

函数名	px_matrix PX_MatrixAdd(px_matrix Mat1,px_matrix Mat2);
说明	矩阵相加
参数	Mat1 , Mat2 需要相加的两个矩阵
返回值	返回矩阵相加结果矩阵

函数名	px_matrix PX_MatrixSub(px_matrix Mat1,px_matrix Mat2);
说明	矩阵相减
参数	Mat1 , Mat2 需要相减的两个矩阵
返回值	返回矩阵相减结果矩阵

函数名	px_bool PX_MatrixEqual(px_matrix Mat1,px_matrix Mat2);
说明	判断两个矩阵是否相等
参数	Mat1 , Mat2 需要判断的两个矩阵
返回值	如果相等返回 PX_TRUE,否者返回 PX_FALSE

函数名	px_void PX_MatrixTranslation(px_matrix *mat,float x,float y,float z);
说明	构造平移矩阵
参数	Mat 指向该矩阵的指针 X,Y,Z 平移向量

函数名	px_void PX_MatrixRotateX(px_matrix *mat,float Angle);
说明	构造 x 轴旋转矩阵
参数	Mat 指向该矩阵的指针 Angle 旋转角度 (角度制)

函数名	px_void PX_MatrixRotateY(px_matrix *mat,float Angle);
说明	构造 y 轴旋转矩阵
参数	Mat 指向该矩阵的指针 Angle 旋转角度 (角度制)

函数名	px_void PX_MatrixRotateZ(px_matrix *mat,float Angle);
说明	构造 z 轴旋转矩阵
参数	Mat 指向该矩阵的指针 Angle 旋转角度 (角度制)

函数名	px_void PX_MatrixRotateXRadian (px_matrix *mat,float rad);
说明	构造 x 轴旋转矩阵
参数	Mat 指向该矩阵的指针

	rad 旋转角度 (弧度制)
--	------------------

函数名	px_void PX_MatrixRotateYRadian (px_matrix *mat,float rad);
说明	构造 Y 轴旋转矩阵
参数	Mat 指向该矩阵的指针 rad 旋转角度 (弧度制)

函数名	px_void PX_MatrixRotateZRadian(px_matrix *mat,float rad);
说明	构造 Z 轴旋转矩阵
参数	Mat 指向该矩阵的指针 rad 旋转角度 (弧度制)

函数名	px_void PX_MatrixScale(px_matrix *mat,float x,float y,float z);
说明	构造缩放矩阵
参数	Mat 指向该矩阵的指针 X,Y,Z 缩放方向

函数名	px_bool PX_MatrixInverse(px_matrix *mat);
说明	矩阵求逆
参数	Mat 指向该矩阵的指针 如果求逆成功 , 则 mat 为逆矩阵

函数名	px_void PX_MatrixTranspose(px_matrix *matrix);
说明	矩阵转置
参数	Marixt 为需要转置的矩阵指针

函数名
说明
参数

点/向量

函数名	px_point2D PX_Point2DRotate(px_point2D p,px_float angle); px_point PX_PointRotate(px_point p,px_float angle);
-----	--

说明	返回一个点绕原点顺时针旋转角度后的点
参数	P 绕原点旋转的参考点 angle 旋转角度
返回值	返回结果点

函数名	px_point PX_PointAdd(px_point p1,px_point p2); px_point2D PX_Point2DAdd(px_point2D p1,px_point2D p2); px_point4D PX_Point4DAdd(px_point4D p1,px_point4D p2);
说明	向量相加
参数	P1,p2 需要相加的两个向量
返回值	返回结果向量

函数名	px_point PX_PointSub(px_point p1,px_point p2); px_point2D PX_Point2DSub(px_point2D p1,px_point2D p2); px_point4D PX_Point4DSub(px_point4D p1,px_point4D p2);
说明	向量相减
参数	P1,p2 需要相减的两个向量
返回值	返回结果向量

函数名	px_point PX_PointMul(px_point p1,px_float m); px_point2D PX_Point2DMul(px_point2D p1,px_float m); px_point4D PX_Point4DMul(px_point4D p1,px_float m);
说明	向量缩放
参数	P1 缩放向量 M 倍数
返回值	返回结果向量

函数名	px_point PX_PointDiv(px_point p1,px_float m); px_point2D PX_Point2DDiv(px_point2D p1,px_float m); px_point4D PX_Point4DDiv(px_point4D p1,px_float m);
说明	向量缩放
参数	P1 缩放向量 M 除以倍数
返回值	返回结果向量

函数名	px_float PX_PointDot(px_point p1,px_point p2); px_float PX_Point2DDot(px_point2D p1,px_point2D p2); px_float PX_Point4DDot(px_point4D p1,px_point4D p2);
说明	向量点乘
参数	P1,p2 两点乘向量
返回值	返回结果

函数名	px_point PX_PointCross(px_point p1,px_point p2); px_point2D PX_Point2DCross(px_point2D p1,px_point2D p2); px_point4D PX_Point4DCross(px_point4D p1,px_point4D p2);
说明	向量叉乘
参数	P1,P2 两叉乘向量
返回值	返回叉乘结果

函数名	px_float PX_PointMod(px_point p); px_float PX_Point2DMod(px_point2D p); px_float PX_Point4DMod(px_point4D p);
说明	向量的模
参数	P 该向量
返回值	返回该向量的模

函数名	px_float PX_PointSquare(px_point p);
说明	向量模的平方
参数	P 该向量
返回值	返回该向量模的平方

函数名	px_float PX_PointDistance(px_point p1,px_point p2);
说明	求两点间距离
参数	P1 p2 两点
返回值	返回两点间距离

函数名	px_point PX_PointNormalization(px_point p); px_point2D PX_Point2DNormalization(px_point p); px_point4D PX_Point4DNormalization(px_point p);
说明	求单位向量(归一化向量)
参数	P 该向量
返回值	返回该向量模的单位向量

函数名	px_point PX_PointMulMatrix(px_point p,px_matrix mat) px_point2D PX_Point2DMulMatrix(px_point p,px_matrix mat)
说明	点乘矩阵
参数	P 该向量，mat 变换矩阵
返回值	返回变换后的点

函数名	px_float PX_Point_sin (px_point p)
说明	求该向量与 x 轴夹角 sin 值

参数	P 该向量
返回值	Sin 值

函数名	px_float PX_Point_cos (px_point p)
说明	求该向量与 x 轴夹角 cos 值
参数	P 该向量
返回值	cos 值

函数名	px_point PX_PointReflectX(px_point vector_refer,px_point respoint)
说明	参照向量关于 x 轴夹角对一个点做相应旋转
参数	vector_refer 向量 respoint 原点
返回值	旋转后的点

函数名	px_point PX_PointInverse(px_point p1)
说明	向量方向取反
参数	P1 需要取反的向量
返回值	取反的向量

交叉碰撞

函数名	px_bool PX_isLineCrossRect(px_point p1,px_point p2,px_rect rect,px_point *cp1,px_point *cp2);
说明	检测一条线与矩形是否相交
参数	P1,p2 这条线的两个点 Rect 矩形范围 Cp1 如果有相交,输出交点 1 Cp2 如果有相交,输出交点 2
返回值	如果有交叉返回 PX_TRUE,否者返回 PX_FALSE

函数名	px_bool PX_isRectCrossRect(px_rect rect1,px_rect rect2);
说明	检测两个矩形是否有交叉碰撞
参数	Rect1 Rect2 两矩形范围
返回值	如果有交叉返回 PX_TRUE,否者返回 PX_FALSE

函数名	px_bool PX_isRectCrossCircle(px_rect rect1,px_point center,px_float
-----	---

	radius);
说明	检测一个矩形和一个圆是否有交叉碰撞
参数	Rect1 矩形 Center 圆心 Radius 圆半径
返回值	如果有交叉返回 PX_TRUE,否者返回 PX_FALSE

函数名	px_bool PX_isCircleCrossCircle(px_point center1,px_float radius1,px_point center2,px_float radius2);
说明	检测两个圆是否有交叉碰撞
参数	Center1 Center2 圆心 1,2 Radius1 Radius2 圆 1,2 半径
返回值	如果有交叉返回 PX_TRUE,否者返回 PX_FALSE

快速排序(由小到大)

函数名	void PX_Quicksort_MinToMax(PX_QuickSortAtom array[], px_int left, px_int right);
说明	对特定结构体进行由小到大排序
参数	typedef struct { px_int power;//决定排序的权值 px_void *pData; }PX_QuickSortAtom; array 排序序列 left 开始索引 right 结束索引

快速排序(由大到小)

函数名	void PX_Quicksort_MaxToMin(PX_QuickSortAtom array[], px_int left, px_int right);
说明	对特定结构体进行由大到小排序
参数	typedef struct { px_int power;//决定排序的权值 px_void *pData;

	<pre>}PX_QuickSortAtom; array 排序序列 left 开始索引 right 结束索引</pre>
--	--

矩形

```
px_rect  
typedef struct _px_rect  
{  
    px_float x,y,width,height;  
}px_rect;
```

x,y 为矩形左上角坐标，width 与 height 为宽高

函数名	<code>px_rect PX_RECT(px_float x,px_float y,px_float width,px_float height);</code>
说明	构造一个矩形
参数	x,y 左上角坐标 width,height 矩形的宽度和高度
返回值	返回构造的矩形结构体

函数名	<code>px_rect PX_RECTPOINT2(px_point p1,px_point p2);</code>
说明	用两个点构造一个矩形
参数	p1,p2 构造的两点
返回值	返回构造的矩形结构体

函数名	<code>px_bool PX_isPointInRect(px_point p,px_rect rect)</code>
说明	判断点是否在矩形内
参数	p 该点，rect 为该矩形
返回值	若在矩形内返回 PX_TRUE 否者返回 PX_FALSE

函数名	<code>px_bool PX_isRectInRect(px_rect rect1,px_rect rect2)</code>
说明	判断是两个矩形是否相交
参数	Rect1 矩形 1,rect2 矩形 2
返回值	若在矩形内返回 PX_TRUE 否者返回 PX_FALSE

CRC 校验 (CRC32)

函数名	px_uint32 PX_crc32(px_byte *buffer, px_uint size);
说明	对一段缓存区进行 CRC32 校验
参数	Buffer 缓存区 Size 缓存区大小
返回值	CRC32 校验值

CRC 校验 (CRC16)

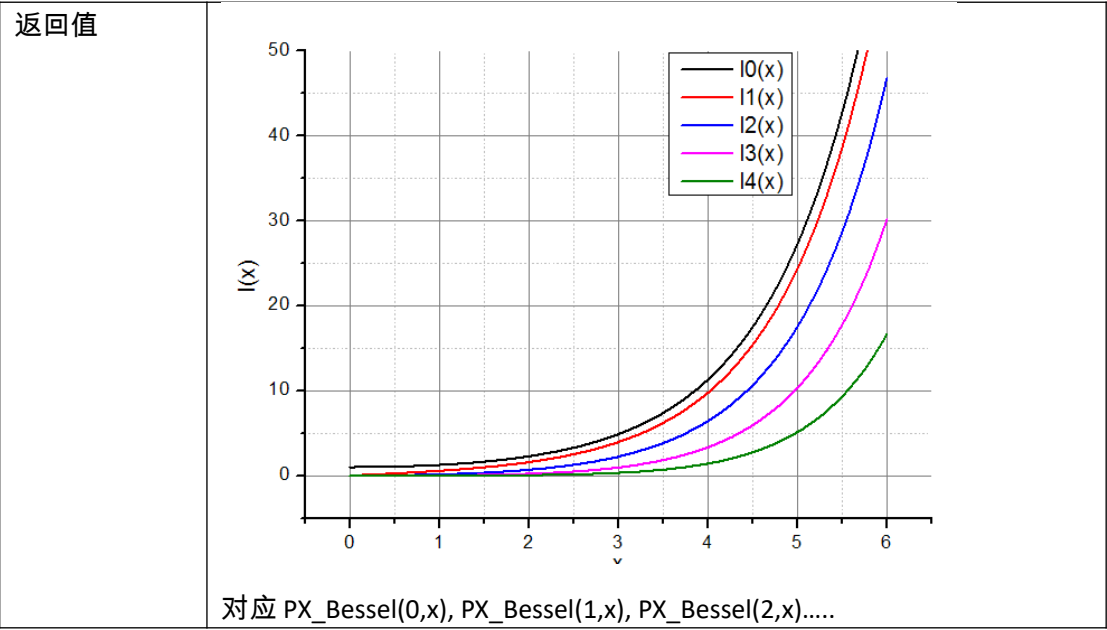
函数名	px_uint16 PX_crc16(px_byte *buffer, px_uint size);
说明	对一段缓存区进行 CRC16 校验
参数	Buffer 缓存区 Size 缓存区大小
返回值	CRC32 校验值

累加和校验 (Sum32)

函数名	px_uint32 PX_sum32(px_byte *buffer, px_uint size);
说明	对一段缓存区进行累加和校验
参数	Buffer 缓存区 Size 缓存区大小
返回值	累加和校验值

第一类修正贝塞尔函数

函数名	px_double PX_Bessel(int n,double x);
说明	第一类修正贝塞尔函数
参数	n,x



通讯链接器(io 虚拟层)

通讯链接器(PX_Linker)是 PainterEngine 默认的对通讯协议/IO 的虚拟层,它可以被解释为 TCP,UDP,串口,并口,文件读写...等通讯协议的 IO 统一及虚拟

初始化链接器

函数名	px_void PX_LinkerInitialize(PX_Linker *linker,PX_LinkerRead read,px_void *readUserPtr,PX_LinkerWrite write,px_void *writeUserPtr);
说明	初始化链接器
参数	linker 链接器结构体 read 读函数(需要自行实现) readuserptr 读函数传递指针 write 写函数(需要自行实现) writeuserptr 写函数传递指针 *注意,对于有连接协议的抽象,在读写函数中必须自行维护连接的可用性
返回值	-

链接器读

函数名	px_int PX_LinkerRead(PX_Linker *linker,px_void *data,px_int datasize);
-----	--

说明	链接器读
参数	linker 链接器结构体 data 读缓存 datasize 读缓存大小
返回值	成功读字节数

链接器写

函数名	px_int PX_LinkWrite(PX_Linker *linker,px_void *data,px_int datasize);
说明	链接器写
参数	linker 链接器结构体 data 写缓存 datasize 写缓存大小
返回值	成功写字节数

数据结构

取得结构体偏移量

函数名	PX_STRUCT_OFFSET(t,m)
说明	取得结构体成员 m 对于结构体的偏移量
参数	t 结构体类型 m 成员名
返回值	一个 px_uint 类型,为 m 在该结构体的偏移量

字母

函数名	px_void PX_CharIsNumeric(px_char ch)
说明	判断字母是否是数字
参数	ch 字母
返回值	若是返回 PX_TRUE,若不是返回 PX_FALSE

链表

PainterEngine 提供链表模板函数 px_list，对该链表使用前都要改由 PX_ListInitialize 进行初始化并在使用完毕后使用 PX_ListFree 释放内存

初始化

函数名	px_void PX_ListInitialize (px_list *list,px_memorypool *mp);
说明	对链表结构进行初始化
参数	list 需要初始化的结构指针 mp 链表部署内存池指针

插入数据

函数名	px_void PX_ListPush(px_list *list,px_void *data,px_int size);
说明	插入一个数据到链表
参数	list 结构指针 data 数据指针 size 数据大小

删除数据

函数名	px_bool PX_ListPop(px_list *list,px_list_node *node);
说明	将一个节点为 node 的数据从链表中删除
参数	list 结构指针 node 链表节点
返回值	若成功删除返回 PX_TRUE，否者为 PX_FALSE

函数名	px_bool PX_ListErase(px_list *list,px_int i);
说明	将一个节点为 node 的数据从链表中删除
参数	list 结构指针 i 节点索引

返回值	若成功删除返回 PX_TRUE , 否者为 PX_FALSE
-----	--------------------------------

取得链表节点

函数名	px_list_node* PX_ListNodeAt(px_list *list,px_int index);
说明	取得链表当前节点指针
参数	list 结构指针 index 索引
返回值	索引节点,如果索引不合法返回 PX_NULL

取得下一个链表节点

函数名	px_list_node* PX_ListNodeNext(px_list_node* node);
说明	取得节点的下一个节点指针
参数	node 当前链表节点
返回值	下一个链表节点指针

取得链表当前节点数

函数名	px_int PX_ListSize(px_list *list);
说明	取得链表当前节点数
参数	list 结构指针
返回值	当前节点数

移动节点

函数名	px_void PX_ListMove(px_list *list,px_int index,px_int moveto);
说明	移动一个索引节点到目标索引
参数	index 要移动的节点索引 moveto 移动到的索引
返回值	

取得链表节点数

函数名	px_int PX_ListSize(px_list *list);
说明	取得链表节点数
参数	list 结构指针
返回值	链表节点数

清空链表 (释放占用内存)

函数名	px_void PX_ListClear(px_list *list);
说明	清空链表
参数	list 结构指针
返回值	-

函数名	px_void PX_ListFree(px_list *list);
说明	清空链表
参数	list 结构指针
返回值	-

取节点数据

函数名	PX_LIST_NODEDATA(x) PX_LIST_NODETDATA (t,x) ((t *) (x->pdata)) #define PX_LISTAT(t,x,i) ((t *) (PX_ListNodeAt(x,i)->pdata))
说明	取节点数据
参数	x 结构指针 *t 类型
返回值	-

DEMO

```
px_list list;  
PX_ListInitialize(&list,&mempool2);  
PX_ListPush(&list,"Hello",6);
```

```
PX_ListPush(&list, "World",6);
PX_ListPush(&list, "binar",6);

PX_ListPop(&list,PX_ListAt(&list,1));
PX_ListPop(&list,PX_ListAt(&list,1));
PX_ListPop(&list,PX_ListAt(&list,1));

for (int i=0;i<PX_ListSize(&list);i++)
{
    printf((px_char *)PX_LIST_NODEDATA(PX_ListAt(&list,i)));
}
```

容器

PainterEngine 提供容器模板函数 px_vector ，对该链表使用前都要改由 PX_VectorInitialize 进行初始化并在使用完毕后使用 PX_VectorFree 释放内存

初始化

函数名	px_void PX_VectorInitialize (px_vector *vec,px_memorypool *mp,px_int nodeSize,px_int init_size);
说明	对容器结构进行初始化
参数	vec 容器指针 mp 容器部署内存池指针 nodesize 每个元素的大小 init_size 初始容器大小(初始化分配的元素个数)

添加元素

函数名	px_bool PX_VectorPushback(px_vector *vec,px_void *data);
说明	在容器中添加一个元素
参数	vec 容器指针 data 压入数据指针
返回值	成功返回 PX_TRUE,否者返回 PX_FALSE

函数名	px_bool PX_VectorPushTo(px_vector *vec,px_void *data,px_int index);
-----	---

说明	在容器中指定位置添加一个元素
参数	vec 容器指针 data 压入数据指针 index 插入的位置索引
返回值	成功返回 PX_TRUE,否者返回 PX_FALSE

设定元素

函数名	px_bool PX_VectorSet(px_vector *vec,px_uint index,px_void *data);
说明	在容器中设定对应索引的元素,注意,如果索引小于分配容量,改容器将会重新分配内存空间以设定元素,同时索引小于使用大小,容器也会设定对应使用大小以适应设定索引
参数	vec 容器指针 index 索引 data 压入数据指针
返回值	成功返回 PX_TRUE,否者返回 PX_FALSE

删除元素

函数名	px_bool PX_VectorErase(px_vector *vec,px_int index);
说明	在容器中删除一个元素
参数	vec 容器指针 index 需要删除的索引
返回值	若成功返回 PX_TRUE , 否者 PX_FALSE

清空容器(不释放预留内存)

函数名	px_void PX_VectorClear(px_vector *vec);
说明	清空容器
参数	vec 容器指针
返回值	-

取容器数据

函数名	PX_VectorAt(vec,i)
说明	取容器数据
参数	vec 容器指针 I 索引
返回值	-

释放容器(释放预留内存)

函数名	PX_VectorFree(x)
说明	释放容器,同时释放内存,若需要重新使用该容器必须对其重新初始化
参数	x 容器指针
返回值	-

拷贝容器

函数名	px_bool PX_VectorCopy(px_vector *destvec,px_vector *resvec);
说明	拷贝容器
参数	destvec 目标容器 resvec 原容器
返回值	成功返回 PX_TRUE,否者 PX_FALSE

重置容器的大小

函数名	px_bool PX_VectorResize(px_vector *vec,px_int size);
说明	重置容器的大小
参数	vec 目标容器 size 重置大小 注意,重置大小的容器数据将直接丢失
返回值	成功返回 PX_TRUE,否者 PX_FALSE

字符串映射表

初始化映射表

函数名	px_bool PX_MapInitialize(px_memorypool *mp,px_map *hashmap);
说明	初始化一个映射表
参数	mp,内存池 hashmap 映射表
返回值	如果成功返回 PX_TRUE,否者 PX_FALSE

插入元素

函数名	PX_HASHMAP_RETURN PX_MapPut (px_map * m, px_char* key, px_void * value);
说明	插入一个节点到表中
参数	m 哈希表 key 映射字符串 value 值
返回值	如果成功返回 PX_HASHMAP_RETURN_OK

函数名	PX_HASHMAP_RETURN PX_MapPutInt (px_map * m, px_char* key, px_int value);
说明	插入一个 int 节点到表中
参数	m 哈希表 key 映射字符串 value 值
返回值	如果成功返回 PX_HASHMAP_RETURN_OK

函数名	PX_HASHMAP_RETURN PX_MapPutFloat (px_map * m, px_char* key, px_int value);
说明	插入一个 float 节点到表中
参数	m 哈希表 key 映射字符串 value 值
返回值	如果成功返回 PX_HASHMAP_RETURN_OK

查找元素

函数名	px_void *PX_MapGet(px_map * m, px_char* stringkey);
说明	查找哈希表对应 key 的元素
参数	m 哈希表 key 映射字符串
返回值	对于返回类型

函数名	px_bool PX_MapGetInt(px_map * m, px_char* stringkey,px_int *v);
说明	查找哈希表对应 key 的 int 元素
参数	m 哈希表 key 映射字符串 v 返回赋值指针
返回值	如果成功返回 PX_TRUE,否则返回 PX_FALSE

函数名	px_float PX_MapGetFloat(px_map * m, px_char* stringkey);
说明	查找哈希表对应 key 的 float 元素
参数	m 哈希表 key 映射字符串 v 返回赋值指针
返回值	如果成功返回 PX_TRUE,否则返回 PX_FALSE

删除元素

函数名	PX_HASHMAP_RETURN PX_MapErase (px_map * m, px_char* key);
说明	删除对应 key 的元素
参数	m 哈希表 key 映射字符串
返回值	如果成功返回 PX_HASHMAP_RETURN_OK

释放 map

函数名	px_void PX_MapFree(px_map * m)
说明	删除对应 map 所有元素并释放内存资源
参数	m 哈希表
返回值	-

自适应内存

PainterEngine Core 提供自动内存类型 px_memory,注意,px_memory 使用前,都应该使用 PX_MemoryInit 进行初始化,并在使用完毕后使用 PX_MemoryFree 释放内存

初始化

函数名	px_void PX_MemoryInitialize(px_memorypool *mp,px_memory *memory);
说明	对自适应内存结构进行初始化
参数	mp 容器部署内存池指针 memory 内存指针

拼接内存

函数名	px_bool PX_MemoryCat(px_memory *memory,px_void *buffer,px_int size);
说明	对自适应内存进行数据拼接
参数	memory 内存指针 buffer 需要拼接的内存指针 size 拼接大小
返回值	若成功返回 PX_TRUE,否者返回 PX_FALSE

重置内存分配大小

函数名	px_void PX_MemoryResize(px_memory *memory, px_int size);
说明	重置内存分配大小
参数	memory 内存指针 size 新的分配大小
返回值	若成功返回 PX_TRUE,否者返回 PX_FALSE

释放内存

函数名	px_void PX_MemoryFree(px_memory *memory);
说明	释放内存
参数	memory 内存指针

取内存数据

函数名	px_byte * PX_MemoryData(px_memory *memory);
说明	取内存数据指针
参数	memory 内存指针
返回值	内存数据指针

内存查找

函数名	px_byte * PX_MemoryFind(px_memory *memory,px_void *buffer,px_int size);
说明	在内存中查找匹配内存
参数	memory 内存指针 buffer 匹配内存缓存 size 匹配内存缓存大小
返回值	如果找到返回该内存数据指针,否则返回 PX_NULL

内存拷贝

函数名	px_void PX_MemoryCopy(px_memory *memory,px_void *buffer,px_uint startoffset,px_int size);
说明	拷贝内存到目标内存结构中,注意,这个函数会依据拷贝内存大小自动调整内存部署
参数	memory 内存指针 buffer 拷贝内存缓存 startoffset 拷贝到的起始地址

	size 拷贝内存缓存大小
返回值	

内存清理

函数名	px_void PX_MemoryClear(px_memory *memory);
说明	清空内存,这个函数并不会释放占用内存空间
参数	memory 内存指针
返回值	

移除内存

函数名	px_void PX_MemoryRemove(px_memory *memory,px_int start,px_int end)
说明	移除一块内存区域,在区域之后的内存数据将自动拼接 to 移除的位置中
参数	memory 内存指针 start 开始索引 end 结束索引 *开始索引必须小于等于结束索引,结束索引不得大于等于实际已用内存大小
返回值	

内存自适应字符串

PainterEngine Core 提供自动内存调节的字符串类型 px_string,注意,px_string 的需要更多的性能开销,字符串使用前,都应该使用 PX_StringInit 进行初始化,并在使用完毕后使用 PX_StringFree 释放内存

初始化

函数名	px_bool PX_StringInitialize(px_memorypool *mp,px_string *str);
说明	对字符串结构进行初始化

参数	mp 容器部署内存池指针 str 字符串指针
返回值	成功返回 PX_TRUE, 否者 PX_FALSE

函数名	px_void PX_StringInitFromConst(px_string *str, px_char *conststr);
说明	使用一个字符串常量对字符串结构初始化, 这个字符串额外占据的内存为 0
参数	str 字符串指针 conststr 字符串常量指针

转换

转换为整数

函数名	px_int PX_StringToInteger(px_string *str);
说明	转换字符串为整数, 支持十六进制转换(以 0x 或 0X 开头)
参数	str 字符串指针
返回值	转换整数结果,

转换为小数

函数名	px_float PX_StringToFloat(px_string *str);
说明	转换字符串为浮点数
参数	str 字符串指针
返回值	转换浮点结果,

设置字符串文本

函数名	px_bool PX_StringSet(px_string *str, px_char fmt[]);
说明	设置字符串文本
参数	str 字符串指针 fmt 需要设置的文本
返回值	成功返回 PX_TRUE, 否者 PX_FALSE(内存不足)

去除首位空格

函数名	px_void PX_StringTrim(px_string *str);
说明	去除首位空格,
参数	str 字符串指针
返回值	-

取字符串长度

函数名	px_int PX_StringLen(px_string *str);
说明	取得字符串长度
参数	str 字符串指针
返回值	字符长度

字符串拼接

函数名	px_void PX_StringCatChar(px_string *str,px_char ch);
说明	将 ch 拼接到字符串 str 的尾部
参数	str 字符串指针 ch 拼接字母
返回值	-
函数名	px_void PX_StringCat(px_string *str,px_char *str2);
说明	将 str2 拼接到字符串 str 的尾部
参数	str 字符串指针 str2 拼接字符串指针
返回值	-

释放字符串

函数名	px_void PX_StringFree(px_string *str);
说明	释放字符串内存
参数	str 字符串指针

清理字符串

函数名	px_void PX_StringClear(px_string *str);
说明	清理字符串文本(不释放内存)
参数	str 字符串指针

插入字符

函数名	px_bool PX_StringInsertChar(px_string *str,px_int index,px_char ch);
说明	将一个字符插入到字符串中
参数	str 字符串指针 index 插入的位置索引 ch 要插入的字符

删除字符

函数名	px_bool PX_StringRemoveChar(px_string *str,px_int index);
说明	将一个字符到从字符串中删除
参数	str 字符串指针 index 插入的位置索引 ch 要插入的字符

裁剪字符串

函数名	px_void PX_StringTrimLeft(px_string *str,px_int leftCount);
说明	裁剪字符串左边的字符
参数	str 字符串指针 leftCount 裁去左边的字符数

函数名	px_void PX_StringTrimRight(px_string *str,px_int RightCount);
说明	裁剪字符串右边的字符
参数	str 字符串指针 RightCount 裁去左边的字符数

格式化字符串

函数名	<pre>px_bool PX_StringFormat8(px_string *str,px_char fmt[],px_stringformat _1, px_stringformat _2, px_stringformat _3, px_stringformat _4,px_stringformat _5, px_stringformat _6, px_stringformat _7, px_stringformat _8); px_bool PX_StringFormat7(px_string *str,px_char fmt[],px_stringformat _1, px_stringformat _2, px_stringformat _3, px_stringformat _4,px_stringformat _5, px_stringformat _6, px_stringformat _7); px_bool PX_StringFormat6(px_string *str,px_char fmt[],px_stringformat _1, px_stringformat _2, px_stringformat _3, px_stringformat _4,px_stringformat _5, px_stringformat _6); px_bool PX_StringFormat5(px_string *str,px_char fmt[],px_stringformat _1, px_stringformat _2, px_stringformat _3, px_stringformat _4,px_stringformat _5); px_bool PX_StringFormat4(px_string *str,px_char fmt[],px_stringformat _1, px_stringformat _2, px_stringformat _3, px_stringformat _4); px_bool PX_StringFormat3(px_string *str,px_char fmt[],px_stringformat _1, px_stringformat _2, px_stringformat _3); px_bool PX_StringFormat2(px_string *str,px_char fmt[],px_stringformat _1, px_stringformat _2); px_bool PX_StringFormat1(px_string *str,px_char fmt[],px_stringformat _1);</pre>
说明	格式化字符串
参数	参考 px_sprintf1...的实现

筛选

函数名	<pre>px_bool PX_StringIsNumeric(px_string *str);</pre>
说明	判断字符串是否为合法数字
参数	str 字符串指针
返回值	如果是返回 PX_TRUE,否者 PX_FALSE

函数名	<pre>px_bool PX_StringIsFloat(px_string *str);</pre>
说明	判断字符串是否为合法浮点数
参数	str 字符串指针
返回值	如果是返回 PX_TRUE,否者 PX_FALSE

字符串替换 1

函数名	px_void PX_StringReplaceRange(px_string *str,px_int startindex,px_int endindex,px_char *replaceto)
说明	字符串替换,将目标范围的字符串替换为 replaceto 里的字符串
参数	str 字符串指针 startindex 开始索引 endindex 结束索引 replaceto 替换成的字符串
返回值	-

字符串替换 2

函数名	px_void PX_StringReplace(px_string *str,px_char *source,px_char *replaceto);
说明	字符串替换,将 str 中包含 source 的文本都替换为 replaceto
参数	str 字符串指针 source 需要搜索的原始文本 replaceto 替换成的文本
返回值	-

几何绘制

线

函数名	px_void PX_GeoDrawLine(px_surface *psurface, px_int x0, px_int y0, px_int x1, px_int y1 ,px_int lineWidth, px_color color);
说明	绘制一个反走样线段
参数	psurface 渲染表面 x0 y0 起始点坐标 x1 y1 终点坐标 lineWidth 线宽 color 颜色

边框

函数名	px_void PX_GeoDrawBorder(px_surface *psurface, px_int left, px_int top, px_int right, px_int bottom ,px_int lineWidth,px_color color);
说明	绘制一个边框
参数	psurface 渲染表面 left top right bottom 位置描述 lineWidth 边框宽度像素 color 颜色

矩形

函数名	px_void PX_GeoDrawRect(px_surface *psurface, px_int left, px_int top, px_int right, px_int bottom,px_color color);
说明	绘制一个实心矩形
参数	psurface 渲染表面 left top right bottom 位置描述 color 颜色

实心圆

函数名	px_void PX_GeoDrawSolidCircle(px_surface *psurface, px_int x,px_int y,px_int Radius,px_color color);
说明	绘制一个反走样实心圆
参数	psurface 渲染表面 x , y 圆心 radius 半径 color 颜色

圆

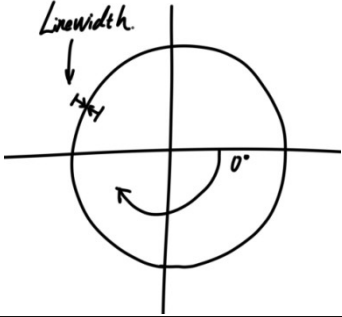
函数名	px_void PX_GeoDrawCircle(px_surface *psurface,px_int x,px_int y,px_int Radius ,pt_int lineWidth,px_color color);
说明	绘制一个反走样圆
参数	psurface 渲染表面 x , y 圆心 radius 半径 lineWidth 线宽 color 颜色

圆笔

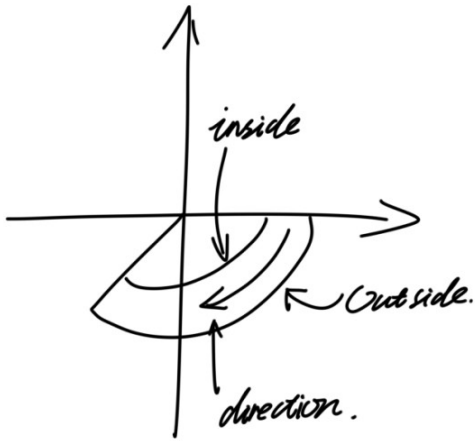
函数名	px_void PX_GeoDrawPenCircle(px_surface *psurface, px_float x,px_float y,px_float Radius,px_color color)
说明	绘制一个反走样圆,这个算法用于高质量圆笔的绘制,可用这个算法组合为线段路径
参数	psurface 渲染表面 x , y 圆心 radius 半径 lineWidth 线宽 color 颜色

环

函数名	px_void PX_GeoDrawRing(px_surface *psurface, px_int x,px_int y,px_int Radius,px_int lineWidth,px_color color,px_uint start_angle,px_uint end_angle)
说明	绘制一个反走样环
参数	psurface 渲染表面 x , y 环心 radius 半径 lineWidth 线宽 color 颜色 start_angle 起始角度

	<p>end_angle 终止角度 这个环遵循顺时针方向</p> 
--	--

扇形

函数名	<pre>px_void PX_GeoDrawSector(px_surface *psurface, px_int x,px_int y,px_int Radius_outside,px_int Radius_inside,px_color color,px_uint start_angle,px_uint end_angle);</pre>
说明	绘制一个反走样扇形
参数	<p>psurface 渲染表面 x , y 环心 radius 半径 outside 外径 inside 内径 color 颜色 start_angle 起始角度 end_angle 终止角度 这个扇形遵循顺时针方向</p> 

圆角矩形

函数名	px_void PX_GeoDrawRoundRect(px_surface *psurface, px_int left, px_int top, px_int right, px_int bottom,px_float roundRaduis,px_float linewidth,px_color color);
说明	绘制一个反走样圆角矩形
参数	psurface 渲染表面 left top right bottom 位置描述 roundRadius 圆角半径 lineWidth 线宽 color 颜色

实心圆角矩形

函数名	px_void PX_GeoDrawSolidRoundRect(px_surface *psurface, px_int left, px_int top, px_int right, px_int bottom,px_float roundRaduis,px_color color);
说明	绘制一个反走样实心圆角矩形
参数	psurface 渲染表面 left top right bottom 位置描述 roundRadius 圆角半径 color 颜色

描边路径

函数名	px_void PX_GeoDrawPath(px_surface *psurface, px_point path[],px_int pathCount,px_float linewidth,px_color color)
说明	描边路径 *注意:渲染算法使用圆点插值算法进行绘制以获得高质量的描线,alpha值因为 step 也会有所放大(约 4 倍),这个算法耗费性能资源较大,这个渲染算法应该谨慎用于实时渲染表面
参数	psurface 渲染表面 path 路径上所有的点 pathCount 点的数量 linewidth 线宽 color 颜色

三角形

函数名	px_void PX_GeoDrawTriangle(px_surface *psurface,px_point2D p0,px_point2D p1,px_point2D p2,px_color color);
说明	绘制一个反走样三角形
参数	psurface 渲染表面 p0 p1 p2 三角形三点 color 颜色

箭头

函数名	px_void PX_GeoDrawArrow(px_surface *psurface,px_point2D p0,px_point2D p1,px_float size,px_color color)
说明	绘制一个由 p0 指向 p1 的箭头
参数	psurface 渲染表面 p0 p1 箭头向量两点 size 箭头尺寸 color 颜色

绘制贝塞尔曲线

函数名	px_void PX_GeoDrawBezierCurvePoint(px_surface *rendersurface,px_point pt[],px_int pt_count,px_float t,px_float radius,px_color clr);
说明	绘制一条圆笔控制步长的由多个控制点组成的贝塞尔曲线
参数	Rendersurface 渲染表面 pt 贝塞尔控制点数组 pt_count 控制点个数 t 圆笔步长 radius 圆笔半径 clr 圆笔每点颜色

图形图像

渲染表面

颜色

函数名	px_color PX_COLOR(px_uchar a,px_uchar r,px_uchar g,px_uchar b);
说明	构造颜色
参数	A,r,g,b 颜色分量

函数名	px_void PX_ColorIncrease(px_color *color,px_uchar inc);
说明	增量颜色值
参数	Color 指向需要增量的颜色结构 Inc 增量值

函数名	px_color PX_ColorAdd(px_color color1,px_color color2);
说明	颜色相加
参数	Color1 , color2 需要相加的两个颜色
返回值	返回相加的颜色结果

函数名	px_color PX_ColorSub(px_color color1,px_color color2);
说明	颜色相减
参数	Color1 , color2 需要相减的两个颜色
返回值	返回相减的颜色结果

函数名	px_bool PX_ColorEqual(px_color color1,px_color color2);
说明	判断两个颜色是否相等
参数	Color1 , color2 需要判断的两个颜色
返回值	如果相等返回 PX_TRUE,否者返回 PX_FALSE

表面操作

函数名	px_bool PX_SurfaceCreate(px_memorypool *mp,px_uint height,px_uint width,px_surface *surface);
说明	创建一个渲染表面,它是用于描述渲染内存的结构体，仅支持 32 位 BGRA 色彩格式。
参数	Mp 内存池指针 Height 高 Width 宽 Surface 渲染表面指针
返回值	如果成功返回 PX_TRUE,否者返回 PX_FALSE

函数名	px_void PX_SurfaceFree(px_surface *psurface);
说明	释放一个渲染表面
参数	psurface 指向需要释放的渲染表面

像素操作

函数名	px_void PX_SurfaceDrawPixel(px_surface *ps,px_int x,px_int y,px_color color);
说明	绘制一个像素
参数	ps 指向表面 x , y 坐标 color 颜色
返回值	

函数名	PX_SUFACECOLOR(Surface,X,Y)
说明	查找像素颜色
参数	Surface 指向表面 X , Y 坐标
返回值	对应 px_color

函数名	px_void PX_SetPixel(px_surface *ps,px_int x,px_int y,px_color color);
说明	设置一个像素
参数	ps 指向表面

	x , y 坐标 color 颜色
返回值	

函数名	px_void PX_SurfaceSetRect(px_surface *psurface, px_int left, px_int top, px_int right, px_int bottom,px_color color)
说明	将一个渲染表面的一个矩形区域设置为某种颜色
参数	psurface 渲染表面 left top right bottom 位置描述 color 颜色
返回值	-

函数名	px_int PX_SurfaceMemorySize(px_uint width,px_uint height);
说明	预估一个表面的占用内存(实际占用依据内存池结构应适当增大)
参数	width 表面宽度 height 表面高度
返回值	-

设置渲染限制区域

函数名	px_void PX_SurfaceSetLimit(px_surface *ps,px_int limit_left,px_int limit_top,px_int limit_right,px_int limit_bottom);
说明	设置渲染的限制区域,该区域不得超出渲染表面的区域(否者函数会对其进行裁剪) 这个限制区域将会 PX_SurfaceDrawPixel 和 PX_SurfaceSetPixel 生效
参数	ps limit_left 限制区域左上角 x 坐标 limit_top 限制区域左上角 y 坐标 limit_right 限制区域右下角 x 坐标 limit_bottom 限制区域右下角 y 坐标
返回值	-

取得当前渲染限制信息

函数名	PX_SurfaceLimitInfo PX_SurfaceGetLimit(px_surface *ps);
说明	取得渲染的限制区域

参数	ps 渲染表面
返回值	-限制信息

设置当前渲染限制信息

函数名	px_void PX_SurfaceSetLimitInfo(px_surface *ps,PX_SurfaceLimitInfo info);
说明	设置渲染的限制区域
参数	ps 渲染表面 info 限制信息
返回值	

纹理

px_texture 是 PainterEngine 的纹理对象结构，建议所有的图像数据都最终加载为 px_texture 其本质上仍然是一个 surface,其指针可以和 px_surface 互换使用.

创建纹理

函数名	px_bool PX_TextureCreate(px_memorypool *mp,px_texture *tex,px_int width,px_int height);
说明	创建一个 Texture 表面
参数	mp 内存池指针 tex 输出表面指针 width 宽 height 高
返回值	若成功返回 PX_TRUE 否者 PX_FALSE

函数名	px_bool PX_TextureCreateFromMemory(px_memorypool *mp,px_void *data,px_int size,px_texture *tex);
说明	从支持的数据创建一个 Texture 表面 (如 bmp 或 TRaw)
参数	mp 内存池指针 data 数据指针 size 宽 tex 纹理格式
返回值	若成功返回 PX_TRUE 否者 PX_FALSE

缩放纹理

函数名	px_bool PX_TextureCreateScale(px_memorypool *mp,px_texture *resTexture,px_int newWidth,px_int newHeight,px_texture *out);
说明	使用窗采样缩放一个纹理并创造缩放后的纹理
参数	mp 内存池指针 resTexture 原纹理指针 newWidth newHeight 缩放后的大小 out 缩放后的纹理指针,在使用完毕后改指针必须被释放
返回值	若成功返回 PX_TRUE 否者 PX_FALSE

拷贝纹理

函数名	px_bool PX_TextureCopy(px_memorypool *mp,px_texture *resTexture,px_texture *out);
说明	深拷贝一个纹理到目标纹理中
参数	mp 内存池指针 resTexture 原纹理指针 out 拷贝到目标纹理
返回值	若成功返回 PX_TRUE 否者 PX_FALSE

渲染纹理

函数名	px_void PX_TextureRender(px_surface *psurface,px_texture *tex,px_int x,px_int y,PX_ALIGN refPoint,PX_TEXTURERENDER_BLEND *blend);
说明	渲染一个纹理到表面
参数	psurface 渲染到的表面 px_texture 需要渲染的纹理 x , y 偏移量(该坐标以纹理左上角为参照) refPoint 参考中心点，用于表示纹理绘制的相对位置 blend blend 类型结构体，用于调整绘制纹理的 alpha , hdr 值 当它为 PX_NULL 时，表示采用默认 blend 值

	<pre>typedef struct { float hdr_R; //HDR of Red float hdr_G; //HDR of Green float hdr_B; //HDR of Blue float alpha; //Blend of alpha }PX_TEXTURERENDER_BLEND;</pre>
返回值	-

函数名	px_void PX_TextureRenderPixelShader(px_surface *psurface,px_texture *tex,px_int x,px_int y ,PX_ALIGN refPoint,PX_TexturePixelShader shader,px_void *ptr);
说明	渲染一个纹理到表面,每个像素渲染将会调用 shader
参数	psurface 渲染到的表面 px_texture 需要渲染的纹理 x , y 偏移量(该坐标以纹理左上角为参照) refPoint 参考中心点，用于表示纹理绘制的相对位置 shader pixels shader 函数 ptr 传递给 PixelsShader 的指针
返回值	-

函数名	px_void PX_TextureRenderRotation(px_surface *psurface,px_texture *tex,px_int x,px_int y ,PX_ALIGN refPoint,PX_TEXTURERENDER_BLEND *blend,px_int Angle);
说明	渲染一个纹理到表面,并顺时针旋转一个角度
参数	psurface 渲染到的表面 px_texture 需要渲染的纹理 x , y 偏移量(该坐标以纹理左上角为参照) refPoint 参考中心点，用于表示纹理绘制的相对位置 blend blend 类型结构体，用于调整绘制纹理的 alpha，hdr 值 当它为 PX_NULL 时，表示采用默认 blend 值 <pre>typedef struct { float hdr_R; //HDR of Red float hdr_G; //HDR of Green float hdr_B; //HDR of Blue float alpha; //Blend of alpha }PX_TEXTURERENDER_BLEND;</pre> Angle 旋转的角度

	注意该函数使用双线性插值滤波运算,对于性能要求较高的渲染函数可以使用 PX_TextureRenderEx,以较低的显示损失获得 8 倍左右的性能
返回值	-
函数名	px_void PX_TextureRenderRotation_vector(px_surface *psurface,px_texture *tex,px_int x,px_int y ,PX_ALIGN refPoint,PX_TEXTURERENDER_BLEND *blend,px_point p_vector);
说明	渲染一个纹理到表面,并顺时针旋转一个角度
参数	<p>psurface 渲染到的表面</p> <p>px_texture 需要渲染的纹理</p> <p>x , y 偏移量(该坐标以纹理左上角为参照)</p> <p>refPoint 参考中心点 , 用于表示纹理绘制的相对位置</p> <p>blend blend 类型结构体 , 用于调整绘制纹理的 alpha , hdr 值</p> <p>当它为 PX_NULL 时 , 表示采用默认 blend 值</p> <p>typedef struct</p> <pre>{ float hdr_R; //HDR of Red float hdr_G; //HDR of Green float hdr_B; //HDR of Blue float alpha; //Blend of alpha }PX_TEXTURERENDER_BLEND;</pre> <p>P_vector 旋转向量,以 x 轴正方向为基准</p> <p>注意该函数使用双线性插值滤波运算,对于性能要求较高的渲染函数可以使用 PX_TextureRenderEx,以较低的显示损失获得 8 倍左右的性能</p>
返回值	-

函数名	px_void PX_TextureRenderRotation_sincos(px_surface *psurface,px_texture *tex,px_int x,px_int y ,PX_ALIGN refPoint,PX_TEXTURERENDER_BLEND *blend,px_float sin,px_float cos);
说明	渲染一个纹理到表面,并顺时针旋转一个角度
参数	<p>psurface 渲染到的表面</p> <p>px_texture 需要渲染的纹理</p> <p>x , y 偏移量(该坐标以纹理左上角为参照)</p> <p>refPoint 参考中心点 , 用于表示纹理绘制的相对位置</p> <p>blend blend 类型结构体 , 用于调整绘制纹理的 alpha , hdr 值</p> <p>当它为 PX_NULL 时 , 表示采用默认 blend 值</p> <p>typedef struct</p> <pre>{ float hdr_R; //HDR of Red float hdr_G; //HDR of Green float hdr_B; //HDR of Blue float alpha; //Blend of alpha }PX_TEXTURERENDER_BLEND;</pre>

	<p>Sin cos 旋转矩阵对于 sin cos 值</p> <p>注意该函数使用双线性插值滤波运算,对于性能要求较高的渲染函数可以使用 PX_TextureRenderEx,以较低的显示损失获得 8 倍左右的性能</p>
返回值	-

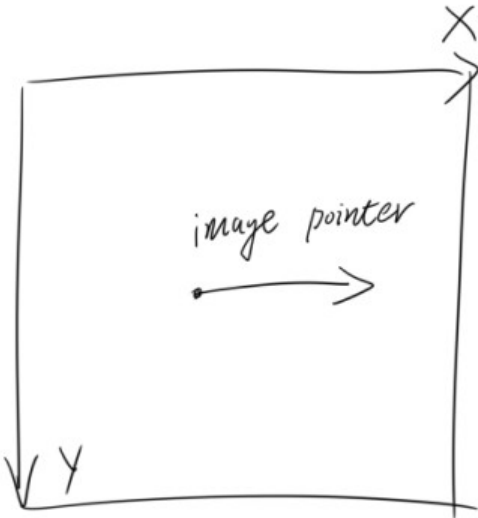
函数名	<p>px_void PX_TextureRenderEx(px_surface *psurface,px_texture *tex,px_int x,px_int y,PX_ALIGN refPoint,PX_TEXTURERENDER_BLEND *blend,px_float scale,px_float rotation);</p>
说明	<p>渲染一个纹理到表面,注意,这个渲染方式的速度不如 PX_TextureRender,但提供了纹理的缩放和旋转(同样处于效率考虑,其缩放旋转均采用单点采样,其质量不如使用纹理缩放的函数,但速度快于这两函数,对于高精度旋转和缩放动画,采用该函数可以获得一个折中的效率,内存空间与动画效果)</p>
参数	<p>psurface 渲染到的表面</p> <p>px_texture 需要渲染的纹理</p> <p>x , y 偏移量(该坐标以纹理左上角为参照)</p> <p>refPoint 参考中心点 , 用于表示纹理绘制的相对位置</p> <p>blend blend 类型结构体 , 用于调整绘制纹理的 alpha , hdr 值</p> <p>当它为 PX_NULL 时 , 表示采用默认 blend 值</p> <p>typedef struct</p> <pre>{ float hdr_R; //HDR of Red float hdr_G; //HDR of Green float hdr_B; //HDR of Blue float alpha; //Blend of alpha }PX_TEXTURERENDER_BLEND;</pre> <p>scale 缩放大小</p> <p>rotation 旋转角度</p>
返回值	-

函数名	<p>px_void PX_TextureRenderEx_sincos(px_surface *psurface,px_texture *tex,px_int x,px_int y,PX_ALIGN refPoint,PX_TEXTURERENDER_BLEND *blend,px_float scale,px_float sin,px_float cos);</p>
说明	<p>渲染一个纹理到表面,注意,这个渲染方式的速度不如 PX_TextureRender,但提供了纹理的缩放和旋转(以旋转矩阵 sin cos 形式,同样处于效率考虑,其缩放旋转均采用单点采样,其质量不如使用纹理缩放的函数,但速度快于这两函数,对于高精度旋转和缩放动画,采用该函数可以获得一个折中的效率,内存空间与动画效果)</p>
参数	<p>psurface 渲染到的表面</p> <p>px_texture 需要渲染的纹理</p> <p>x , y 偏移量(该坐标以纹理左上角为参照)</p>

	<p>refPoint 参考中心点，用于表示纹理绘制的相对位置</p> <p>blend blend 类型结构体，用于调整绘制纹理的 alpha，hdr 值</p> <p>当它为 PX_NULL 时，表示采用默认 blend 值</p> <pre>typedef struct { float hdr_R; //HDR of Red float hdr_G; //HDR of Green float hdr_B; //HDR of Blue float alpha; //Blend of alpha }PX_TEXTURERENDER_BLEND;</pre> <p>scale 缩放大小</p> <p>sin 旋转矩阵 sin 值</p> <p>cos 旋转矩阵 cos 值</p>
返回值	-

函数名	px_void PX_TextureRenderEx(px_surface *psurface,px_texture *tex,px_int x,px_int y,PX_ALIGN refPoint,PX_TEXTURERENDER_BLEND *blend,px_float scale,px_float rotation);
说明	渲染一个纹理到表面,注意,这个渲染方式的速度不如 PX_TextureRender,但提供了纹理的缩放和旋转(同样处于效率考虑,其缩放旋转均采用单点采样,其质量不如使用纹理缩放的函数,但速度快于这两函数,对于高精度旋转和缩放动画,采用该函数可以获得一个折中的效率,内存空间与动画效果)
参数	<p>psurface 渲染到的表面</p> <p>px_texture 需要渲染的纹理</p> <p>x，y 偏移量(该坐标以纹理左上角为参照)</p> <p>refPoint 参考中心点，用于表示纹理绘制的相对位置</p> <p>blend blend 类型结构体，用于调整绘制纹理的 alpha，hdr 值</p> <p>当它为 PX_NULL 时，表示采用默认 blend 值</p> <pre>typedef struct { float hdr_R; //HDR of Red float hdr_G; //HDR of Green float hdr_B; //HDR of Blue float alpha; //Blend of alpha }PX_TEXTURERENDER_BLEND;</pre> <p>scale 缩放大小</p> <p>rotation 旋转角度</p>
返回值	-

函数名	px_void PX_TextureRenderEx_vector(px_surface *psurface,px_texture *tex,px_int x,px_int y,PX_ALIGN refPoint,PX_TEXTURERENDER_BLEND
-----	---

	<code>*blend,px_float scale,px_point p_vector);</code>
说明	<p>渲染一个纹理到表面,注意,这个渲染方式的速度不如 <code>PX_TextureRender</code>,但提供了纹理的缩放和旋转(以旋转矩阵 <code>vector</code> 形式,同样处于效率考虑,其缩放旋转均采用单点采样,其质量不如使用纹理缩放的函数,但速度快于这两函数,对于高精度旋转和缩放动画,采用该函数可以获得一个折中的效率,内存空间与动画效果)</p> <p>默认的指向方向为</p> 
参数	<p><code>psurface</code> 渲染到的表面 <code>px_texture</code> 需要渲染的纹理 <code>x</code> , <code>y</code> 偏移量(该坐标以纹理左上角为参照) <code>refPoint</code> 参考中心点,用于表示纹理绘制的相对位置 <code>blend</code> <code>blend</code> 类型结构体,用于调整绘制纹理的 <code>alpha</code> , <code>hdr</code> 值 当它为 <code>PX_NULL</code> 时,表示采用默认 <code>blend</code> 值</p> <pre>typedef struct { float hdr_R; //HDR of Red float hdr_G; //HDR of Green float hdr_B; //HDR of Blue float alpha; //Blend of alpha }PX_TEXTURERENDER_BLEND;</pre> <p><code>scale</code> 缩放大小 指向方向</p>
返回值	-

函数名	<code>px_void PX_TextureRegionRender(px_surface *psurface,px_texture *resTexture,px_int x,px_int y,px_int oft_left,px_int oft_top,px_int oft_right,px_int oft_bottom,PX_ALIGN refPoint,PX_TEXTURERENDER_BLEND</code>
-----	--

	*blend)
说明	渲染部分纹理
参数	<p>psurface 渲染表面</p> <p>ptexture 纹理</p> <p>x,y 绘制参考点</p> <p>left,right,top,bottom 需要绘制的纹理位置描述</p> <p>refPoint 参考点类型的位置描述</p> <p>blend 类型结构体，用于调整绘制纹理的 alpha，hdr 值</p> <p>当它为 PX_NULL 时，表示采用默认 blend 值</p> <p>typedef struct</p> <pre>{ float hdr_R; //HDR of Red float hdr_G; //HDR of Green float hdr_B; //HDR of Blue float alpha; //Blend of alpha }PX_TEXTURERENDER_BLEND;</pre>
返回值	-

创建旋转纹理

函数名	px_bool PX_TextureCreateRotationAngle(px_memorypool *mp,px_texture *resTexture,px_float Angle,px_texture *out);
说明	将一个纹理按角度进行顺时针旋转并创建一个新纹理
参数	<p>mp 内存池指针</p> <p>resTexture 原纹理</p> <p>Angle 旋转的角度</p> <p>out 输出纹理</p>
返回值	若成功返回 PX_TRUE 否者 PX_FALSE

函数名	px_bool PX_TextureRotationAngleToTexture(px_texture *resTexture,px_float Angle,px_texture *out);
说明	将一个纹理按角度进行顺时针旋转并拷贝到目标纹理
参数	<p>resTexture 原纹理</p> <p>Angle 旋转的角度</p> <p>out 输出纹理</p>
返回值	若成功返回 PX_TRUE 否者 PX_FALSE

函数名	px_bool PX_TextureCreateRotationRadian(px_memorypool *mp,px_texture *resTexture,px_float Rad,px_texture *out);
说明	将一个纹理按弧度进行顺时针旋转并创建一个新纹理
参数	mp 内存池指针 resTexture 原纹理 Rad 旋转的弧度 out 输出纹理
返回值	若成功返回 PX_TRUE 否者 PX_FALSE

释放纹理

函数名	px_void PX_TextureFree(px_texture *tex);
说明	释放纹理内存
参数	tex 纹理指针
返回值	-

填充纹理

函数名	px_void PX_TextureFill(px_memorypool *mp,px_texture *ptexture,px_int x,px_int y,px_color test_color,px_color fill_color)
说明	填充纹理
参数	mp 运行计算内存池 ptexture 纹理 x,y 填充位置 test_color 测试颜色 fill_color 填充颜色
返回值	-

获取纹理可见数据包围盒

函数名	px_void PX_TextureGetVisibleRange(px_texture *ptexture,px_int *pLeft,px_int *pRight,px_int *pTop,px_int *pBottom);
说明	填充纹理
参数	ptexture 纹理 pleft,pright,ptop,pbottom 输出可见包围信息
返回值	-

图像算子

Priwitt 算子

函数名	px_void PX_ImageFilter_PriwittX(px_texture *ptexture,px_float out[]);
说明	Priwitt 算子 水平方向
参数	ptexture 纹理指针 out 输出矩阵，大小应该至少是纹理长宽
返回值	-

函数名	px_void PX_ImageFilter_PriwittY(px_texture *ptexture,px_float out[]);
说明	Priwitt 算子 垂直方向
参数	ptexture 纹理指针 out 输出矩阵，大小应该至少是纹理长宽
返回值	-

函数名	px_void PX_ImageFilter_Priwitt(px_texture *ptexture,px_float out[]);
说明	Priwitt 算子
参数	ptexture 纹理指针 out 输出矩阵，大小应该至少是纹理长宽
返回值	-

Sobel 算子

函数名	px_void PX_ImageFilter_SobelX(px_texture *ptexture,px_float out[]);
-----	---

说明	Sobel 算子 水平方向
参数	ptexture 纹理指针 out 输出矩阵，大小应该至少是纹理长宽
返回值	-

函数名	px_void PX_ImageFilter_SobelY(px_texture *ptexture,px_float out[]);
说明	Sobel 算子 垂直方向
参数	ptexture 纹理指针 out 输出矩阵，大小应该至少是纹理长宽
返回值	-

函数名	px_void PX_ImageFilter_Sobel (px_texture *ptexture,px_float out[]);
说明	Sobel 算子
参数	ptexture 纹理指针 out 输出矩阵，大小应该至少是纹理长宽
返回值	-

Roberts 算子

函数名	px_void PX_ImageFilter_RobertsX(px_texture *ptexture,px_float out[]);
说明	Sobel 算子 水平方向
参数	ptexture 纹理指针 out 输出矩阵，大小应该至少是纹理长宽
返回值	-

函数名	px_void PX_ImageFilter_RobertsY(px_texture *ptexture,px_float out[]);
说明	Sobel 算子 垂直方向
参数	ptexture 纹理指针 out 输出矩阵，大小应该至少是纹理长宽
返回值	-

函数名	px_void PX_ImageFilter_Roberts (px_texture *ptexture,px_float out[]);
说明	Sobel 算子
参数	ptexture 纹理指针 out 输出矩阵，大小应该至少是纹理长宽
返回值	-

Laplacian 算子

函数名	px_void PX_ImageFilter_LaplacianX(px_texture *ptexture,px_float out[]);
说明	Sobel 算子 水平方向
参数	ptexture 纹理指针 out 输出矩阵，大小应该至少是纹理长宽
返回值	-

函数名	px_void PX_ImageFilter_LaplacianY(px_texture *ptexture,px_float out[]);
说明	Sobel 算子 垂直方向
参数	ptexture 纹理指针 out 输出矩阵，大小应该至少是纹理长宽
返回值	-

函数名	px_void PX_ImageFilter_Laplacian (px_texture *ptexture,px_float out[]);
说明	Sobel 算子
参数	ptexture 纹理指针 out 输出矩阵，大小应该至少是纹理长宽
返回值	-

轮廓

PX_Shape 是 painterEngine 内建的轮廓格式,轮廓可以被认为是仅带有灰度及位置信息的纹理,轮廓的每个位置信息都由一个 256 阶的灰度表示(1 字节)在渲染轮廓的时候,需要制定一个颜色信息以对轮廓进行渲染.

创建轮廓

函数名	px_bool PX_ShapeCreate(px_memorypool *mp,px_shape *shape,px_int width,px_int height);
说明	创建一个轮廓
参数	mp 内存池指针 shape 输出轮廓 width 宽 height 高
返回值	若成功返回 PX_TRUE 否者 PX_FALSE

创建纹理轮廓

函数名	px_bool PX_ShapeCreateFromTexture(px_memorypool *mp,px_shape *shape,px_texture *texture);
说明	从纹理创建一个轮廓
参数	mp 内存池指针 shape 输出轮廓 texture 映射纹理
返回值	若成功返回 PX_TRUE 否者 PX_FALSE

函数名	px_bool PX_ShapeCreateFromMemory(px_memorypool *mp,px_void *data,px_int size,px_shape *shape);
说明	从内存创建一个轮廓(支持 TRaw 内存格式)
参数	mp 内存池指针 shape 输出轮廓 data 数据 size 数据大小
返回值	若成功返回 PX_TRUE 否者 PX_FALSE

渲染轮廓

函数名	px_void PX_ShapeRender(px_surface *psurface,px_shape *shape,px_int x,px_int y,PX_ALIGN refPoint,px_color blendColor);
说明	渲染一个轮廓到表面
参数	psurface 渲染到的表面 px_shape 需要渲染的轮廓 x , y 偏移量(该坐标以纹理左上角为参照) refPoint 参考中心点 , 用于表示纹理绘制的相对位置 blend blend 类型结构体 , 用于调整绘制纹理的 alpha , hdr 值 当它为 PX_NULL 时 , 表示采用默认 blend 值 blendcolor 渲染颜色
返回值	-

函数名	px_void PX_ShapeRenderEx(px_surface *psurface,px_shape *shape,px_int x,px_int y,PX_ALIGN refPoint,px_color blendColor,px_float scale,px_float Angle);
说明	旋转缩放后渲染一个轮廓到表面
参数	psurface 渲染到的表面

	shape 需要渲染的轮廓 x , y 偏移量(该坐标以纹理左上角为参照) refPoint 参考中心点，用于表示纹理绘制的相对位置 blend blend 类型结构体，用于调整绘制纹理的 alpha，hdr 值 当它为 PX_NULL 时，表示采用默认 blend 值 blendcolor 渲染颜色 scale 缩放大小 rotation 旋转角度
返回值	-

函数名	px_void PX_ShapeRenderEx_sincos(px_surface *psurface,px_shape *shape,px_int x,px_int y,PX_ALIGN refPoint,px_color blendColor,px_float scale,px_float sinx,px_float cosx);
说明	旋转后渲染一个轮廓到表面
参数	psurface 渲染到的表面 shape 需要渲染的轮廓 x , y 偏移量(该坐标以纹理左上角为参照) refPoint 参考中心点，用于表示纹理绘制的相对位置 blend blend 类型结构体，用于调整绘制纹理的 alpha，hdr 值 当它为 PX_NULL 时，表示采用默认 blend 值 blendcolor 渲染颜色 sinx cosx,旋转矩阵对应 sinx cosx 位置值,这个函数在以向量为标准的旋转标定渲染中尤为有用
返回值	-

释放轮廓

函数名	px_void PX_ShapeFree(px_shape *shape);
说明	释放轮廓
参数	shape 轮廓指针
返回值	-

位图

验证位图数据合法性

函数名	px_bool PX_BitmapVerify(void *BitmapBuffer,px_int size);
说明	验证数据是否为合法位图数据
参数	BitmapBuffer 位图数据 Size 数据长度
返回值	如果合法返回 PX_TRUE,否者 PX_FALSE

取得位图高度

函数名	px_uint PX_BitmapGetHeight(void *BitmapBuffer);
说明	取得位图高度
参数	BitmapBuffer 位图数据
返回值	位图的高度(可能为负数)

取得位图宽度

函数名	px_uint PX_BitmapGetWidth(void *BitmapBuffer);
说明	取得位图宽度
参数	BitmapBuffer 位图数据
返回值	位图的宽度(可能为负数)

取得位图像素数据

函数名	px_word PX_BitmapGetBitCount(void *BitmapBuffer);
说明	取得位图像素数据
参数	BitmapBuffer 位图数据

返回值	位图的位图像素数据
-----	-----------

渲染位图到表面

函数名	void PX_BitmapRender(px_surface *psurface,void *BitmapBuffer,px_int BufferSize,int x,int y);
说明	渲染位图到表面
参数	psurface 渲染表面 BitmapBuffer 位图数据 BufferSize 数据长度 x,y 偏移量
返回值	

Surface 转位图数据

函数名	px_bool PX_BitmapBuild(px_surface *psurface,px_char *BitmapBuffer,px_int *size);
说明	将一个 px_surface 转换成位图格式数据
参数	psurface 渲染表面 BitmapBuffer 输出数组,如果为 PX_NULL,表示仅计算需要的大小 size 实际输出的字节大小
返回值	如果成功返回 PX_TRUE,否者 PX_FALSE

TRaw 纹理格式

```
typedef struct _PX_TRaw_Header
{
    px_dword Magic;//TRAW 0x57415254
    px_int Width;
    px_int Height;
}PX_TRaw_Header;
```

TRaw 文件头，TRaw 文件是一个后缀为 tex 的文件

判断是否为 TRaw 合法文件格式

函数名	px_bool PX_TRawVerify(px_void *data,px_int size);
说明	判断是否为 TRaw 合法文件格式
参数	data 数据指针 size 数据大小
返回值	返回 TRUE 表示合法数据否者非合法数据

取得 TRaw 文件的图像宽度

函数名	px_int PX_TRawGetWidth(px_void *data);
说明	取得 TRaw 文件的图像宽度
参数	data 数据指针
返回值	返回图像宽度

取得 TRaw 文件的图像高度

函数名	px_int PX_TRawGetHeight(px_void *data);
说明	取得 TRaw 文件的图像高度
参数	data 数据指针
返回值	返回图像宽度

绘制 TRaw 数据

函数名	px_void PX_TRawRender(px_surface *psurface,px_void *data,px_int x,px_int y);
说明	绘制 TRaw 数据
参数	psurface 表面指针 data TRaw 数据 x , y 偏移量
返回值	-

将 pSurface 转换为 TRAW 格式数据

函数名	px_bool PX_TRawBuild(px_surface *psurface,px_byte *TRawBuffer,px_int *size);
说明	将 pSurface 转换为 TRAW 格式数据
参数	psurface 表面指针 TRawBuffer 输出数据(当 TRAWBuffer 不为 PX_NULL 时) size 输出大小
返回值	-

Delaunary 三角划分

函数名	px_bool PX_DelaunaryPointsBuild(px_memorypool *mp,px_point2D pt[],px_int count,px_vector *out_triangles);
说明	将点集以 Delaunary Triangle 进行空间划分,生成图形是一个凸包
参数	mp 运行计算内存池 pt 点集 count 点集点数量 out_Triangles 输出三角形
返回值	成功返回 PX_TRUE,否者 PX_FALSE

带限制区域的 Delaunary 三角建立

函数名	px_bool PX_PointsMeshBuild(px_memorypool *mp,px_point2D limit_line_pt[],px_int line_pt_count,px_point2D pt[],px_int pt_count,px_vector *out_triangles,PX_DELAUNAY_RETURN_TYPE type);
说明	将点集以 Delaunary 规则建立 Triangle , 同时, 剔除 limit_line_pt 绘制的闭环之外的三角形
参数	mp 运行计算内存池 limit_line_pt 用于绘制封闭图形的线段集 line_pt_count 用于绘制封闭图形的线段集点个数 pt 点集 count 点集点数量 out_Triangles 输出三角形 type 返回数据类型(以三角坐标返回或以索引形式返回)
返回值	成功返回 PX_TRUE,否者 PX_FALSE

纹理特效

发光

函数名	px_bool PX_EffectShine(px_surface *s,px_int radius,px_color color,px_float intension);
说明	对一个纹理施加发光效果
参数	S 目标纹理 Radius 半径 Color 颜色 Intension 强度
返回值	-

描边

函数名	px_bool PX_EffectOutline(px_surface *s,px_int radius,px_color color);
说明	对一个纹理施加描边效果
参数	S 目标纹理 Radius 半径 Color 颜色
返回值	-

字体

PainterEngine 内嵌了默认 ANSI 字模库,同时运行用户加载自己的字模库进行字体绘制

绘制字体

函数名	px_int PX_FontModuleDrawText(px_surface *psurface,PX_FontModule *mod,int x,int y,PX_ALIGN align,const px_char *Text,px_color Color);
说明	绘制文本,注意,输入的字模必须是初始化指定编码的

参数	<p>Psurface 目标表面</p> <p>X 原点 x 坐标</p> <p>Y 原点 y 坐标</p> <p>Text 文本</p> <p>Color 字颜色</p> <p>Mod 字模库,如果这个参数为 PX_NULL,表示以默认字模库进行绘制(仅支持 ANSI 字符)</p> <p>Align 字体对齐模式</p> <p>PX_ALIGN_LEFTTOP, 左上角对齐</p> <p> PX_ALIGN_MIDTOP,居中顶部对齐</p> <p> PX_ALIGN_RIGHTTOP,右上角对齐</p> <p> PX_ALIGN_LEFTMID,靠左居中对齐</p> <p> PX_ALIGN_CENTER,中心对齐</p> <p> PX_ALIGN_RIGHTMID,靠右居中对齐</p> <p> PX_ALIGN_LEFTBOTTOM,靠左底部对齐</p>
返回值	绘制文本的像素宽度

字模初始化

函数名	px_bool PX_FontModuleInitialize(px_memorypool *mp,PX_FontModule *module)
说明	初始化一个字模库
参数	Mp 内存池 Module 字模库
返回值	如果成功返回 PX_TRUE,否者返回 PX_FALSE

加载字模

函数名	px_bool PX_FontModuleLoad(PX_FontModule *module,px_byte *buffer,px_int size);
说明	加载 pxf 字模到字模库,重复的字模以之前加载的为准
参数	module 字模库 buffer pxf 数据 size pxf 数据大小
返回值	如果成功返回 PX_TRUE,否者返回 PX_FALSE

释放字模库

函数名	px_void PX_FontModuleFree(PX_FontModule *module);
说明	释放字模库
参数	module 字模库
返回值	-

绘制单个字

函数名	px_int PX_FontModuleDrawCharacter(px_surface *psurface,PX_FontModule *mod,int x,int y,const px_dword code,px_color Color);
说明	绘制单个字
参数	Psurface 渲染表面 mod 字模库 x ,y 左上角坐标 code 字编码 color 颜色
返回值	-

计算绘制文本的宽高

函数名	px_void PX_FontModuleTextGetRenderWidthHeight(PX_FontModule *module,const px_char *Text,px_int *advance,px_int *height);
说明	计算绘制文本的宽高
参数	module 字模库 Text 文本 Advance 步进长度 Height 绘制高度
返回值	-

信号处理

离散傅里叶变换 (DFT)

复数结构体

```
typedef struct __complex
{
    float re;// really
    float im;// imaginary
}px_complex;
```

函数名	void PX_DFT(_IN complex x[],_OUT complex X[],int N);
说明	对复数信号进行 DFT 正变换
参数	x 复信号 X 变换结果 N 复信号长度

函数名	void PX_IDFT(_IN complex X[],_OUT complex x[],int N);
说明	对复数信号进行 IDFT 即逆变换
参数	X 复信号 x 逆变换结果 N 复信号长度

离散余弦变换 (DCT)

复数结构体

函数名	void PX_DCT(_IN px_double x[],_OUT px_double X[],int N);
说明	对信号进行 DCT 正变换
参数	x 复信号 X 变换结果 N 复信号长度

函数名	void PX_IDFT(_IN px_double X[],_OUT px_double x[],int N);
说明	对信号进行 IDCT 即逆变换
参数	X 复信号 x 逆变换结果 N 复信号长度

快速傅里叶变换 (FFT)

函数名	void PX_FFT(_IN complex x[],_OUT complex X[],int N);
说明	对复数信号进行 FFT 正变换
参数	x 复信号 X 变换结果 N 复信号长度 (必须是 2 基数)

函数名	void PX_IFFT(_IN complex X[],_OUT complex x[],int N);
说明	对复数信号进行 IFFT 即逆变换
参数	X 复信号 x 逆变换结果 N 复信号长度 (必须是 2 基数)

函数名	void PX_FFT_2(_IN complex x[],_OUT complex X[],int N_N);
说明	对复数信号进行二维 FFT 正变换
参数	x 二维复信号矩阵 X 变换结果 N_N 复信号边长 (复信号必须是 2 基边长的正方矩阵)

函数名	void PX_IFFT_2(_IN complex X[],_OUT complex x[],int N_N);
说明	对复数信号进行 IFFT 即逆变换
参数	X 二维复信号矩阵 x 逆变换结果 N_N 复信号边长 (复信号必须是 2 基边长的正方矩阵)

函数名	void PX_FFT_2_Shift(_IN complex _in[],_OUT complex _out[],int N_N);
说明	对复数信号进行 FFTShift
参数	_in 二维复信号矩阵 _out 逆变换结果 N_N 复信号边长 (复信号必须是 2 基边长的正方矩阵)

强制共轭对称

函数名	void PX_FT_Symmetry(_IN px_complex X[],_OUT px_complex x[],px_int N);
说明	对一个傅里叶变换频域信号强制共轭对称(参考前半段数据)

参数	x 输入信号 X 输出信号 N 长度
----	--------------------------

上采样(插值, UpSampled)

函数名	void PX_UpSampled(_IN px_complex x[],_OUT px_complex X[],px_int N,px_int L)
说明	对一个信号进行上采样
参数	x 输入信号 X 输出信号 N 原长度 L 上采样倍数(1/M) 注意:输出数据缓存的长度应该至少是原长度的 2*L 倍 如果原信号或输出信号长度不是 2 基数,将使用较慢的 DFT 进行上采样

下采样(抽取, SubSampled, DownSampled)

函数名	void PX_DownSampled (_IN px_complex x[],_OUT px_complex X[],px_int N,px_int M)
说明	对一个信号进行下采样(SubSampled)
参数	x 输入信号 X 输出信号 N 原长度 M 下采样倍数(1/M) 注意:输出数据缓存的长度应该至少是原长度的两倍 如果原信号或输出信号长度不是 2 基数,将使用较慢的 DFT 进行下采样

tukey 窗

函数名	px_void PX_WindowFunction_tukey(px_double data[],px_int N);
说明	构造一个 tukey 窗函数
参数	data 数据缓存 N 长度
返回值	

triangular 窗

函数名	px_void PX_WindowFunction_ triangular (px_double data[],px_int N);
说明	构造一个 triangular 窗函数
参数	data 数据缓存 N 长度
返回值	

blackMan 窗

函数名	px_void PX_WindowFunction_blackMan(px_double data[],px_int N);
说明	构造一个 blackman 窗函数
参数	data 数据缓存 N 长度
返回值	

hamming 窗

函数名	px_void PX_WindowFunction_ hamming (px_double data[],px_int N);
说明	构造一个 hamming 窗函数
参数	data 数据缓存 N 长度
返回值	

hanning 窗

函数名	px_void PX_WindowFunction_ hanning (px_double data[],px_int N);
说明	构造一个 hanning 窗函数
参数	data 数据缓存 N 长度
返回值	

kaiser 窗

函数名	px_void PX_WindowFunction_kaiser(px_double beta,px_double data[],px_int N);
说明	构造一个 kaiser 窗函数
参数	beta 就是字面的意思 data 数据缓存 N 长度
返回值	

应用窗函数

函数名	px_void PX_WindowFunction_Apply(px_double data[],px_double window[],px_int N);
说明	将一个窗函数应用于离散信号
参数	data 数据缓存 window 窗函数缓存 N 长度
返回值	

计算系统幅频响应和相频响应

函数名	px_void PX_gain(px_double b[],px_double a[],px_int m,px_int n,px_double x[],px_double y[],px_int len,px_int sign);
说明	计算系统幅频响应和相频响应
参数	b 存放滤波器分子多项式系数 b[i],长度为 m+1 a 存放滤波器分母多项式系数 a[i],长度为 n+1 m 滤波器分子多项式阶数 n 滤波器分母多项式阶数 x,y 输出数组,长度为 len,当 sign 为 1 时存放滤波器幅频响应 $ H(w) $ (x 为实部,y 为虚部),当 sign 为 2 时存放分贝表示的滤波器幅频相频 $\varphi(w)$ len 见上一项说明 sign 见上一项说明
返回值	

计算级联型系统幅频响应和相频响应

函数名	px_double b[],px_double a[],px_int n,px_int ns,px_double x[],px_double y[],px_int len,px_int sign
说明	计算级联型系统幅频响应和相频响应
参数	b 存放滤波器分子多项式系数 b[i],矩阵,体积为 ns*(n+1) a 存放滤波器分母多项式系数 a[i],矩阵,体积为 ns*(n+1) n 级联滤波器每节多项式阶数 ns 级联滤波器 n 阶节数 L x,y 输出数组,长度为 len,当 sign 为 1 时存放滤波器幅频响应 $ H(w) $ (x 为实部,y 为虚部),当 sign 为 2 时存放分贝表示的滤波器幅频相频 $\phi(w)$ len 见上一项说明 sign 见上一项说明
返回值	

倒谱

函数名	void PX_Cepstrum(_IN px_complex x[],_OUT px_complex X[],px_int N, PX_CEOSTRUM_TYPE type);
说明	计算一个信号的倒谱域
参数	x 输入信号 X 倒谱域 N 信号长度 Type 倒谱类型, PX_CEOSTRUM_TYPE_COMPLEX 表示复倒谱 PX_CEOSTRUM_TYPE_REAL 表示实倒谱
返回值	

人声基音频率估算

函数名	px_int PX_PitchEstimation(_IN px_complex x[],px_int N,px_int sampleRate);
说明	估算人声基音频率
参数	x 输入信号 N 信号长度 sampleRate 信号采样率
返回值	估算的基音频率

估算瞬时频率

函数名	px_sine PX_PhaseVocoder(px_sine src,px_double p2,px_double delta_t)
说明	估算瞬时频率
参数	src 原正弦波信号 p2 第二相位 delta_t 取样时间
返回值	带有瞬时频率的正弦曲线

预加重

函数名	void PX_PreEmphasise(const px_double *data, int len, px_double *out, px_double preF)
说明	对一个实信号进行预加重
参数	data 输入信号 len 长度 out 输出预加重信号 preF 加重权值 范围为 0.9-1.0 一般取 0.97
返回值	

卡尔曼滤波

初始化卡尔曼滤波器

函数名	px_void PX_KalmanFilterInitialize(PX_KalmanFilter *filter,px_double A,px_double B,px_double Q,px_double H,px_double R);
说明	初始化卡尔曼滤波器
参数	filter 卡尔曼滤波实例 A,B,Q,H,R 参照卡尔曼滤波公式解释
返回值	

更新卡尔曼滤波器

函数名	px_void PX_KalmanFilterUpdate(PX_KalmanFilter *filter,px_double Zk,px_double uk,px_double wk);
说明	初始化卡尔曼滤波器
参数	filter 卡尔曼滤波实例 Zk 下一次更新值 uk,wk 参照公式
返回值	

预测值

函数名	px_double PX_KalmanFilterGetGuess(PX_KalmanFilter *filter)
说明	取得卡尔曼滤波器预测值
参数	filter 卡尔曼滤波实例
返回值	

MFCC

初始化 MFCC 滤波器组

函数名	px_void PX_MFCCInitialize(PX_MFCC *mfcc,px_int framesize,px_int sampleRate,px_int low,px_int high);
说明	初始化一个 MFCC 滤波器组
参数	mfcc mfcc 滤波器组 framesize 帧长度 samplerate 采样率 low hight 带通频率
返回值	

MFCC 滤波

函数名	px_bool PX_MFCCParse(PX_MFCC *mfcc,px_double *data,PX_MFCC_FEATURE *out);
说明	将数据通过 MFCC 滤波器组
参数	mfcc mfcc 滤波器组 data 帧数据，必须和帧长度匹配 out 输出的 mfcc 系数
返回值	如果满足一帧数据的 parse 返回 PX_TRUE，否则返回 PX_FALSE

WAV 文件格式

取得 wav 声道数目

函数名	px_uint PX_WAVEGetChannel(px_byte *buffer,px_int size);
说明	取得一个 wav 数据的声道数量
参数	buffer 指向 wav 数据指针 size wav 数据长度
返回值	声道数量

验证 wav 数据

函数名	px_bool PX_WAVEVerify(px_byte *buffer,px_int size);
说明	验证一段数据是否是可加载的 wav 数据
参数	buffer 指向 wav 数据指针 size wav 数据长度
返回值	如果可以返回 PX_TRUE,否则返回 PX_FALSE

取得 wav 数据的 PCM 流大小

函数名	px_uint PX_WAVEGetPCMSize(px_byte *buffer,px_int size);
说明	取得 wav 数据的 PCM 流大小

参数	buffer 指向 wav 数据指针 size wav 数据长度
返回值	返回数据长度

混音器

初始化

函数名	px_bool PX_SoundPlayInitialize(px_memorypool *mp, PX_SoundPlay *pSoundPlay);
说明	初始化一个声音混音器
参数	mp 内存池 pSoundPlay sound 结构体
返回值	如果成功返回 PX_TRUE,否者 PX_FALSE

设置用户自定义读函数

函数名	px_void PX_SoundPlaySetUserRead(PX_SoundPlay *pSoundPlay,px_void (*userread)(px_void *userptr,px_byte *pBuffer,px_int readSize),px_void *ptr);
说明	用户自定义读函数
参数	pSoundPlay sound 结构体 userread 自定义读函数
返回值	

暂停/继续数据播放

函数名	px_void PX_SoundPlayPause(PX_SoundPlay *pSoundPlay,px_bool pause);
说明	暂停/继续数据播放
参数	pSoundPlay sound 结构体 pause 为 PX_TRUE 表示暂停,PX_FALSE 表示继续
返回值	

加载播放数据

函数名	px_bool PX_SoundAdd(PX_SoundPlay *pSound,PX_Sound sounddata);
说明	加载一个播放数据混音器中
参数	pSoundPlay sound 结构体 sounddata 播放数据
返回值	返回 PX_TRUE 表示成功,否者返回 PX_FALSE

读取混音 PCM 数据流

函数名	px_bool PX_SoundRead(PX_SoundPlay *pSound,px_byte *pBuffer,px_int readSize);
说明	读取混音后的 PCM 流
参数	pSoundPlay sound 结构体 buffer PCM 数据流 readsize 读取长度
返回值	返回 PX_TRUE 表示成功,否者返回 PX_FALSE

释放混音器

函数名	px_void PX_SoundFree(PX_SoundPlay *pSound);
说明	释放混音器
参数	pSoundPlay sound 结构体
返回值	-

清除混音器所有音效

函数名	px_void PX_SoundClear(PX_SoundPlay *pSound);
说明	清除混音器所有音效
参数	pSoundPlay sound 结构体
返回值	-

计算混音器音源个数

函数名	px_int PX_SoundPlayGetDataCount(PX_SoundPlay *pSoundPlay);
说明	计算混音器音源个数
参数	pSoundPlay sound 结构体
返回值	混音器内音源个数

创建音源实例

函数名	PX_Sound PX_SoundCreate(PX_SoundData *data,px_bool loop);
说明	创建音源实例
参数	PX_SoundData 音源数据 loop 这个音源是否循环播放
返回值	音源实例

加载静态音源

函数名	px_bool PX_SoundStaticDataCreate(PX_SoundStaticData *sounddata,px_memorypool *mp,px_byte *data,px_int datasize);
说明	加载一个静态音源,在使用结束后,这个音源需要手动释放
参数	sounddata 静态音源 mp 存储内存池 data 原始解析数据(例如 WAV 格式) data 原始解析数据大小
返回值	成功返回 PX_TRUE,否者返回 PX_FALSE

释放静态音源

函数名	px_void PX_SoundStaticDataFree(PX_SoundStaticData *sounddata);
说明	释放静态音源
参数	sounddata 静态音源 mp 存储内存池 data 原始解析数据(例如 WAV 格式) data 原始解析数据大小
返回值	成功返回 PX_TRUE,否者返回 PX_FALSE

拷贝静态音源数据

函数名	px_bool PX_SoundStaticDataCopy(px_memorypool *mp,PX_SoundData *resSounddata,PX_SoundData *targetSounddata);
说明	拷贝一个静态音源,在使用结束后,这个音源需要手动释放
参数	mp 存储内存池 resSounddata 原始音源数据 targetSounddata 目标音源数据
返回值	成功返回 PX_TRUE,否者返回 PX_FALSE

调音台

时域延拓

函数名	px_bool PX_TuningTimeScale(px_double timescale,px_double in[],px_int count,px_memory *out);
说明	基于相位声码器的变速不变调 TSM(时域压拓)系统
参数	timescale 时域压拓系数 in 输入信号(建议归一化处理) count 信号数量 out 输出信号,该内存变量需要被初始化
返回值	成功返回 PX_TRUE,否者 PX_FALSE

调音台初始化

函数名	px_void PX_TuningInitialize(px_memorypool *mp,PX_Tuning *tuning,px_double pitchShift,px_double window[],px_double filter[],px_double fix[],PX_TUNING_WINDOW_SIZE windowsize);
说明	基于重采样和相位声码器的变调不变速系统调音台初始化
参数	mp 计算内存池 tuning 调音台指针

	<p>pitchshift 调变系数,例如音调提升 2 倍则为 2,降低一倍则为 0.5</p> <p>window 窗函数(建议为能量相等的处理窗,50% overlap)</p> <p>filter 滤波器,长度等于 windowSize 指示长度</p> <p>fix 增量值,长度等于 windowSize 指示长度(归一化后)</p> <p>windowSize 窗函数类型</p> <p>PX_TUNING_WINDOW_SIZE_8, 长度为 8 的窗,主要用于测试</p> <p>PX_TUNING_WINDOW_SIZE_128, 长度为 128 的窗</p> <p>PX_TUNING_WINDOW_SIZE_256, 长度为 256 的窗</p> <p>PX_TUNING_WINDOW_SIZE_512, 长度为 512 的窗,建议值</p> <p>PX_TUNING_WINDOW_SIZE_1024, 长度为 1024 的窗,建议值</p> <p>PX_TUNING_WINDOW_SIZE_2048, 长度为 2048 的窗,建议值</p> <p>PX_TUNING_WINDOW_SIZE_4096, 长度为 4096 的窗</p> <p>PX_TUNING_WINDOW_SIZE_8192, 长度为 8192 的窗</p>
返回值	

调音台滤波器系统

函数名	<code>px_int PX_TuningFilter(PX_Tuning *tuning,_IN px_double frame_unit[],px_int Size,_OUT px_double out_unit[]);</code>
说明	<p>基于重采样和相位声码器的变调不变速系统调音台滤波器系统</p> <p>*该滤波器为 FIR 类型滤波器</p> <p>*输入信号必须是连续的</p>
参数	<p>tuning 调音台指针</p> <p>frame_unit[] 输入数据(建议归一化处理)</p> <p>Size 输入数据长度</p> <p>out_unit 输出数据</p>
返回值	输出长度

调音台 TSM

函数名	<code>px_void PX_TuningTimeScale(px_double timescale,px_double in[],px_int count,px_memory *out);</code>
说明	调音台 TSM 算法
参数	<p>timescale 压拓系数</p> <p>in 输入信号(归一化建议)</p> <p>count 输入信号长度</p>

	out 输出内存
返回值	

设置调音台的 PitchShift

函数名	px_void PX_TuningSetPitchShift(PX_Tuning *tuning,px_double pitchShift);
说明	设置调音台的 PitchShift
参数	tuning 调音台指针 PitchShift 移调系数
返回值	

设置调音台的滤波器

函数名	px_void PX_TuningSetFilter(PX_Tuning *tuning,px_double filter[]);
说明	设置调音台的滤波器
参数	tuning 调音台指针 filter 滤波器系数
返回值	

设置调音台的频域补偿系数

函数名	px_void PX_TuningSetFix(PX_Tuning *tuning,px_double fix[]);
说明	设置调音台的频域补偿系数
参数	tuning 调音台指针 fix 频域补偿系数
返回值	

设置调音台的 ZCR 阈值

函数名	px_void PX_TuningSetZCR(px_double low,px_double high);
说明	设置调音台的 ZCR 阈值,当输入信号的 ZCR 不在阈值范围内时,将不对信号进行变调处理
参数	tuning 调音台指针

	low 最低 ZCR high 最高 ZCR
返回值	

反向传播神经网络

PainterEngine 提供一个基础的反向传播神经网络框架,集成了 linear tanh sigmod reLU 等激活函数,regularzation 方式及权值初始化方式.

初始化神经网络框架

函数名	<code>px_bool PX_ANNInitialize(px_memorypool *mp,PX_ANN *ann,px_double learningRate,PX_ANN_REGULARZATION regularzation,px_double regularization_rate);</code>
说明	初始化一个反向传播神经网络框架
参数	mp 运行内存池 ann 神经网络结构 learningRate 学习率 regularization 正则化方式 参考: typedef enum { PX_ANN_REGULARZATION_NONE, PX_ANN_REGULARZATION_L1, PX_ANN_REGULARZATION_L2 }PX_ANN_REGULARZATION; regularization_rate 正则化速率
返回值	返回 PX_TRUE 表示成功,否者返回 PX_FALSE

添加网络层

函数名	<code>px_bool PX_ANNAddLayer(PX_ANN *pAnn,px_int Neural,px_double</code>
-----	--

	bias,PX_ANN_ACTIVATION_FUNCTION activation,PX_ANN_LAYER_WEIGHT_INITMODE mode,px_double weight_c);
说明	添加一个网络层(其中第一层为输入层,最后一层为输出层)
参数	ann 神经网络结构 Neurons 神经元数量 bias 偏置 activation 激活函数类型,参照 typedef enum { PX_ANN_ACTIVATION_FUNCTION_SIGMOID, //sigmoid PX_ANN_ACTIVATION_FUNCTION_TANH, //tanh PX_ANN_ACTIVATION_FUNCTION_LINEAR, //linear PX_ANN_ACTIVATION_FUNCTION_RELU, //ReLU }PX_ANN_ACTIVATION_FUNCTION; mode 权值初始方式,参照 typedef enum { PX_ANN_LAYER_WEIGHT_INITMODE_CONST, //常量 PX_ANN_LAYER_WEIGHT_INITMODE_RAND, //随机数 PX_ANN_LAYER_WEIGHT_INITMODE_GAUSSRAND, //正态分布随机数 }PX_ANN_LAYER_WEIGHT_INITMODE; weight_c 当权值初始方式为常量时的常量值
返回值	返回 PX_TRUE 表示成功,否则返回 PX_FALSE

训练网络

函数名	px_double PX_ANNTrain(PX_ANN *pAnn,px_double *input,px_double *expect);
说明	监督学习训练网络
参数	ann 神经网络结构 input 输入数组(与输入层神经元对应) expect 期望输出数组(与输出层神经元对应)
返回值	train loss 值

前向传播

函数名	px_void PX_ANNForward(PX_ANN *pAnn,px_double *input);
说明	神经网络前向传播
参数	ann 神经网络结构 input 输入数组(与输入层神经元对应)
返回值	-

取得输出结果

函数名	px_void PX_ANNGetOutput(PX_ANN *pAnn,px_double *result);
说明	神经网络前向传播的输出结果
参数	ann 神经网络结构 result 输出数组(与输入层神经元对应)
返回值	-

重置神经网络

函数名	px_void PX_ANNReset(PX_ANN *pANN);
说明	重置神经网络的所有权重
参数	ann 神经网络结构
返回值	-

释放神经网络框架

函数名	px_void PX_ANNFree(PX_ANN *pAnn);
说明	释放框架
参数	ann 神经网络结构
返回值	-

导出神经网络框架

函数名	px_bool PX_ANNExport(PX_ANN *pAnn,px_void *buffer,px_int *size);
说明	将神经网络当前的训练框架导出到内存结构
参数	ann 神经网络结构. buffer 导出 buffer(当该值为 PX_NULL 时表示只计算大小) size 导出数据大小
返回值	成功返回 PX_TRUE,否者返回 PX_FALSE

导入神经网络框架

函数名	px_bool PX_ANNImport(px_memorypool *mp,PX_ANN *pAnn,px_void *buffer,px_int size);
说明	从内存结构中导入神经网络框架
参数	mp 神经网络使用内存池 ann 未初始化的神经网络结构. buffer 导入 buffer size 导入数据大小
返回值	成功返回 PX_TRUE,否者返回 PX_FALSE

物理

四叉树 AABB 碰撞检测

PainterEngine Core 提供四叉树 AABB 碰撞检测算法,该算法用于加速矩形 box 区域的碰撞检测,使用自定义的树深调节对

初始化

函数名	px_bool PX_QuadtreeCreate(px_memorypool *mp,PX_Quadtree *pQuadtree,px_float mapStartX,px_float mapStartY,px_float mapWidth,px_float mapHeight,px_int ObjectsCount,px_int deep);
说明	创建四叉树根结构

参数	mp 一次性内存池,该内存池不能够存在其他的节点,仅用于四叉树计算使用 pQuadtree, 四叉树结构体 mapStartX 需要分隔的域左上角起始坐标 X mapStartY 需要分隔的域左上角起始坐标 Y mapWidth,mapHeight 域的宽度和高度 ObjectsCount 需要计算的节点数 deep 深度,建议在 2-4 范围,起始深度为 0,每个深度分 4 个区域
返回值	如果成功返回 PX_TRUE,否者为 PX_FALSE

添加节点

函数名	px_void PX_QuadtreeAddNode(PX_Quadtree *pQuadtree,px_float CenterX,px_float CenterY,px_float width,px_float height,PX_Quadtree_UserData userData);
说明	在四叉树添加一个节点,该节点会不参与碰撞计算
参数	pQuadtree 四叉树结构体 CenterX, CenterY,width,height,AABB 描述 userData 碰撞测试的用户数据,这个数据将被附加到碰撞结果中,注意,相同的 userData 将被认为是重复数据

碰撞测试

函数名	px_void PX_QuadtreeTestNode(PX_Quadtree *pQuadtree,px_float x,px_float y,px_float width,px_float height,PX_Quadtree_UserData userData)
说明	在四叉树设置接参与碰撞计算的节点,这个节点不会被加入四叉树中
参数	pQuadtree 四叉树结构体 x,y,z,width,height,length AABB box userData 该节点的标识号,该索引将出现在碰撞结果中,注意,如果节点的 ptr 相同,则该节点将标记为重复节点不计入碰撞检测中

重置碰撞表

函数名	px_void PX_QuadtreeResetTest(PX_Quadtree *pQuadtree);
说明	重置四叉树的碰撞表
参数	pQuadtree 四叉树结构体

密码学

AES

函数名	px_void PX_AES_Initialize(PX_AES *aesStruct,PX_AES_KeySize keySize,AES_BYTE keyByte[]);
说明	初始化 AES 编码器
参数	aseStruct AES 编码器 keySize 128,192,256 加密 keyByte 对应位数长度密钥

函数名	px_bool PX_AES_Cipher(PX_AES *aesStruct,void *input, void *output);
说明	编码 16 字节的数据
参数	aseStruct AES 编码器 input 输入 output 输出

函数名	px_bool PX_AES_CipherBuffer(PX_AES *aesStruct,void *input, px_int size,void *output);
说明	编码一段的数据,这个数据的长度必须是 16 的整数倍
参数	aseStruct AES 编码器 input 输入 size 长度 output 输出

函数名	px_void PX_AES_InvCipher(PX_AES *aesStruct,void
-----	---

	<code>*input, void *output);</code>
说明	解码 16 字节的数据
参数	aseStruct AES 编码器 input 输入 output 输出

函数名	<code>px_bool PX_AES_InvCipherBuffer(PX_AES *aesStruct,void *input, px_int size,void *output);</code>
说明	解码一段的数据,这个数据的长度必须是 16 的整数倍
参数	aseStruct AES 编码器 input 输入 size 长度 output 输出

curve25519

curve25519 是 ECC 加密衍生的 Diffie-Hellman 函数,用于生成一个公共密钥保证通讯安全

函数名	<code>px_void curve25519_donna(px_byte *mypublic, const px_byte *secret, const px_byte *basepoint);</code>
说明	解码 16 字节的数据
参数	mypublic 32 字节的公钥,输出 secret 32 字节私钥,输入 basepoint 32 字节长度的基点 私钥需要如下处理 mysecret[0] &= 248; mysecret[31] &= 127; mysecret[31] = 64; 基点可以为 static const px_byte basepoint[32] = {9};

SHA256

函数名	PX_Sha256Calculate (void*Buffer,px_uint32 BufferSize, PX_SHA256_HASH* Digest);
说明	Sha256
参数	buffer 输入缓存 buffersize 缓存大小 digest 输出 hash 值

编码及数据压缩

BASE64

编码

函数名	px_uint PX_Base64Encode(const px_byte *in, px_uint inlen, px_char *out);
说明	使用 Base64 编码一段数据
参数	in 输入数据缓存 input_size 输入数据长度 _out 输出数据缓存,不可为 NULL,可以使用 PX_Base64GetEncodeLen 预先计算编码后长度
返回值	编码长度

解码

函数名	px_uint PX_Base64Decode(const px_char *_in, px_uint input_size, px_byte *out);
说明	使用 Base64 解码一段数据
参数	in 输入数据缓存 input_size 输入数据长度

	_out 输出数据缓存,不可为 NULL,可以使用 PX_Base64GetDecodeLen 预先计算解码后长度
返回值	解码长度,如果为 0 表示无法解码

长度计算

函数名	px_uint PX_Base64GetEncodeLen(px_uint codeLen); px_uint PX_Base64GetDecodeLen(px_uint codeLen);
说明	计算 BASE64 的编码/解码长度
参数	Codelen 输入数据长度
返回值	编码/解码后的长度,如果为 0 表示无法编/解码

ARLE

ARLE(AdvanceRun Length Encoding)改进行程长度压缩算法,主要适用于图像及音频数据的压缩,最坏情况下,长度为 $N*4/3$,最优秀情况下,为长度的 $N*2/127$

压缩

函数名	px_void PX_ArleCompress(px_byte *_in,px_uint input_size,px_byte *_out,px_uint *_outsize);
说明	使用 ARLE 压缩一段数据
参数	_in 输入数据缓存 input_size 输入数据长度 _out 输出数据缓存,可以为 NULL 表示仅计算输出长度 _outsize 输出长度

解压缩

函数名	px_void PX_ArleCompress(px_byte *_in,px_uint input_size,px_byte *_out,px_uint *_outsize);
说明	使用 ARLE 压缩一段数据

参数	<code>_in</code> 输入数据缓存 <code>input_size</code> 输入数据长度 <code>_out</code> 输出数据缓存,可以为 NULL 表示仅计算输出长度 <code>_outsize</code> 输出长度
----	--

Huffman

压缩

函数名	<code>px_void PX_HuffmanCompress(px_byte *_in,px_uint input_size,px_byte *_out,px_uint *_outsize);</code>
说明	使用 Huffman 压缩一段数据
参数	<code>_in</code> 输入数据缓存 <code>input_size</code> 输入数据长度 <code>_out</code> 输出数据缓存,可以为 NULL 表示仅计算输出长度 <code>_outsize</code> 输出长度

解压缩

函数名	<code>px_void PX_Huffmandecompress(px_byte *_in,px_uint input_size,px_byte *_out,px_uint *_outsize);</code>
说明	使用 Huffman 压缩一段数据
参数	<code>_in</code> 输入数据缓存 <code>input_size</code> 输入数据长度 <code>_out</code> 输出数据缓存,可以为 NULL 表示仅计算输出长度 <code>_outsize</code> 输出长度

3D 渲染管线

PainterEngine 提供一个轻量级软件 3D 渲染管线,其中大部分参数都以默认方式部署(质量优先),可用于 3D 动画渲染,静态模型渲染,但因为渲染性能消耗较大,并不建议使用该 3D 渲染管线直接开发需要动态渲染的复杂游戏,3D 渲染管线是作为一个模块存在于 PainterEngine 中的,PainterEngine 并不仅作为 3D 渲染器,不建议将 3D 渲染模式作为 PainterEngine 中的主要使用模式中.

世界矩阵

函数名	<code>px_void PX_3D_WorldInitialize(PX_3D_World *world,px_float x,px_float y,px_float z,px_float rotX,px_float rotY,px_float rotZ,px_float scale);</code>
说明	初始化一个世界坐标,在渲染中该世界矩阵将直接作用于渲染列表
参数	world 世界矩阵 x,y,z 世界矩阵的平移坐标 rotX,rotY,rotZ 绕 X,Y,Z 轴的旋转角度 scale 缩放比例

欧拉相机矩阵

函数名	<code>px_void PX_3D_CameraEulerInitialize(px_memorypool *mp,PX_3D_Camera *camera,px_point4D cameraPosition,px_float rotX,px_float rotY,px_float rotZ,px_float near,px_float far,px_float fov,px_float viewPortWidth,px_float viewPortHeight);</code>
说明	初始化一个欧拉相机矩阵,在渲染中该矩阵将直接作用于渲染列表
参数	mp 内存池,用于分配 z 缓存 camera 相机矩阵 camera 相机位置 rotX,rotY,rotZ 绕 X,Y,Z 轴的旋转角度 far 远裁剪面 near 近裁剪面,这个值不能小于 1.0 fov 视锥角度 viewportWidth 窗口宽度(像素为单位) viewPortHeight 窗口高度(像素为单位)

UVN 相机矩阵

函数名	<code>px_void PX_3D_CameraUVNInitialize(px_memorypool *mp,PX_3D_Camera *camera,px_point4D cameraPosition,px_point4D cameraTarget,px_float near,px_float far,px_float fov,px_float viewPortWidth,px_float viewPortHeight);</code>
说明	初始化一个 UVN 相机矩阵,在渲染中该矩阵将直接作用于渲染列表
参数	mp 内存池,用于分配 z 缓存

	camera 相机矩阵 cameraposition 相机位置 target 相机指向位置 far 远裁剪面 near 近裁剪面,这个值不能小于 1.0 fov 视锥角度 viewportWidth 窗口宽度(像素为单位) viewPortHeight 窗口高度(像素为单位)
--	---

设置 UVN 相机位置

函数名	px_void PX_3D_CameraSetPosition(PX_3D_Camera *camera,px_point4D cameraPosition,px_point4D cameraTarget);
说明	设置 uvn 相机位置
参数	cameraposition 相机位置 target 相机指向位置

释放相机矩阵

函数名	px_void PX_3D_CameraFree(PX_3D_Camera *camera)
说明	释放相机矩阵
参数	camera 相机矩阵

渲染列表

函数名	px_bool PX_3D_RenderListInitialize(px_memorypool *mp,PX_3D_RenderList *list,px_dword PX_3D_PRESENTMODE,PX_3D_CULLMODE cullmode,px_texture *ptexture);
说明	渲染列表是 PainterEngine 渲染管线的基本渲染单元 所有的面都必须加载到该渲染列表中进行渲染
参数	mp 使用内存池 list 渲染列表 PX_3D_PRESENTMODE 显示方式

	<p>包括:</p> <pre>#define PX_3D_PRESENTMODE_LINE 1 线段渲染 #define PX_3D_PRESENTMODE_TEXTURE 2 纹理渲染 #define PX_3D_PRESENTMODE_PURE 4 纯色渲染</pre> <p>cullmode 裁剪模式 PX_3D_CULLMODE_NONE, 不裁剪 PX_3D_CULLMODE_CW, 顺时针裁剪 PX_3D_CULLMODE_CCW, 逆时针裁剪</p> <p>ptexture 纹理</p>
--	--

渲染列表添加三角形面

函数名	px_bool PX_3D_RenderListPush(PX_3D_RenderList *list,PX_3D_Face face);
说明	将一个三角形面添加到渲染列表当中
参数	<p>list 渲染列表 face 三角形面</p> <pre>typedef struct { px_point4D position;//顶点位置 px_vector4D normal;//法线(由渲染管线计算) px_color clr;//颜色 px_float u,v;//uv 纹理映射坐标 }PX_3D_Vertex;</pre> <pre>typedef struct { px_dword state;//由渲染管线控制 PX_3D_Vertex vertex[3];//顶点 PX_3D_Vertex transform_vertex[3];//变换后顶点,由渲染管线控制 }PX_3D_Face;</pre> <p>每次使用 PX_3D_Face 建议对其内存清 0</p>

推算凹凸纹理

函数名	px_point *PX_3D_CreateTextureNormal(px_memorypool *mp,px_texture *pTexture);
说明	以明暗关系推算一张纹理的法向量
参数	Mp 法向量矩阵分配空间内存池 pTexture 需要计算的纹理
返回值	和纹理等高*宽的法向量数组

推算高度图的凹凸纹理

函数名	px_point *PX_3D_CreateBumpTextureNormal(px_memorypool *mp,px_texture *pTexture);
说明	以高度图推算其凹凸纹理
参数	Mp 法向量矩阵分配空间内存池 pTexture 需要计算的高度图纹理
返回值	和纹理等高*宽的法向量数组

渲染列表矩阵变换

函数名	px_void PX_3D_RenderListTransform(PX_3D_RenderList *list,px_matrix mat,PX_3D_RENDERLIST_TRANSFORM t);
说明	将渲染列表中每一个顶点都使用 mat 矩阵变换
参数	list 渲染列表 mat 变换矩阵 PX_3D_RENDERLIST_TRANSFORM PX_3D_RENDERLIST_TRANSFORM_LOCAL_TO_LOCAL,变换原始顶点并替换原始顶点 PX_3D_RENDERLIST_TRANSFORM_LOACL_TO_GLOBAL,变换原始顶点,并将结果放置在变换顶点中 PX_3D_RENDERLIST_TRANSFORM_GLOBAL_TO_GLOBAL,变换变换顶点,并将结果放置在变换顶点中

渲染列表清空

函数名	px_void PX_3D_RenderListReset(PX_3D_RenderList *list);
说明	清空渲染列表,注意:该函数并不清理内存,该列表仍然可用
参数	list 渲染列表

释放列表清空

函数名	px_void PX_3D_RenderListFree(PX_3D_RenderList *list);
说明	释放渲染列表,并清理内存,该列表将不可用
参数	list 渲染列表

开始渲染

函数名	px_void PX_3D_Scene(PX_3D_RenderList *list,PX_3D_World *world,PX_3D_Camera *camera)
说明	开始渲染(在 Present 之前调用)
参数	list 渲染列表 word 世界矩阵 camera 相机矩阵

渲染列表

函数名	px_void PX_3D_Present(px_surface *psurface, PX_3D_RenderList *list);
说明	渲染列表到 surface
参数	psurface 渲染表面 list 渲染列表 world 世界矩阵 camera 相机矩阵

设置 PixelShader

函数名	px_void PX_3D_RenderListSetPixelShader(PX_3D_RenderList *list,PX_3D_PixelShader func);
说明	设置渲染流水线的 PixelShader
参数	list 渲染列表 func PixelShader 回调函数

Kernel

词法分析机

PainterEngine Kernel 提供通用词法分析机,用于对常用文本类型的脚本,数据格式语言的词法分析,在使用词法分析机之前,都应该使用 PX_LexerInit 对词法分析机进行初始化操作,在词法分析机使用完毕后,应该使用 PX_LexerFree 释放其占用资源.

初始化词法分析机

函数名	px_void PX_LexerInit(px_lexer *lexer,px_memorypool *mp);
说明	初始化词法分析机
参数	lexer px_lexer 结构指针 mp 词法分析内存池

注册注释块

函数名	px_void PX_LexerRegisterComment(px_lexer *lexer,px_char Begin[],px_char End[]);
说明	注册注释块
参数	lexer px_lexer 结构指针 begin 注释块的起始字符串 end 注释块的结束字符串
备注	如果注释块的结束字符串是回车,则该注释块在解析后为一个回车符

注册包含块

函数名	px_uint PX_LexerRegisterContainer(px_lexer *lexer,px_char Begin[],px_char End[]);
说明	注册注释块
参数	lexer px_lexer 结构指针 begin 包含块的起始字符串 end 包含块的结束字符串
返回值	该包含块的索引 ID

注册空白符

多个连续的空白符在解析过程中将被解析为单个空白符

函数名	px_void PX_LexerRegisterSpacer(px_lexer *lexer,px_char Spacer);
说明	注册空白符
参数	lexer px_lexer 结构指针 Spacer 空白符

注册分隔符

函数名	px_uint PX_LexerRegisterDelimiter(px_lexer *lexer,px_char Delimiter);
说明	注册分隔符
参数	lexer px_lexer 结构指针 Delimiter 分隔符
返回值	该分隔符的 ID

取得分隔符 ID

函数名	px_uint PX_LexerGetDelimiterType(px_lexer *lexer,px_char Delimiter);
说明	取得分隔符 ID
参数	lexer px_lexer 结构指针 Delimiter 分隔符
返回值	该分隔符的注册 ID

取得字符串包含块类型

函数名	px_uint PX_LexerGetContainerType(px_lexer *lexer,px_char *pContainerText);
说明	取得字符串包含块类型
参数	lexer px_lexer 结构指针 pContainerText 解释文本
返回值	该包含块的注册 ID

释放词法分析机资源

函数名	px_void PX_LexerFree(px_lexer *lexer);
说明	释放词法分析机资源
参数	lexer px_lexer 结构指针

加载分析文本

函数名	px_bool PX_LexerSortText(px_lexer *lexer,px_char *SourceText); px_bool PX_LexerLoadSourceFromMemory(px_lexer *lexer,px_byte *buffer);
说明	释放词法分析机资源
参数	lexer px_lexer 结构指针 SourceText Buffer 分析文本
返回值	如果成功解析返回 PX_TRUE,否则返回 PX_FALSE(词法错误)

读取文本

函数名	px_bool PX_LexerReadString(px_lexer *lexer,px_string *str,px_uint size);
说明	在当前词法分析状态下读取文本,该文本不置入词法分析
参数	lexer px_lexer 结构指针 str 字符串指针 size 大小
返回值	如果成功解析返回 PX_TRUE,否则返回 PX_FALSE(词法错误)

取得特殊符号

函数名	px_char PX_LexerGetSymbol(px_lexer *lexer);
说明	取得当前分析的符号(分隔符,空白符或回车符)
参数	lexer px_lexer 结构指针
返回值	当前分析的符号

取得当前词

函数名	px_void PX_LexerGetLexemeString(px_lexer *lexer,px_string *str);
说明	取得当前词文本
参数	lexer px_lexer 结构指针 str px_string 指针
返回值	

可以通过访问 px_lexer-> CurLexeme 直接取得词字符串

取得包含文本

函数名	px_void PX_LexerGetIncludedString(px_lexer *lexer,px_string *str);
说明	如果当前词为包含类型,则取得其包含文本
参数	lexer px_lexer 结构指针 str px_string 指针
返回值	

设置词大小写敏感

函数名	px_void PX_LexerSetTokenCase(px_lexer *lexer,PX_LEXER_LEXEME_CASE _case);
说明	设置返回词大小写,为 PX_LEXER_LEXEME_CASE 枚举类型,如果被设置,读取的词除包含类型外都会转换为对应大小写
参数	lexer px_lexer 结构指针 _case 大小写类型
返回值	

取得词法分析机当前状态

函数名	PX_LEXER_STATE PX_LexerGetState(px_lexer *lexer);
说明	取得当前词法分析机状态,可以在之后重设该状态
参数	lexer px_lexer 结构指针
返回值	词法分析机当前状态

设置词法分析机当前状态

函数名	px_void PX_LexerRestoreState(PX_LEXER_STATE state);
说明	重新设置词法分析机状态
参数	state 词法分析机状态结构体

返回值	-
-----	---

判断当前词是否是合法数字

函数名	px_bool PX_LexerIsLememelsNumeric(px_lexer *lexer);
说明	判断当前词是否是合法数字
参数	lexer px_lexer 结构指针
返回值	如果是返回 PX_TRUE 如果不是返回 PX_FALSE

读取下一个字母

函数名	px_char PX_LexerGetNextChar(px_lexer *lexer);
说明	直接读取下一个字母
参数	lexer px_lexer 结构指针
返回值	下一个字母

读取下一个词

函数名	PX_LEXER_LEXEME_TYPE PX_LexerGetNextLexeme(px_lexer *lexer);
说明	直接读取下一个词
参数	lexer px_lexer 结构指针
返回值	该词的词性,参考 PX_LEXER_LEXEME_TYPE PX_LEXER_LEXEME_TYPE_END =0, 结束 PX_LEXER_LEXEME_TYPE_SPACER =1,空白符 PX_LEXER_LEXEME_TYPE_DELIMITER =2,分割符 PX_LEXER_LEXEME_TYPE_CONATINER =3,包含块 PX_LEXER_LEXEME_TYPE_NEWLINE =4,换行符 PX_LEXER_LEXEME_TYPE_TOKEN =5,词块 PX_LEXER_LEXEME_TYPE_ERR = -1,错误的词性

取得当前词性

函数名	PX_LEXER_LEXEME_TYPE PX_LexerGetCurrentLexeme(px_lexer *lexer);
说明	取得当前词词性
参数	lexer px_lexer 结构指针
返回值	该词的词性,参考 PX_LEXER_LEXEME_TYPE PX_LEXER_LEXEME_TYPE_END =0, 结束 PX_LEXER_LEXEME_TYPE_SPACER =1,空白符 PX_LEXER_LEXEME_TYPE_DELIMITER =2,分割符 PX_LEXER_LEXEME_TYPE_CONATINER =3,包含块 PX_LEXER_LEXEME_TYPE_NEWLINE =4,换行符 PX_LEXER_LEXEME_TYPE_TOKEN =5,词块 PX_LEXER_LEXEME_TYPE_ERR = -1,错误的词性

PainterScript 语法规则

脚本语法

Painter Script 是 PainterEngine 所支持下的编译型脚本,是一个文件系统无关的支持脚本.
Painter Script 以下简称为 SS,其具有如下的语法特性.

- Painter Script 是大小写无关的语言.
- Painter Script 是文件系统无关的语言,所有 SS 脚本都不以文件系统作为标识,所有的 SS 源文件必须以一个#name “id”作为标识
- 其中 id 可以为任意合法字符,在一个 Painter Script 项目中,这个 id 必须是唯一的
- Painter Script 是一个弱类型语言,类型有 整数 浮点数 字符串类型 内存类型
- Painter Script 使用双引号表示一个字符串类型
- Painter Script 使用单引号表示一个字符类型
- SS 使用大括号表示一个内存类型
- SS 以分号作为语句结束

以下以 BNF 文法来描述 Painter Script 语法结构

语法规则(BNF 文法)

定义

Token::=符合命名规范的词或数字;

vName::= 符合命名规范的词

Numeric::=一个合法的数字,包括以 0x 开头的 16 进制数(最大不超过 32 位)

预处理

#define name Token

#define 用于将一个标识符替换为另一个标识符

所有 name 类型都会被替换为 Token

<预定义>::=" #define" <Token> <Token>

例如

#define PI 3.14159265

表示之后的 PI 都会被替换为 3.14159265

#include "name"

#include 用于将一个以 name 命名的源文件替换到当前预定义处

#include_STATEMENT::=" #include" <"name">

#runtime thread count

#runtime thread 用于定义最大支持线程数量

#runtime stack count

#runtime stack 用于定义默认的栈大小

注释

PainterScript 的注释方式与 C 语言一致

一种为双斜杠,代表本行注释,例如,下面是一个合法的注释

//note

同时,PainterScript 支持/**/类型注释.例如下文也是一个合法的注释

```
/*note  
note  
*/
```

常量

PainterScript 有以下几种常量

数字

例如 123,456 及 3.14159 都是合法的数字常量,同时,PainterScript 支持十六进制数常量

例如 0x01,0xffffffff,其大小不应该超过 32 位数的定义

最后,字符 ascii 的表达方式也被解释为数字,例如

‘a’表示 a 的 ascii 码,其也作为一种数字常量

字符串

用引号来包含一个字符串运算,例如

“hello world”

“Painter Script”

都是一个合法的字符串常量

数据流

使用一对@来定义一个数据流常量类型

例如

@FF0102030405060708090a0b0c0f@

数据流中,每字节数据都由成对的十六进制符定义完成,这代表序列中的数据必须被严格的配对定义

变量类型

PainterScript 支持以下几种类型

int

在 PainterScript 中,int 类型被定义为一个 32 位的有符号整数.

float

在 PainterScript 中,float 类型被定义为一个 32 位的有符号浮点数,该类型需要在符合 IEEE754 标准的编译器中被定义,否则,它将使用 Q0.15 的定点数标准进行实现..

string

字符串类型

memory

数据流类型

set

用户自定义集合类型

在 PainterScript 中,所有变量必须预声明后使用,一个变量名在其有效域范围内仅能够被定义一次

其 BNF 描述为

VARIABLE_TYPE::=int|float|string|memory

VARIABLE_DEFINE_STATEMENT::=< VARIABLE_TYPE > {<vName>[,]}

变量定义

常规变量定义

如下皆为合法常规变量定义

定义一个整形

```
int l;
```

连续定义多个整形

```
int a,b,c;
```

定义集合类型和集合数组

```
set a,b[10];
```

指针类型

PainterScript 支持一级指针,所有的变量类型皆可定义为一个一级指针类型

```
POINTER_STATEMENT::=<VARIABLE_TYPE> * <vName>;
```

eg:如下代码定义了一个 int 类型指针

```
int *p;
```

数组

PainterScript 支持一维数组,其 BNF 文法描述为

```
ARRAY_STATEMENT::= <VARIABLE_TYPE> <vName> "[" <Numeric> "]" ;
```

eg:使用如下方式定义一个 int 类型数组

```
int a[10];
```

定义一个一级整形指针数组,和一个整形数组

```
int *a[10],b[100];
```

集合

PainterScript 是参照 C 语言设计的一门脚本语言,在 PainterScript 中,使用数学描述 Set(集合)来替代 C 语言的 struct,同时,PainterScript 吸取了 C++中结构体的定义方式,定义一个集合文法如下

```
SET_STATEMENT::= set <vName> "{" <.....> "}"
```

例如,下面是一个合法的定义例子

```
set MySet
{
    int a,b,c;
    float d;
    string e[2];
```

```
memory f;
}
```

结构体运行相互嵌套,例如,下面的结构体定义也是合法的

```
set MySet2
{
MySet a,b[5];
int c;
}
```

表达式

在 PainterScript 中,脚本由逻辑结构及表达式组成:

运算符及优先级表

PainterScript 参照了 C 语言的运算符进行设计,支持以下运算符
其中有所区别的是,sizeof 被解析为返回内存类型或字符串类型的长度

运算符	解释	结合方式
() [] -> .	括号（函数等），数组，两种结构成员访问	由左向右
! ~ ++ -- + - * & (类型)	否定，按位否定，增量，减量，正负号， 间接，取地址，类型转换	由右向左
* / %	乘，除，取模	由左向右
+ -	加，减	由左向右
<< >>	左移，右移	由左向右
< <= >= >	小于，小于等于，大于等于，大于	由左向右
== !=	等于，不等于	由左向右

&	按位与	由左向右
^	按位异或	由左向右
	按位或	由左向右
&&	逻辑与	由左向右
	逻辑或	由左向右
=	赋值运算	由右向左
,	逗号 (顺序)	由左向右

其运算符优先级同样参照 C 语言运算符的优先级表进行设计

- 1 () [] . ->
- 2 ! ~ - (负号) ++ -- & *
- 3 * / %
- 4 + -
- 5 >> <<
- 6 > >= < <=
- 7 == !=
- 8 &
- 9 ^
- 10 |
- 11 &&
- 12 ||
- 13 ?
- 14 =
- 15 ,

类型访问

int,float 类型访问

int 及 float 变量类型可以直接对其助记符进行访问

例如

```
int a;  
a=123;  
a=a+3;  
a++;
```

都是合法的访问

int 及 float 的相互赋值将导致隐式转换,例如

```
int a;  
a=3.14;
```

那么 a 的值将转换为 3

```
a=5;  
a=a/2;a 的值为 2;
```

string 类型访问

string 类型仅允许 string 类型及字符串常量间的相互操作

例如下面都是合法的语句

```
string foo;  
foo="abc";
```

string 类型允许加法运算

例如

```
string foo;  
  
foo="Hello";  
foo=foo+"World";  
foo=foo+foo;  
//foo 最终为 HelloWorldHelloWorld;
```

同时,string 类型允许访问其元素,例如

```
string foo="HelloWolrd";  
int a;  
a=foo[0];  
//a 的值为 H 的 ase 码  
foo[0]='M';  
//foo 变换为 MelloWorld;
```


memory 类型访问

memory 类型仅允许 memory 类型及字符串常量间的相互操作

例如下面都是合法的语句

```
memory foo;
foo=@010203@
```

memory 类型允许加法运算,但加法仅接受一个范围为 0x00-0xff 的 int 类型,如果大于 0xff 的值将被直接截断

例如

```
memory foo;
foo=foo+0xff;
foo=foo+foo;
//foo 最终为 0xffff;
```

同时, memory 类型允许访问其元素,例如

```
memory foo=@0102@;
int a;
a=foo[0];
//a 的值为 1
foo[0]=0xff;
//foo 变换为 ff 02;
```

set 类型访问

直接访问 set 类型元素使用符合.,访问指针类型 set 使用->运算符

例如定义下列集合

```
set mySet
{
  int a,b;
}
```

可以进行如下访问

```
mySet theSet;
theSet.a=99;
mySet *pSet=&theSet;
pSet->b=99;
```

关键字类型

1. int(<Expression>) 转换为 int 类型,参数可以是一个整形,浮点,字符串类型
2. float(<Expression>)转换为 float 类型, 参数可以是一个整形,浮点,字符串类型
3. string(<Expression>)转换为 string 类型, 参数可以是一个整形,浮点,字符串,或内存类型
4. memory(<Expression>)转换为 memory 类型,参数可以是一个字符串或内存类型;
5. strlen(<Expression>) 内建函数,返回 string 的长度
6. memlen(<Expression>) 内建函数,返回 memory 的长度
7. sin(<Expression>) 内建函数,sin 三角函数
8. cos(<Expression>) 内建函数,cos 三角函数
9. break 跳出当前的循环结构,可以是一个 while for compare 结构
10. return <Expression> 函数返回值

函数调用

PainterScript 的函数调用方式为

FUNCTIONCALL_STATEMENT::=<Function>(<.....>);

例如下面是一个合法的函数调用

```
add(1,2);
```

语法结构

if 条件判断语句

if 语句的设计与 C 语言保持一致

IF_STATEMENT::=if(<Expression>)

{<.....>}

else

{

<.....>

}

或者是

if(<Expression>)

<Expression>

else

<Expression>

当 if 判断块中的 Expression 计算结果不为 0 时,执行对于语句块,否则跳转到 else 中执行,例如,下面是一个合法的 if 语句

```
if(a==1)
{
    print("a is 1");
}
else
print ("a is not 1");
```

while 循环语句

while 语句的设计与 C 语言保持一致

WHILE_STATEMENT::=while(<Expression>)

{<....>}

或者是

while(<Expression>)

<Expression>

当 while 判断块中的 Expression 计算结果不为 0 时,循环执行器对应语句块,例如下面是合法的 while 语句

```
while(1)
{
    //do something
}
```

for 循环语句

for 语句的设计与 C 语言保持一致

FOR_STATEMENT::=for(<init>;<condition>;<extend>)

{<....>}

或者是

for(<init>;<condition>;<extend>)

<Expression>

其中,init 表达式会先执行一次,然后判断 condition 表达式是否为 1,如果是,执行对于语句块,否则跳出 for 循环,最后执行 extend,再次判断 condition

compare 语句

compare 语句是 PainterScript 中独有的语法结构其语法格式为

```
compare(<VARIABLE>)  
{  
  with <Expression1;Expression2;....>  
  {  
    <.....>  
  }  
}
```

即 VARIABLE 将会与 with 对于的表达式进行比较,如果满足 with 语句中任意表达式对等,则执行 with 语句块中的语句,compare 语句设计用于简化 if-elseif-else..结构,并设计为 switch 语句的替代语句.

下面的 compare 语句都是合法的

```
int a;  
compare(a) with(1;2;3) print("a is 1 or 2 or 3.");
```

```
compare(a)  
{  
  with(1)  
  {  
    print("a is 1");  
  }  
  with(2;1+2)  
  {  
    print("a is 2 or 3");  
  }  
  print("compare done");  
}
```

函数定义

PainterScript 的函数定义与 C 语言基本一致

```
FUNCTION_STATEMENT::=<VARIABLE_TYPE> <FunctionName>(<.....>)
```

在 PainterScript 中,函数必须被前置声明后使用,同时在 PainterScript 中,函数返回值不允许使用 Set 类型,其参数也不允许使用 set 类型,如果需要使用 set 类型引用,必须使用 set 指针进行传递,使用 set 指针作为返回值是合法的但需要注意局部 set 在函数执行结束后其占用的资源

将被释放,这将应用该 set 将导致访问不可预知的数据.

PainterScript 中,函数分三种类型(内定函数,导入函数(远程函数),导出函数),并使用两个关键字进行修饰,如果不使用修饰关键字,该函数将被当做是一个内定函数,作为一个内定函数,其在 PainterScript 中必须被实现

例如,下面是一个合法的内定函数定义

```
int MyFunction(int a,float b,string c,MySet *pSet);
```

内定函数仅允许 PainterScript 脚本内部进行调用,虚拟机无法找到内定函数,如果你希望虚拟机可以调用该函数,它必须被声明为导出函数;

导入函数(远程函数)即以 host 进行修饰的函数,例如,下面是一个导入函数的声明

```
host void print(string msg);
```

导入函数即虚拟机中实现的函数,所有的导入函数都不能在 PainterScript 编写实现,例如

```
host void print(string msg)
```

```
{  
    //do something  
}
```

是不合法的,导入函数的名称必须与虚拟机中注册的名称保持一致,如果调用一个不存在的导入函数,将会导致虚拟机报错.

导出函数即以 export 进行修饰的函数,导入函数可以在 PainterScript 中被内部调用或者被虚拟机中调用,例如,下面是一个导入函数的声明

```
export void main(string msg);
```

如果你希望虚拟机可以调用该函数,那么它必须被声明为导出函数,否则它无法被虚拟机所发现调用。作为一个导出函数,其在 PainterScript 中必须被实现。

下面是一个函数的示范代码

```
int add(int a,int b)
```

```
{  
    return a+b;  
}
```

```
export int ss_main()
```

```
{  
    return add(1,2);  
}
```

内联汇编

PainterVM 允许使用内联汇编,格式为

```
_asm
{
    //内联的汇编语句
}
```

在内联汇编中,允许对变量进行访问,例如

```
int l;
_asm
{
    Mov l,1;
}
```

是被允许的

PainterScript 简化编译器

初始化编译器

函数名	px_bool PX_CompilerInitialize(px_memorypool *mp,PX_Compiler *compiler);
说明	初始化编译器
参数	mp 编译器部署内存池 compiler 编译器实例
返回值	如果成功返回 PX_TRUE,否者 PX_FALSE

添加源代码

函数名	px_bool PX_CompilerAddSource(PX_Compiler *compiler,const px_char script[]);
说明	将一段源代码加入编译器中,这个函数可以多次调用以加入多个源代码
参数	compiler 编译器实例 script 代码
返回值	如果成功返回 PX_TRUE,否者 PX_FALSE

编译源代码

函数名	px_bool PX_CompilerCompile(PX_Compiler *compiler,px_memory *bin,const px_char entryScript[]);
说明	编译源代码
参数	compiler 编译器实例 bin 输出的二进制指令流 entryScript 入口函数名
返回值	如果成功返回 PX_TRUE,否者 PX_FALSE

释放编译器

函数名	px_void PX_CompilerFree(PX_Compiler *compiler);
说明	释放编译器
参数	compiler 编译器实例
返回值	-

PainterScript Asm 汇编语法

标志

汇编引擎以如下标号作为跳转标志

Name:

其中,Name 为合法变量助记符

可以对其前置 FUNCTION 关键字

FUNCTION Name:

其效果和 Name:等价

如果需要对标志进行导出以方便其它语言进行调用,必须前缀 EXPORT 关键字

语法规则为

EXPORT FUNCTION Name:

汇编指令集

mov

赋值指令,将 op2 的值赋值给 op1

```
mov [reg,local,global],[num,string,reg,local,global]
```

.

add

加法指令, $op1=op1+op2$

```
add [reg,local,global],[num,reg,local,global]
```

.

sub

减法指令, $op1=op1-op2$

```
sub [reg,local,global],[num,reg,local,global]
```

.

neg

符号求反指令, $op1=-op1$

```
neg [reg,local,global]
```

div

除法指令, $op1=op1/op2$

```
div [reg,local,global],[num,reg,local,global]
```

.

mul

乘法指令, $op1 = op1 * op2$

mul [reg,local,global],[num,reg,local,global]

.

mod

余数指令,两个操作数必须为整数 $op1 = op1 \% op2$

mod [reg,local,global],[int,reg,local,global]

shl

左移位指令,两个操作数必须为整数 $op1 = op1 \ll op2$

shl [reg,local,global],[int,reg,local,global]

shr

右移位指令,两个操作数必须为整数 $op1 = op1 \gg op2$

shr [reg,local,global],[int,reg,local,global]

and

与运算指令, $op1 = op1 \& op2$

and [reg,local,global],[num,reg,local,global]

or

或运算指令, $op1 = op1 | op2$

or [reg,local,global],[num,reg,local,global]

.

xor

异或运算指令 $op1 = op1 \wedge op2$

xor [reg,local,global],[num,reg,local,global]

.

inv

取反指令, $op1 = \sim op1$

inv [reg,local,global]

not

逻辑非指令 $op1 = !op1$

not [reg,local,global]

andl

逻辑与指令 $op1 = op1 \& op2$

andl [reg,local,global],[num,reg,local,global]

orl

逻辑或指令 $op1 = op1 \vee op2$

andl [reg,local,global],[num,reg,local,global]

pow

阶乘指令($op1$ 为底数, $op2$ 为指数,结果在 $op1$ 中) $op1 = op1_{op2}$

pow [reg,local,global],[num,reg,local,global]

sin

正弦函数 op1=sin(op2)

sin [reg,local,global],[num,reg,local,global]

cos

余弦函数 op1=cos(op2)

cos [reg,local,global],[num,reg,local,global]

int

强制类型转换为 int 型(原类型 float)

int [reg,local,global]

flt

强制类型转换为 float 型

flt [reg,local,global]

strlen

字符型长度指令

op1=strlen(op2)

strlen [reg,local,global],[reg,local,global,string]

strcat

字符型拼接指令

strcat(op1,op2)

strcat [reg,local,global],[int,reg,local,global,string]

strrep

字符串替换函数

将 op1 存在的 op2 字符串替换为 op3 中的字符串, 注意:op2 op3 必须为字符串类型

strrep [reg,local,global],[reg,local,global,string],[reg,local,global,string]

strchr

将 op2 在索引 op3 中的字存储在 op1 中, 注意:op2 必须为字符串类型

strchr [reg,local,global],[reg,local,global,string],[reg,local,global,int]

strtoi

将 op2 转换为整数保存在 op1 中,注意:op2 必须为字符串类型

strtoi [reg,local,global],[reg,local,global,string]

strtof

将 op2 转换为浮点数保存在 op1 中,注意:op2 必须为字符串类型

strtof [reg,local,global],[reg,local,global,string]

strfri

将 op2 整数类型转换为字符串类型保存在 op1 中

strfri [reg,local,global],[reg,local,global,int]

strfrf

将 op2 浮点类型转换为字符串类型保存在 op1 中

strfrf [reg,local,global],[reg,local,global,float]

strset

将 op1 所在字符串索引为 op2 int 的字符替换为 op3

如果 op3 为一个 int,则取 asc 码(第八位 1 字节),如果 op3 为一个字符串,则取第一个字母

strset [reg,local,global],[reg,local,global,int],[reg,local,global,string,int]

strfind

在 op2 字符串中查找 op3 子字符串,如果找到了 op1 等于其开始索引,否则等于-1

如果 op3 为一个 int,则取 asc 码(第八位 1 字节),如果 op3 为一个字符串,则取第一个字母

strset [reg,local,global],[reg,local,global,string],[reg,local,global,string]

strtmem

将 op1 字符串类型转换为内存类型

strfrf [reg,local,global]

asc

将 op2 的第一个字母以 asc 码的形式

asc [reg,local,global],[reg,local,global,string]

membyte

将 op3 内存类型对应 op2 索引复制到 op1 中,这个类型是一个 int 类型(小于 256)

membyte [reg,local,global],[reg,local,global,int],[reg,local,global,memory]

memset

设置 op1 对应 op2 索引的内存为 op3

memset [reg,local,global],[reg,local,global,int],[reg,local,global,int]

memtrm

将 op1 内存进行裁剪,其中,op2 为开始位置,op2 为大小

memcpy [reg,local,global],[reg,local,global,int],[reg,local,global,memory]

memfind

查找 op2 对应于 op3 内存所在的索引位置,返回结果存储在 op1 中,如果没有找到,op1 将会置为-1

memfind [reg,local,global],[reg,local,global,memory],[reg,local,global,memory]

memlen

将 op2 的内存长度存储在 op1 中

memlen [reg,local,global],[reg,local,global,memory]

memcat

将 op2 的内存拼接到 op1 的尾部

memcat [reg,local,global],[int,reg,local,global,memory]

memtstr

将 op1 内存类型转换为字符串类型,如果 op1 的内存结尾不为 0,将会被强制置为 0
memtstr [reg,local,global]

datacpy

复制虚拟机 data 数据,从地址 op2 到地址 op1,长度为 op3
datacpy [reg,local,global,int], [reg,local,global,int], [reg,local,global,int]

jmp

跳转指令 跳转到 op1 地址
jmp [reg,num,local,global,label]

je

条件跳转,当 op1 等于 op2,跳转到 op3
je [num,string,reg,local,global],[num,string,reg,local,global],[reg,int,local,global,label]

jne

条件跳转,当 op1 不等于 op2,跳转到 op3
jne [num,string,reg,local,global],[num,string,reg,local,global],[reg,int,local,global,label]

jl

条件跳转,当 op1 小于 op2,跳转到 op3

jl [num,string,reg,local,global],[num,string,reg,local,global],[reg,int,local,global,label]

jle

条件跳转,当 op1 小于等于 op2,跳转到 op3

jle [num,string,reg,local,global],[num,string,reg,local,global],[reg,int,local,global,label]

jg

条件跳转,当 op1 大于 op2,跳转到 op3

jg [num,string,reg,local,global],[num,string,reg,local,global],[reg,int,local,global,label]

jge

条件跳转,当 op1 大于等于 op2,跳转到 op3

jge [num,string,reg,local,global],[num,string,reg,local,global],[reg,int,local,global,label]

lge

逻辑比较指令,当 op2 等于 op3 时将 op1 置 1,否则为 0

lge [reg,local,global], [num,string,reg,local,global] , [num,string,reg,local,global]

lgne

逻辑比较指令,当 op2 等于 op3 时将 op1 置 0,否则为 1

lgne [reg,local,global], [num,string,reg,local,global] , [num,string,reg,local,global]

lgz

逻辑比较指令,当 op1 等于 0 时将 op1 置 1,否则为 0

lgz [reg,local,global]

lggz

逻辑比较指令,当 op1 大于 0 时将 op1 置 1,否则为 0

lggz [reg,local,global]

lggez

逻辑比较指令,当 op1 大于等于 0 时将 op1 置 1,否则为 0

lggez [reg,local,global]

lglz

逻辑比较指令,当 op1 小于 0 时将 op1 置 1,否则为 0

lglz [reg,local,global]

lglez

逻辑比较指令,当 op1 小于等于 0 时将 op1 置 1,否则为 0

lglez [reg,local,global]

call

调用指令,如果 op1 是本地地址则将当期下一条指令地址压栈,然后跳转到 op1,如果 op1 是一个 host 地址,则该 call 为一个 hostcall,hostcall 不会将返回地址压栈

call [reg,int,local,global,label,host]

*Host Call 的返回值在 r[0]中

*由被调用者清理堆栈

push

将 op1 压栈 sp-1,stack[0]=op1

push [num,reg,local,global,string,label]

pop

出栈，并将该值

pop [reg,local,global]

adr

取堆栈的绝对地址,返回该堆栈的绝对地址

ADR [reg,local,global], [local,global]

popn

将 op1 个元素出栈

popn [reg,local,global]

ret

返回,pop 一个返回地址,跳转到该地址.

wait

等待一个信号量置为 0,否则这个虚拟机实例将被暂时挂起(但并不影响 suspend 标准位),

在每个虚拟机实例中都有 16 个信号量,通过 signal 指令对这些信号量进行设置

signal

等待 op1 对应索引的信号量置为 op2, 在每个虚拟机实例中都有 16 个信号量,这意味着 op1 的范围是 0-15,当一个信号量被设置为非 0 值时,执行 wait 指令后改虚拟机实例会被阻塞,直到这个信号量被置为 0 时才能继续执行后续指令

bpx

如果启动了调试器,将会在该指令上断点,否者作为一个空指令

nop

空指令

表达式

在文件开头以

.stack m

.global n

指定堆栈元素大小

注:以下参数表达 reg 表示寄存器,local 表示局部变量,global 表示全局变量,num 表示数值,label 表示标签或函数,host 表示引擎 API,int 表示整数,float 表示浮点数,string 表示字符串类型

其中 local 表示对当前堆栈操作 local[0]表示栈顶,local[1]表示栈顶往后的第一个指针

global 表示对当前堆操作,按照 global[n] 表示 , n 为索引量

op1 表示参数 1,op2 表示参数 2.... 以此类推

在 Label 中以 \$ 标志做前缀,表示这是一个 hostcall

{ } 包含符表示为一内存类型例如 {02AFEFCDDDEE},两个字符为一字节,必须 2 字节对齐

""包含符表示为一字符串类型,支持\n 换行\t 对齐转义
"表示这个字符代表的 asc 码值

PainterScript Asm 汇编器

编译汇编代码

函数名	px_bool PX_ScriptAsmCompile(px_memorypool *mp,px_byte *asmcode,px_memory *binmemory);
说明	编译一个脚本汇编文件
参数	mp 临时内存池 asmcode 以\0 结束的汇编代码 binmemory 输出 bin 内存指针
返回值	如果成功返回 PX_TRUE,否者返回 PX_FALSE

PainterVM 虚拟机

vm 实例

```
typedef struct __PX_ScriptVM_Instance
{
    px_int VM_memsize; //虚拟机使用元内存大小
    px_int stringsize; //字符串常量内存
    px_int memorysize; //二进制常量内存大小
    px_int binsize; //shellcode 内存大小
    px_int funcCount; //导出函数数量
    px_int hostCount; //host 函数数量
    px_bool Suspend; //是否被挂起
    px_int T; //当前执行线程索引
    px_byte *_bin; //shellcode 指针
    px_byte *_memory; //二进制常量指针
    px_char *_string; //字符串常量指针
    px_memorypool *mp; //归属内存池
```

```

PX_SCRIPT_EXPORT_FUNCTION *_func;//导出函数表
PX_SCRIPT_ASM_HOST_NODE *_host;//hostcall 表

PX_SCRIPTVM_VARIABLE *_mem;//元内存指针
px_bool debug; //是否处于调试状态
px_bool bp_next;//下一条语句断点
px_int bp_IP;//断点指令地址
}PX_ScriptVM_Instance;

```

host 函数定义

函数名	typedef px_bool (*PX_ScriptVM_Function_Modules)(PX_ScriptVM_Instance *Ins,px_void *userptr);
说明	PainterScript host 函数声明标准
参数	PX_ScriptVM_Instance 实例 userptr 用户指针
返回值	如果成功返回 PX_TRUE,否者返回 PX_FALSE
备注	通过 #define PX_ScriptVM_STACK(Ins,i) ((Ins)->_mem[(Ins)->BP+i]) 访问传入参数,其中,第一个参数下标为 0,第二个为 1 以此类推 通过 #define PX_ScriptVM_RETURN(Ins) ((Ins)->R[1]) 设定 host 函数返回类型

实例初始化

函数名	px_bool PX_ScriptVM_InstanceInit(PX_ScriptVM_Instance *Ins,px_memorypool *mp,px_byte *code,px_int size);
说明	初始化一个虚拟机实例
参数	PX_ScriptVM_Instance 实例 mp 内存池 code 虚拟机指令流 size 指令流大小
返回值	如果成功返回 PX_TRUE,否者返回 PX_FALSE

执行虚拟机函数

函数名	px_bool PX_ScriptVM_InstanceRunFunction(PX_ScriptVM_Instance *Ins,px_int threadID, px_char *func,px_int paramcount,...);
说明	通过函数名执行脚本函数,这个函数被调用后,对应函数会立即执行直到

	其结束或被信号量中断
参数	PX_ScriptVM_Instance 实例 thread 由 thread 索引线程执行该函数 func 函数名 paramcount 参数个数 ... 由特定函数初始化的函数类型
返回值	如果成功返回 PX_TRUE,否者返回 PX_FALSE

函数名	px_bool PX_ScriptVM_InstanceRunFunctionIndex(PX_ScriptVM_Instance *Ins,px_int threadID, px_int funcIndex,px_int paramcount,...);
说明	通过函数索引执行脚本函数, 这个函数被调用后,对应函数会立即执行直到其结束或被信号量中断
参数	PX_ScriptVM_Instance 实例 thread 由 thread 索引线程执行该函数 funcIndex 函数索引 paramcount 参数个数 ... 由特定函数初始化的函数类型
返回值	如果成功返回 PX_TRUE,否者返回 PX_FALSE

函数名	px_bool PX_ScriptVM_InstanceBeginThreadFunction(PX_ScriptVM_Instance *Ins,px_int threadID, px_char *func,px_int paramcount,...);
说明	通过函数名创建线程函数,这个函数被调用后,对应函数并不会立即执行,需要调用 PX_ScriptVM_InstanceRunThread 来执行对应线程
参数	PX_ScriptVM_Instance 实例 thread 由 thread 索引线程执行该函数 func 函数名 paramcount 参数个数 ... 由特定函数初始化的函数类型
返回值	如果成功返回 PX_TRUE,否者返回 PX_FALSE

函数名	px_bool PX_ScriptVM_InstanceBeginThreadFunctionIndex(PX_ScriptVM_Instance *Ins,px_int threadID, px_int *funcIndex,px_int paramcount,...);
说明	通过函数索引创建线程函数,这个函数被调用后,对应函数并不会立即执行,需要调用 PX_ScriptVM_InstanceRunThread 来执行对应线程
参数	PX_ScriptVM_Instance 实例 thread 由 thread 索引线程执行该函数 funcIndex 函数索引 paramcount 参数个数 ... 由特定函数初始化的函数类型
返回值	如果成功返回 PX_TRUE,否者返回 PX_FALSE

运行虚拟机

函数名	PX_SCRIPTVM_RUNRETURN PX_ScriptVM_InstanceRunThread(PX_ScriptVM_Instance *Ins ,px_int tick);
说明	执行虚拟机实例直到当前线程其指令片耗尽
参数	PX_ScriptVM_Instance 实例 tick 指令片(执行指令数),如果这个值小于 0,该线程将一直执行到线程退出或信号量中断
返回值	PX_SCRIPTVM_RUNRETURN 有以下几种返回方式 PX_SCRIPTVM_RUNRETURN_ERROR =0,虚拟机执行发生运行时错误(非法指令?) PX_SCRIPTVM_RUNRETURN_TIMEOUT,时间片耗尽返回 PX_SCRIPTVM_RUNRETURN_END,线程结束 PX_SCRIPTVM_RUNRETURN_SUSPEND,线程被挂起 PX_SCRIPTVM_RUNRETURN_WAIT,信号量中断返回

线程上下文切换

函数名	px_bool PX_ScriptVM_InstanceThreadSwitch(PX_ScriptVM_Instance *Ins,px_int T);
说明	切换线程上下文
参数	PX_ScriptVM_Instance 实例 T 线程 ID
返回值	如果成功返回 PX_TRUE,否者返回 PX_FALSE

终止线程

函数名	px_void PX_ScriptVM_ThreadStop(PX_ScriptVM_Instance *Ins,px_int ThreadId);
说明	强制终止实例的一个线程,这个函数将在脚本函数最后 ret 时调用,它不会清除寄存器中的值.
参数	PX_ScriptVM_Instance 实例 T 线程 ID
返回值	-

函数名	px_void PX_ScriptVM_ThreadClear(PX_ScriptVM_Instance *Ins,px_int ThreadId);
说明	清理一个线程中的堆栈和寄存器,调用该函数必须在该线程没有执行时调用
参数	PX_ScriptVM_Instance 实例 T 线程 ID
返回值	-

挂起线程

函数名	px_void PX_ScriptVM_ThreadSuspend (PX_ScriptVM_Instance *Ins,px_int ThreadId);
说明	挂起实例的一个线程
参数	PX_ScriptVM_Instance 实例 T 线程 ID
返回值	-

恢复线程

函数名	px_void PX_ScriptVM_ThreadResume (PX_ScriptVM_Instance *Ins,px_int ThreadId);
说明	恢复被挂起实例的一个线程
参数	PX_ScriptVM_Instance 实例 T 线程 ID
返回值	-

注册 host 函数

函数名	px_bool PX_ScriptVM_RegistHostFunction(PX_ScriptVM_Instance *Ins,px_char *name,PX_ScriptVM_Function_Modules funcModules,px_void *userptr);
说明	注册一个 host 函数
参数	PX_ScriptVM_Instance 实例

	name 注册函数名 funcModules 函数指针 userptr 用户指针
返回值	如果成功返回 PX_TRUE,否者返回 PX_FALSE

释放虚拟机

函数名	px_bool PX_ScriptVM_InstanceFree(PX_ScriptVM_Instance *Ins);
说明	释放一个虚拟机实例
参数	PX_ScriptVM_Instance 实例
返回值	如果成功返回 PX_TRUE,否者返回 PX_FALSE

类型访问

```
#define PX_ScriptVM_STACK(Ins,i) 访问虚拟栈元
#define PX_ScriptVM_HOSTPARAM(Ins,i) 访问虚拟栈元,该宏用于 Host 访问 hostcall 函数调用参数,其中索引 0 表示第一个参数
#define PX_ScriptVM_LOCALPARAM(Ins,i) 访问虚拟栈元,该宏用于访问脚本内部函数调用参数,其中索引 0 表示第一个参数
#define PX_ScriptVM_GLOBAL(Ins,i) 访问虚拟堆元
#define PX_ScriptVM_RETURN(Ins) 访问寄存器 R1
```

弹出栈

函数名	px_void PX_ScriptVM_POPN(PX_ScriptVM_Instance *Ins,px_int n);
说明	弹出 n 字节栈元
参数	PX_ScriptVM_Instance 实例 n 栈元数目
返回值	-

host 返回值

函数名	px_voidPX_ScriptVM_RET(PX_ScriptVM_Instance *Ins,PX_SCRIPTVM_VARIABLE ret);
说明	设置 R0 寄存器(返回值寄存器),该函数主要用于 hostcall 中的返回值
参数	PX_ScriptVM_Instance 实例 ret 值
返回值	-

压栈

函数名	px_voidPX_ScriptVM_PUSH(PX_ScriptVM_Instance *Ins,PX_SCRIPTVM_VARIABLE val);
说明	元数据压栈
参数	PX_ScriptVM_Instance 实例 val 元数据
返回值	-

函数名	PX_SCRIPTVM_VARIABLE PX_ScriptVM_Varibale_int(px_int _int); PX_SCRIPTVM_VARIABLE PX_ScriptVM_Varibale_float(px_float _float); PX_SCRIPTVM_VARIABLE PX_ScriptVM_Varibale_string(px_string _ref_string); PX_SCRIPTVM_VARIABLE PX_ScriptVM_Varibale_memory(px_memory _ref_memory); PX_SCRIPTVM_VARIABLE PX_ScriptVM_Varibale_const_string(px_char *buffer); PX_SCRIPTVM_VARIABLE PX_ScriptVM_Varibale_const_memory(px_byte *buffer,px_int _size);
说明	构造对应类型元数据
参数	~
返回值	构造的元数据

命令调试系统

PainterEngine PainterVM 提供一个基于命令的简单调试器,提供如下三个接口函数供使用

1.启用调试器

```
px_int PX_ScriptVM_DebuggerEnable(PX_ScriptVM_Instance
*Ins,PX_ScriptVM_DebuggerPrint _printFunc,PX_ScriptVM_DebuggerCommand
_cmdFunc);
```

通过调用该函数来启用一个调试器,其中 PX_ScriptVM_DebuggerPrint 是一个指向 printf 函数的函数指针,用于输出调试信息

PX_ScriptVM_DebuggerCommand 指向一个类似于 scanf 的函数指针用于中断输入命令

2.调试器命令

调试器支持以下命令

B IP 断点,BP 后接 addr 例如 B 100

N 单步执行下一条指令

C 继续执行

U 反汇编代码,U 后接接下来的反汇编指令数目 例如 U 1400

R 查看寄存器,例如 R 1 表示查看寄存器 1 的值

G 查看 Global 数据,例如 G 1 表示查看 Global[1]的数据

L 查看 local 数据,例如 L 1 表示查看 Local[1]数据,Local 数据本质是 Global[BP+oft]

Obj 静态 3D 模型加载器

加载器初始化

函数名	px_bool PX_3D_ObjectDataInitialize(px_memorypool *mp,PX_3D_ObjectData *ObjectData);
说明	加载器初始化
参数	mp 内存池 ObjectData 加载器对象
返回值	如果成功返回 PX_TRUE,否者 PX_FALSE

解析 Object 数据

函数名	px_bool PX_3D_ObjectDataLoad(PX_3D_ObjectData *ObjectData,const px_byte *data,px_int size);
说明	解析 Obj 模型数据
参数	ObjectData 加载器对象 data Obj 数据指针 size 数据长度
返回值	如果成功返回 PX_TRUE, 否则 PX_FALSE

数据转换到 RenderList

函数名	px_bool PX_3D_ObjectDataToRenderList(PX_3D_ObjectData *ObjectData,PX_3D_RenderList *renderList);
说明	将数据转换到 PainterEngine 3D 可渲染的 RenderList
参数	ObjectData 加载器对象 renderList PX_3D_RenderList 对象
返回值	如果成功返回 PX_TRUE, 否者 PX_FALSE

释放加载器

函数名	px_void PX_3D_ObjectDataFree(PX_3D_ObjectData *ObjectData);
说明	释放加载器
参数	ObjectData 加载器对象
返回值	

2dx 动画描述语言

2dx 动画文件是 PainterEngine 专门为逐帧动画设计的动画文件,在有文件的文件系统中,其常常以 .2dx 作为后缀,同时 2dx makechain 必须运行在有文件系统的操作系统中

2dx 动画文件是一个依据简单描述脚本经过编译而成的一个动画文件,下面是一个 2dx 脚本

```
texture "1.traw" f1
```

```
texture "2.traw" f2
```

```
texture "3.traw" f3
```

```
animation clock
```

```
loop -1
```

```
tag begin
```

```
frame f1
```

```
sleep 100
```

```
frame f2
```

```
sleep 90
```

```
frame f3
```

```
sleep 80
```

```
goto begin
```

```
end
```

指令

2dx 脚本支持以下指令

Texture 纹理映射指令,其负责将脚本存在目录下的纹理文件映射为一助记符,在脚本中就可以通过该助记符来使用该纹理,其格式为

Texture “文件路径” 助记符

Frame 纹理显示指令,将助记符映射的纹理显示到当前帧上,其格式为

Frame 助记符

Sleep 延迟指令,延迟一段时间,其单位是毫秒,其格式为

Sleep 毫秒时间数

Tag 标记,其不是一个指令,它将搭配 Goto 指令进行使用,其使用方式为

Tag 标记符

Animation 标记,其不是一个指令,它将搭配 Goto 指令进行使用,实际上他和 tag 是同一作用的不同写法,用法为

Animation 标记符

Loop 指令,修改 Loop 寄存器的值,他配合 Goto 指令进行使用,其用法为

Loop 值

Goto 条件跳转指令,跳转到标号的指令执行,在执行前其会检查 Loop 寄存器,如果 Loop 寄存器的值不为 0,则跳转,同时假如 loop 寄存器的值不为-1,其会将 loop 寄存器的值-1,格式为

Goto 标记符

END 结束指令,用于通知动画播放器动画已经播放结束,是一个特殊指令,用法为
END

编译

要编译一起 2dx 动画文件,你需要一个 2dx 脚本及对应纹理,并且将该脚本与纹理放在同一个目录中(否者你需要使用相对路径重新定位其在文件系统中的具体位置)

PainterEngine_2dxMakeTool 是 PainterEngine 附带的 2dx 动画编译器工具配件,其可以在带文件系统的 windows 及 Linux 衍生版本中编译成可执行的二进制文件并用于 2dx 动画编译

其使用方法为

PainterEngine_2dxMakeTool 脚本路径 编译输出路径

如果脚本正确它将会输出一个 2dx 动画文件,你可以使用 painterEngine 将它加载到引擎当中(注意,因为该工具并不附带脚本错误提示器,所以你必须确保你的脚本编写正确,不过不用担心,2dx 的脚本语言本身较为简单,并不会给你编写带来多大困扰).

2dx 动画

2dx library

通过 2dx 数据创建一个动画库

需要创建一个 2dx 动画,首先要加载 2dx 动画数据,并通过该数据创建一个库文件

函数名	px_bool PX_AnimationLibraryCreateFromMemory(px_memorypool *mp,px_animationlibrary *panimationLib,px_byte *_2dxBuffer,px_uint size);
说明	通过 2dx 数据创建一个动画库
参数	mp 内存池 panimationLib px_animationlibrary 库指针 _2dxBuffer 2dx 数据 size 2dx 数据的长度
返回值	如果成功返回 PX_TRUE,否者返回 PX_FALSE

释放动画库文件

函数名	px_void PX_AnimationLibraryFree(px_animationlibrary *panimationLib);
说明	释放动画库文件
参数	panimation
返回值	如果成功返回 PX_TRUE,否者返回 PX_FALSE

取得动画库图像宽度

函数名	px_int PX_AnimationLibraryGetFrameWidth(PX_Animationlibrary *panimationLib,px_int frameIndex);
说明	取得动画库图像宽度
参数	panimationLib 动画库 frameIndex 帧索引
返回值	图像宽度

取得动画库图像高度

函数名	px_int PX_AnimationLibraryGetFrameHeight(PX_Animationlibrary *panimationLib,px_int frameIndex);
说明	取得动画库图像高度
参数	panimationLib 动画库 frameIndex 帧索引
返回值	图像高度

2dx animation

2dx animation 动画是 2dx library 的实例化,每一个 2dx animation 都有自己独立的动画实例
每一个 2dx animation 都必须连接到一个 2dx animation,并且通过 update 进行其动画的帧更新,在此之后,它就能渲染到对应的 surface 实现动画的播放了

创建一个动画

函数名	px_bool PX_AnimationCreate(px_animation *animation,px_animationlibrary *linker);
说明	通过 2dx 库数据创建一个动画
参数	animation 动画对象 linker px_animationlibrary 连接的库指针
返回值	如果成功返回 PX_TRUE,否者返回 PX_FALSE

取得动画对应动画库中的动画类型数量

函数名	px_int PX_AnimationGetAnimationsCount(PX_Animation *animation);
说明	取得动画对应动画库中的动画类型数量
参数	animation 动画对象
返回值	动画对应动画库中的动画类型数量

设置当前播放动画

函数名	px_bool PX_AnimationSetCurrentPlayAnimation(PX_Animation *animation,px_int i);
说明	设置当前播放动画
参数	animation 动画对象 I 当前播放动画索引
返回值	如果成功返回 PX_TRUE,否者 PX_FALSE

函数名	px_bool PX_AnimationSetCurrentPlayAnimationByName(PX_Animation *animation,px_char *name);
说明	设置当前播放动画
参数	animation 动画对象 name 动画名
返回值	如果成功返回 PX_TRUE,否者 PX_FALSE

动画是否结束

函数名	px_bool PX_AnimationIsEnd(PX_Animation *animation);
说明	当前播放动画是否已经结束
参数	animation 动画对象
返回值	如果是返回 PX_TRUE,否者 PX_FALSE

取得动画当前纹理

函数名	px_texture *PX_AnimationGetCurrentTexture(PX_Animation *animation);
说明	取得动画当前纹理
参数	animation 动画对象
返回值	如果成功返回纹理指针,否者返回 PX_NULL

释放 2dx 动画对象

函数名	px_void PX_AnimationFree(px_animation *animation);
说明	释放 2dx 动画对象
参数	animation 动画对象
返回值	-

重置该 2dx 动画对象

函数名	px_void PX_AnimationReset(px_animation *animation);
说明	重置该 2dx 动画对象
参数	animation 动画对象
返回值	-

更新该动画对象

函数名	px_void PX_AnimationUpdate(px_animation *animation,px_uint elapsed);
-----	--

说明	更新该动画对象
参数	animation 动画对象 elapsed 距离上一次更新经过的时间(毫秒级)
返回值	-

渲染动画对象

函数名	px_void PX_AnimationRender(px_surface *psurface,px_animation *animation, px_point position,PX_ALIGN refPoint,PX_TEXTURERENDER_BLEND *blend);
说明	渲染动画对象
参数	animation 动画对象 psurface 渲染表面 position 偏移量 refPoint 渲染参考点 blend blend 属性,参考 PX_TEXTURERENDER_BLEND 定义
返回值	-

以一个旋转角度渲染动画对象

函数名	px_void PX_AnimationRenderRotation(px_surface *psurface,px_animation *animation, px_point position,px_int angle,PX_ALIGN refPoint,PX_TEXTURERENDER_BLEND *blend);
说明	渲染动画对象
参数	animation 动画对象 psurface 渲染表面 position 偏移量 angle 旋转角度 refPoint 渲染参考点 blend blend 属性,参考 PX_TEXTURERENDER_BLEND 定义
返回值	-

以一个比例和方向渲染动画对象

函数名	px_void PX_AnimationRender(px_surface *psurface,px_animation
-----	--

	*animation, px_point position,px_float scale,px_point direction,PX_ALIGN refPoint,PX_TEXTURERENDER_BLEND *blend);
说明	渲染动画对象
参数	animation 动画对象 psurface 渲染表面 position 偏移量 scale 比例 direction 方向,默认方向为(1,0,0)即 x 方向 refPoint 渲染参考点 blend blend 属性,参考 PX_TEXTURERENDER_BLEND 定义
返回值	-

以一个向量方向渲染动画对象(初始方向是 x 方向)

函数名	px_void PX_AnimationRender_vector(px_surface *psurface,px_animation *animation, px_point position,px_point direction,PX_ALIGN refPoint,PX_TEXTURERENDER_BLEND *blend);
说明	以一个向量方向渲染动画对象(初始方向是 x 方向)
参数	animation 动画对象 psurface 渲染表面 position 偏移量 direction 方向向量, 默认方向为(1,0,0)即 x 方向 refPoint 渲染参考点 blend blend 属性,参考 PX_TEXTURERENDER_BLEND 定义
返回值	-

函数名	px_void PX_AnimationRenderEx(px_surface *psurface,px_animation *animation, px_point position,PX_ALIGN refPoint,PX_TEXTURERENDER_BLEND *blend,px_float scale,px_float rotation);
说明	渲染动画对象
参数	animation 动画对象 psurface 渲染表面 position 偏移量 refPoint 渲染参考点 blend blend 属性,参考 PX_TEXTURERENDER_BLEND 定义 scale 缩放比例 rotation 旋转角度
返回值	-

函数名	px_bool PX_AnimationIsEnd(px_animation *panimation);
说明	判断当前动画是否已经结束
参数	panimation 动画对象
返回值	-

函数名	px_rect PX_AnimationGetSize(px_animation *panimation)
说明	取得动画的建议最大宽高
参数	panimation 动画对象
返回值	rect 类型,其中 x,y 必定为 0 width,height 为建议宽度

PainterEngine LiveFramework

LiveFramework 是 PainterEngine 的类似于 live2D 的动画展示

导入 LiveFramework

函数名	px_bool PX_LiveFrameworkImport(px_memorypool *mp,PX_LiveFramework *plive,px_void *buffer,px_int size);
说明	导入 LiveFramework
参数	mp 构造内存池 plive LiveFramework 实例化 buffer live 数据 size 数据长度
返回值	如果成功返回 PX_TRUE,否者 PX_FALSE

LiveFramework 动画播放

函数名	px_bool PX_LiveFrameworkPlayAnimation(PX_LiveFramework *plive,px_int index);
说明	暂停播放 LiveFramework
参数	plive LiveFramework 实例化 index 动画索引
返回值	如果成功返回 PX_TRUE,否者 PX_FALSE

函数名	px_bool PX_LiveFrameworkPlayAnimationByName(PX_LiveFramework *plive,const px_char name[]);
说明	暂停播放 LiveFramework
参数	plive LiveFramework 实例化 name 动画名
返回值	如果成功返回 PX_TRUE,否者 PX_FALSE

函数名	px_bool PX_LiveFrameworkPlay (PX_LiveFramework *plive);
说明	继续播放 LiveFramework
参数	plive LiveFramework 实例化
返回值	如果成功返回 PX_TRUE,否者 PX_FALSE

LiveFramework 暂停播放

函数名	px_void PX_LiveFrameworkPause(PX_LiveFramework *plive);
说明	暂停播放 LiveFramework
参数	plive LiveFramework 实例化
返回值	

重置 LiveFramework

函数名	px_void PX_LiveFrameworkReset(PX_LiveFramework *plive);
说明	重置 LiveFramework
参数	plive LiveFramework 实例化
返回值	

停止 LiveFramework

函数名	px_void PX_LiveFrameworkStop(PX_LiveFramework *plive);
说明	重置 LiveFramework
参数	plive LiveFramework 实例化
返回值	

渲染 LiveFramework

函数名	px_void PX_LiveFrameworkRender(px_surface *psurface,PX_LiveFramework *plive,px_int x,px_int y,PX_ALIGN refPoint,px_dword elapsed);
说明	渲染 live
参数	psurface 渲染表面 plive LiveFramework 实例化 x,y 渲染坐标 refPoint 参考点 elapsed 距离上次渲染经过的毫秒时间
返回值	

PainterEngine Live

PainterEngine Live 是 PainterEngineLiveFramework 的引用对象

创建 Live

函数名	px_bool PX_LiveCreate(px_memorypool *mp,PX_LiveFramework *pLiveFramework,PX_Live *pLive);
说明	创建 Live
参数	mp 构造内存池 pLiveFramework 引用的 LiveFramework pLive Live 实例
返回值	如果成功返回 PX_TRUE,否者 PX_FALSE

释放 Live

函数名	px_void PX_LiveFree(PX_Live *pLive);
说明	释放 Live
参数	pLive Live 实例
返回值	

Live 动画播放

函数名	px_bool PX_LivePlayAnimation(PX_Live *plive,px_int index);
说明	暂停播放 Live
参数	plive live 实例化 index 动画索引
返回值	如果成功返回 PX_TRUE,否者 PX_FALSE

函数名	px_bool PX_LivePlayAnimationByName(PX_Live *plive,const px_char name[]);
说明	暂停播放 Live
参数	plive live 实例化 name 动画名
返回值	如果成功返回 PX_TRUE,否者 PX_FALSE

函数名	px_bool PX_LivePlay (PX_Live *plive);
说明	继续播放 Live
参数	plive live 实例化
返回值	如果成功返回 PX_TRUE,否者 PX_FALSE

Live 暂停播放

函数名	px_void PX_LivePause(PX_LiveFramework *plive);
说明	暂停播放 Live
参数	plive live 实例化
返回值	

重置 live

函数名	px_void PX_LiveReset(PX_LiveFramework *plive);
说明	重置 live
参数	plive live 实例化
返回值	

停止 live

函数名	px_void PX_LiveStop(PX_LiveFramework *plive);
说明	重置 live
参数	plive live 实例化
返回值	

渲染 live

函数名	px_void PX_LiveRender(px_surface *psurface,PX_LiveFramework *plive,px_int x,px_int y,PX_ALIGN refPoint,px_dword elapsed);
说明	渲染 live
参数	psurface 渲染表面 plive live 实例化 x,y 渲染坐标 refPoint 参考点 elapsed 距离上次渲染经过的毫秒时间
返回值	

粒子系统

粒子发射器参数

发射器参数名	说明
tex	粒子的纹理
position	发射器的位置
direction	发射器发射方向
deviation_rangAngle	发射方向的角度容差 (容差是指发射方向随机的+-容差角度范围)
velocity	粒子发射速度
deviation_velocity	速度容差 (容差是指发射速度随机的+-容差范围)
atomsze	粒子比例默认为 1.0 比例
deviation_atomsze	比例容差

rotation	粒子旋转角速度
deviation_rotation	旋转角速度容差
alpha	粒子透明度
deviation_alpha	透明度容差
hdrR	红色混合比,默认为 1.0
deviation_hdrR	红色混合比容差
hdrG	绿色混合比,默认为 1.0
deviation_hdrG	绿色混合比容差
hdrB	蓝色混合比,默认为 1.0
deviation_hdrB	蓝色混合比容差
sizeincrease	粒子每秒的自增比例,如果大于 1.0 粒子将不断增大,0-1.0 则不断缩小 不能为负数
alphaincrease	粒子每秒的透明度变换,0-1.0 粒子将会越来越透明,不能为负数
a	加速度向量
ak	阻尼系数,默认为 1.0,如果大于 1.0,速度将越来越快,小于 1.0 则越来越慢,不能为负数
alive	粒子的存在时间,毫秒为单位
generateDuration	粒子发射间隔,毫秒为单位,如果为 0 则一次性发射最大数量,不能为负数
maxCount	场上同时存在的最多粒子数量
launchCount	发射粒子数量,每发射一个粒子,这个值会-1,直到这个值为 0 不再发射,如果这个值一开始设置为一个小于 0 的值,那么,粒子将一直发射直到达到场上同时存在的最多粒子数量
Create_func	自定义函数,如果这个函数指针不为空,当创建了一个粒子后,将调用这个函数
Update_func	自定义函数,如果这个函数指针不为空,当更新了一个粒子物理信息后,将调用这个函数,可以对粒子的物理属性进一步修正

以默认值初始化一个发射器参数

函数名	px_void PX_ParticalLauncherInitializeDefault(PX_ParticalLauncher_InitializeInfo* info);
说明	以默认值初始化一个发射器参数
参数	要初始化的发射器参数
返回值	-

创建粒子系统

函数名	px_bool PX_ParticalLauncherCreate(PX_Partical_Launcher *launcher,px_memorypool *mp,PX_ParticalLauncher_InitializeInfo Info);
说明	创建一个粒子发射器
参数	launcher 发射器 mp 内存池 Info 发射器参数
返回值	-

渲染粒子发射器

函数名	px_void PX_ParticalRender(PX_Partical_Launcher *launcher,px_surface *surface,px_dword elapsed);
说明	渲染粒子系统
参数	launcher 发射器 surface 渲染表面 elapsed 上一次渲染经过的毫秒时间
返回值	

释放粒子发射器

函数名	px_void PX_ParticalFree(PX_Partical_Launcher *launcher);
说明	释放一个粒子发射器
参数	launcher 发射器
返回值	

设置粒子发射器的位置

函数名	px_void PX_ParticalLauncherSetLauncherPosition(PX_Partical_Launcher *launcher,px_point position);
说明	设置粒子系统的位置
参数	launcher 发射器

	direction 方向,这个方向是以局部坐标为准的
返回值	

设置粒子发射器喷射方向

函数名	px_void PX_ParticalLauncherSetDirection(PX_Partical_Launcher *launcher,px_point direction)
说明	设置粒子系统喷射方向
参数	launcher 发射器 direction 方向,这个方向是以局部坐标为准的
返回值	

MQTT 协议

MQTT for PainterEngine 是基于 MQTT3.1.1 标准的 PainterEngine 内置支持协议,支持 MQTT Client 的基础操作,MQTT Broker(Server)可由其他供应商或公有库架设

初始化协议

函数名	px_bool PX_MQTTInitialize(PX_MQTT *Mqtt,px_memorypool *mp,PX_Linker *linker);
说明	初始化 MQTT 协议
参数	mqtt 实例 mp 内存池 linker 通讯链接器
返回值	如果成功返回 PX_TRUE,否者返回 PX_FALSE

MQTT 连接

函数名	PX_MQTT_CONNECT PX_MQTTConnect(PX_MQTT *Mqtt,PX_MQTT_ConnectDesc connectDesc);
说明	使用 MQTT 协议进行连接

参数	mqtt 实例 conndesc 连接描述结构体 typedef struct { const px_char *userName; 用户名 const px_char *password; 密码 const px_char *willTopic; 遗嘱主题 const px_void *willContent; 遗嘱内容 px_uint willSize; 遗嘱内容长度 px_bool willRetain; 遗嘱保留标志 px_bool CleanSession; 清理 session PX_MQTT_QOS_LEVEL willQoS; 遗嘱 Qos 级别 px_word KeepAliveTime;//seconds 连接保留时间(秒) }PX_MQTT_ConnectDesc;
返回值	PX_MQTT_CONNECT 枚举类型 PX_MQTT_CONNECT_SUCCEEDED 成功连接 PX_MQTT_CONNECT_ERROR_UNKNOW 未知错误 PX_MQTT_CONNECT_ERROR_DISCONNECT 链接器错误/无法连通 PX_MQTT_CONNECT_ERROR_PROTOCOL_NO_SUPPORT 协议不支持 PX_MQTT_CONNECT_ERROR_SESSION_ILLEGAL session 数据不合法 PX_MQTT_CONNECT_ERROR_SERVER_CRASH 服务器错误 PX_MQTT_CONNECT_ERROR_USER_WRONG 登录账户错误 PX_MQTT_CONNECT_ERROR_ILLEGAL 数据不合法

发布数据

函数名	px_bool PX_MQTTPublish(PX_MQTT *Mqtt,PX_MQTT_PublishDesc publishDesc);
说明	使用 MQTT 发布一个数据
参数	mqtt 实例 publishDesc 发布描述 typedef struct { PX_MQTT_QOS_LEVEL qosLevel; qos 级别 px_bool retain; 保留标志 const px_char *Topic; 主题 const px_void *payload; payload px_int payloadSize; payload 大小 }PX_MQTT_PublishDesc;
返回值	如果成功返回 PX_TRUE, 否则返回 PX_FALSE

订阅数据

函数名	px_bool PX_MQTTSubscribe(PX_MQTT *Mqtt,PX_MQTT_SubscribeDesc subscribeDesc);
说明	使用 MQTT 订阅数据
参数	mqtt 实例 subscribeDesc 订阅描述 typedef struct { PX_MQTT_QOS_LEVEL qosLevel; 最大支持 Qos 级别 const px_char *Topic; 订阅主题 }PX_MQTT_SubscribeDesc;
返回值	如果成功返回 PX_TRUE,否者返回 PX_FALSE,注意,返回 PX_FALSE 可能出现了不可挽回错误,可能需要修正该错误再使用该协议

取消订阅数据

函数名	px_bool PX_MQTTUnsubscribe(PX_MQTT *Mqtt,const px_char *Topic);
说明	取消订阅 MQTT 主题
参数	mqtt 实例 Topic 主题名字
返回值	-

MQTT Ping

函数名	px_bool PX_MQTTPingReq(PX_MQTT *Mqtt);
说明	Ping
参数	Mqtt 实例
返回值	-

监听

函数名	px_bool PX_MQTTListen(PX_MQTT *Mqtt);
说明	监听 MQTT,直到收到了一个合法的 Publish 数据返回 PX_TRUE,否者可能因为超时返回 PX_FALSE 获取最后一次返回主题请访问 Mqtt->Topic 获取最后一次返回内容请访问 Mqtt->payload
参数	Mqtt 实例
返回值	-收到了一个合法的 Publish 数据返回 PX_TRUE,因为超时返回 PX_FALSE

释放

函数名	px_void PX_MQTTFree(PX_MQTT *Mqtt);
说明	释放 MQTT 实例
参数	Mqtt 实例
返回值	-

资源管理器

painterengine 提供资源管理器,所有的资源从其它存储介质加载后将被以特定的格式保存在资源管理器中,资源管理器所有的对象均由一个唯一的 key 进行标识存储,其内部使用 CRC+红黑树的形式进行索引.

资源包括以下几种类型:

```
typedef enum
{
    PX_RESOURCE_TYPE_NULL, //无类型
    PX_RESOURCE_TYPE_ANIMATIONLIBRARY, //动画集
    PX_RESOURCE_TYPE_SCRIPT, //编译脚本
    PX_RESOURCE_TYPE_TEXTURE, //纹理
    PX_RESOURCE_TYPE_SHAPE, //轮廓
    PX_RESOURCE_TYPE_DATA, //数据类型
}PX_RESOURCE_TYPE;
```

初始化

函数名	px_bool PX_ResourceLibraryInit(px_memorypool *mp,PX_ResourceLibrary *lib);
说明	资源库初始化
参数	mp 资源存放内存池 lib 资源库
返回值	如果成功返回 PX_TRUE,否者返回 PX_FALSE

加载一个资源

函数名	px_bool PX_ResourceLibraryLoad(px_memorypool *mptemp,PX_ResourceLibrary *lib,PX_RESOURCE_TYPE type,px_byte *data,px_uint datasize,px_char *key);
说明	加载资源
参数	mptemp 用于资源剖析的临时内存池 lib 资源库 type 资源类型 data 数据缓存 datasize 数据大小 key 索引 key
返回值	如果成功返回 PX_TRUE,否者返回 PX_FALSE

添加一个贴图资源

函数名	px_bool PX_ResourceLibraryAddTexture(PX_ResourceLibrary *lib,px_char *key,px_texture *pTexture);
说明	加载贴图资源
参数	lib 资源库 key 索引 key pTexture 需要加载的贴图,这个贴图数据将会被深拷贝到资源池中
返回值	如果成功返回 PX_TRUE,否者返回 PX_FALSE

读取资源

函数名	px_bool PX_ResourceLibraryGet(PX_ResourceLibrary *lib,PX_Resource *pRes,px_char *key);
说明	读取对应 key 的资源类型
参数	lib 资源库 pres 用于存储读取资源结构 key 对应字符索引
返回值	如果成功返回 PX_TRUE,否者返回 PX_FALSE

读取纹理资源

函数名	px_texture *PX_ResourceLibraryGetTexture(PX_ResourceLibrary *lib,px_char *key);
说明	从资源中读取一个纹理
参数	lib 资源库 key 对应字符索引
返回值	如果成功返回该资源指针,否者返回 PX_NULL

读取轮廓资源

函数名	px_shape *PX_ResourceLibraryGetShape(PX_ResourceLibrary *lib,px_char *key);
说明	从资源中读取一个轮廓
参数	lib 资源库 key 对应字符索引
返回值	如果成功返回该资源指针,否者返回 PX_NULL

读取动画资源

函数名	px_animationlibrary *PX_ResourceLibraryGetAnimationLibrary(PX_ResourceLibrary *lib,px_char
-----	--

	*key);
说明	从资源中读取一个动画集
参数	lib 资源库 key 对应字符索引
返回值	如果成功返回该资源指针,否者返回 PX_NULL

读取脚本资源

函数名	px_animationlibrary *PX_ResourceLibraryGetScript(PX_ResourceLibrary *lib,px_char *key);
说明	从资源中读取一个脚本资源
参数	lib 资源库 key 对应字符索引
返回值	如果成功返回该资源指针,否者返回 PX_NULL

读取声音资源

函数名	px_sounddata *PX_ResourceLibraryGetSound(PX_ResourceLibrary *lib,px_char *key);
说明	从资源中读取一个脚本资源
参数	lib 资源库 key 对应字符索引
返回值	如果成功返回该资源指针,否者返回 PX_NULL

读取数据资源

函数名	px_memory *PX_ResourceLibraryGetData(PX_ResourceLibrary *lib,px_char *key);
说明	从资源中读取一个脚本资源
参数	lib 资源库 key 对应字符索引
返回值	如果成功返回该资源指针,否者返回 PX_NULL

删除资源

函数名	px_void PX_ResourceLibraryGet(PX_ResourceLibrary *lib, px_char *key);
说明	删除对应 key 的资源类型
参数	lib 资源库 key 对应字符索引
返回值	-

释放资源库

函数名	px_bool PX_ResourceLibraryGet(PX_ResourceLibrary *lib,PX_Resource *pRes,px_char *key);
说明	读取对应 key 的资源类型
参数	lib 资源库 pres 用于存储读取资源结构 key 对应字符索引
返回值	如果成功返回 PX_TRUE,否者返回 PX_FALSE

帧同步协议

painterEngine 提供两种网络同步协议

一种是可用于多人游戏中帧同步或状态同步的开发,提供高容错,延迟优化,断线重连,数据平滑,第三方数据扰乱,数据攻击,等优化

在默认情况下,该同步协议的平滑延迟为 66 毫秒,约每秒 15 帧,在所有客户端连接后,服务端开始以这个频率对所有游戏客户端进行同步

一种是数据同步协议,该协议将保证每一个客户端的数据和服务端数据同步.

同步服务端

服务端初始化

函数名	px_bool PX_SyncFrameServerInit(PX_Sync_Server *sync,px_memorypool *mp,px_dword updateDuration,PX_Linker *linker);
说明	同步器服务端初始化
参数	sync 同步器结构体 mp 内存池 updateDuration 平滑延迟 Linker 通讯链接器,可以使用任何不可靠通信 IO 实现,但是数据丢失率过高可能会导致该同步算法有高卡顿现象,可以通过提高PX_SYNC_CLIENT_SEND_DURATION 和 PX_SYNC_SERVER_SEND_TIMES 减轻该丢包造成的延迟,但这也意味着将占有更大的带宽
返回值	如果成功返回 PX_TRUE,否者返回 PX_FALSE

服务端添加客户端

函数名	px_bool PX_SyncFrameServerAddClient(PX_Sync_Server *sync,px_dword verify_id,px_dword client_verify_id,px_dword c_id);
说明	服务端添加客户端
参数	sync 同步器结构体 verify_id 验证 id,这个 id 需要以安全随机数产生不可预测,与 client_verify_id 成对使用 client_verify_id 客户端验证 id,这个 id 需要以安全随机数产生不可预测,与 verify_id 成对使用,该表示是验证客户端的唯一标准方式 c_id 用于客户端的玩家标识,将会随着指令一通写入到指令流当中
返回值	如果成功返回 PX_TRUE,否者返回 PX_FALSE

服务端继续运行

函数名	px_void PX_SyncFrameServerRun(PX_Sync_Server *sync);
说明	服务端继续运行
参数	sync 同步器结构体

返回值	-

服务端暂停运行

函数名	px_void PX_SyncFrameServerStop(PX_Sync_Server *sync);
说明	服务端暂停运行
参数	sync 同步器结构体
返回值	-

服务端更新循环

函数名	px_void PX_SyncFrameServerUpdate(PX_Sync_Server *sync,px_dword elapsed);
说明	服务端更新
参数	sync 同步器结构体 elapsed 距离上次更新经过的时间,注意,该值应该做到尽可能的小,否则可能导致过大的抖动
返回值	-

释放服务端

函数名	px_void PX_SyncFrameServerFree(PX_Sync_Server *sync);
说明	释放服务端资源
参数	sync 同步器结构体
返回值	-

取得数据累加和校验

函数名	px_uint32 PX_SyncFrameServerSum32(PX_Sync_Server *sync);
说明	计算当前同步数据的累加和,注意,只有当一帧结束后进行累加校验才有参考意义

参数	sync 同步器结构体
返回值	-

帧客户端

客户端初始化

函数名	px_bool PX_SyncFrameClientInit(PX_Sync_Client *client,px_memorypool *mp,px_dword updateDuration,PX_Sync_Port serverport,px_dword server_verify_id,px_dword verify_id, PX_Linker *linker);
说明	同步器客户端初始化
参数	client 同步器结构体 mp 内存池 verify_id 验证 id,这个 id 需要以安全随机数产生不可预测,与 server_verify_id 成对使用 erver_verify_id 客户端验证 id,这个 id 需要以安全随机数产生不可预测,与 verify_id 成对使用,该表示是验证客户端的唯一标准方式 Linker 通讯链接器,可以使用任何不可靠通信 IO 实现,但是数据丢失率过高可能会导致该同步算法有高卡顿现象,可以通过提高 PX_SYNC_CLIENT_SEND_DURATION 和 PX_SYNC_SERVER_SEND_TIMES 减轻该丢包造成的延迟,但这也意味着将占有更大的带宽
返回值	如果成功返回 PX_TRUE,否者返回 PX_FALSE

添加指令

函数名	px_void PX_SyncFrameClientAddInstr(PX_Sync_Client *client,px_void *instr,px_int size);
说明	添加客户端指令流,该指令流会被所有客户端同步,同步时间由服务端确定
参数	client 同步器结构体 instr 指令流指针 size 指令流长度
返回值	-

更新客户端

函数名	px_void PX_SyncFrameClientUpdate(PX_Sync_Client *sync,px_dword elapsed);
说明	更新客户端
参数	client 同步器结构体 elapsed 距离上次更新经过的时间,注意,该值应该做到尽可能的小,否则可能帧率不稳定
返回值	-

释放客户端

函数名	px_void PX_SyncFrameClientFree(PX_Sync_Client *sync);
说明	释放客户端资源
参数	sync 同步器结构体
返回值	-

取得数据累加和校验

函数名	px_uint32 PX_SyncFrameClientSum32(PX_Sync_Client *sync);
说明	计算当前同步数据的累加和,注意,只有当一帧结束后进行累加校验才有参考意义
参数	sync 同步器结构体
返回值	-

JSON

PainterEngine 支持对 Json 格式的解析,但不支持字符串转义,如果需要转义请自行处理

```
typedef struct _PX_Json_Object
{
    px_list values;//用于存储 PX_Json_Value 的链表
}PX_Json_Object;
```

```
typedef struct
```

```

{
    PX_JSON_VALUE_TYPE type;//值数据类型
    px_string name;//值名
    union
    {
        px_string _string;//字符串数据
        px_double _number;//数值数据
        px_bool _boolean;//布尔类型数据
        PX_Json_Object json_Object;//对象数据
        px_list_Array;//数组数据
    };
}PX_Json_Value;

```

初始化

函数名	px_bool PX_JsonInitialize(px_memorypool *mp,PX_Json *pjson);
说明	初始化一个JSON 结构体
参数	Mp 内存池 Pjson json 结构体
返回值	-如果成功返回 PX_TRUE,否者返回 PX_FALSE

读取 JSON Object 的子数据

函数名	PX_Json_Value *PX_JsonGetObjectValue(PX_Json_Value *json_value,const px_char name[]);
说明	读取 JSON Object 的子数据
参数	json_value Object 类型的 JSon 数据 name Json 值名
返回值	对象类型指针

读取 JSON 数据

函数名	PX_Json_Value * PX_JsonGetValue(PX_Json *json,const px_char _payload[])
说明	读取 JSON 数据
参数	Pjson json 结构体 payload 加载文本

	payload 示范 读取类成员 Human.lily.age 支持数组读取 human.score[10].point
返回值	-对象类型指针

读取 JSON Array 数据

函数名	PX_Json_Value * PX_JsonGetArrayValue(PX_Json_Value *value,px_int i);
说明	读取 JSONArray 数据
参数	Value array 类型 json 值 I array 访问索引
返回值	-对象类型指针

解析 JSON 数据

函数名	px_bool PX_JsonParse(PX_Json *pjson,const px_char *json_content);
说明	解析 JSON 数据
参数	Pjson json 结构体 Json_content JSON 文本
返回值	-如果成功返回 PX_TRUE,否者返回 PX_FALSE

建立 JSON 数据类型

函数名	px_bool PX_JsonBuild_Value(PX_Json_Value *pValue,px_string *_out,px_bool bArrayValue);
说明	建立一个 JSON 数据类型，建立类型插入到 JSon 结构中以建立 JSON 结构 (这个函数为内部调用)
参数	pValue JSON 数据 _out 名称 bArrayValue 数组类型
返回值	-如果成功返回 PX_TRUE,否者返回 PX_FALSE

将 JSON 结构转换为文本

函数名	px_bool PX_JsonBuild(PX_Json *pjson,px_string *_out);
说明	解析 JSON 数据
参数	Pjson json 结构体 Json_content JSON 文本
返回值	-如果成功返回 PX_TRUE,否者返回 PX_FALSE

删除 JSON 结构中一个数据

函数名	px_void PX_JsonDelete(PX_Json *pjson,const px_char payload[]);
说明	删除 JSon 结构中数据
参数	Pjson json 结构体 Payload payload ， 例如 A.B.C 或 A.B[1]
返回值	-

释放 JSON

函数名	px_void PX_JsonFree(PX_Json *pjson);
说明	释放 JSON 结构数据
参数	Pjson json 结构体
返回值	

JSON 数据建立

建立 Object 数据类型

函数名	px_bool PX_JsonCreateObjectValue(px_memorypool *mp,PX_Json_Value *pValue,const px_char name[]);
说明	建立 Object 数据类型
参数	Mp 内存池

	pValue 需要建立数据类型的父对象 name 数据名
返回值	如果成功返回 PX_TRUE,否者返回 PX_FALSE

建立 String 数据类型

函数名	px_bool PX_JsonCreateStringValue(px_memorypool *mp,PX_Json_Value *pValue,const px_char name[],const px_char text[]);
说明	建立 Object 数据类型
参数	Mp 内存池 pValue 需要建立数据类型的父对象 name 数据名 text 数据的 string
返回值	如果成功返回 PX_TRUE,否者返回 PX_FALSE

建立 Number 数据类型

函数名	px_bool PX_JsonCreateNumberValue(px_memorypool *mp,PX_Json_Value *pValue,const px_char name[],px_double value);
说明	建立 Number 数据类型
参数	Mp 内存池 pValue 需要建立数据类型的父对象 name 数据名 value 数据的值
返回值	如果成功返回 PX_TRUE,否者返回 PX_FALSE

建立 Boolean 数据类型

函数名	px_bool PX_JsonCreateBooleanValue(px_memorypool *mp,PX_Json_Value *pValue,const px_char name[],px_bool b);
说明	建立 Boolean 数据类型
参数	Mp 内存池 pValue 需要建立数据类型的父对象 name 数据名 Boolean 布尔值

返回值	如果成功返回 PX_TRUE,否者返回 PX_FALSE
-----	------------------------------

建立 Array 数据类型

函数名	px_bool PX_JsonCreateArrayValue(px_memorypool *mp,PX_Json_Value *pValue,const px_char name[]);
说明	建立 Array 数据类型
参数	Mp 内存池 pValue 需要建立数据类型的父对象 name 数据名
返回值	如果成功返回 PX_TRUE,否者返回 PX_FALSE

插入 String 数据类型到 JSON 结构中

函数名	px_bool PX_JsonAddString(PX_Json *pjson,const px_char parent_payload[],const px_char name[],const px_char text[]);
说明	插入 String 数据类型到 JSON 结构中
参数	pJson 目标数据 payload 数据访问代码,例如访问对象 A 中的 B 则为 A.B,如 B 为数组则为 A.B[数组索引] name 数据名 text 文本
返回值	如果成功返回 PX_TRUE,否者返回 PX_FALSE

插入 Number 数据类型到 JSON 结构中

函数名	px_bool PX_JsonAddNumber(PX_Json *pjson,const px_char parent_payload[],const px_char name[],px_double number);
说明	插入 Number 数据类型到 JSON 结构中
参数	pJson 目标数据 payload 数据访问代码,例如访问对象 A 中的 B 则为 A.B,如 B 为数组则为 A.B[数组索引] name 数据名 number 插入的 number 数据
返回值	如果成功返回 PX_TRUE,否者返回 PX_FALSE

插入 Boolean 数据类型到 JSON 结构中

函数名	px_bool PX_JsonAddBoolean(PX_Json *pjson,const px_char parent_payload[],const px_char name[],const px_bool b);
说明	插入 Boolean 数据类型到 JSON 结构中
参数	pJson 目标数据 payload 数据访问代码,例如访问对象 A 中的 B 则为 A.B,如 B 为数组则为 A.B[数组索引] name 数据名 b Boolean 数据
返回值	如果成功返回 PX_TRUE,否者返回 PX_FALSE

插入 Array 数据类型到 JSON 结构中

函数名	px_bool PX_JsonAddArray(PX_Json *pjson,const px_char parent_payload[],const px_char name[]);
说明	插入一个空的 Array 数据类型到 JSON 结构中
参数	pJson 目标数据 payload 数据访问代码,例如访问对象 A 中的 B 则为 A.B,如 B 为数组则为 A.B[数组索引] name 数据名
返回值	如果成功返回 PX_TRUE,否者返回 PX_FALSE

插入 Object 数据类型到 JSON 结构中

函数名	px_bool PX_JsonAddObject(PX_Json *pjson,const px_char parent_payload[],const px_char name[]);
说明	插入一个空的 Object 数据类型到 JSON 结构中
参数	pJson 目标数据 payload 数据访问代码,例如访问对象 A 中的 B 则为 A.B,如 B 为数组则为 A.B[数组索引] name 数据名
返回值	如果成功返回 PX_TRUE,否者返回 PX_FALSE

修改 String 数据类型的数据

函数名	px_bool PX_JsonSetString(PX_Json *pjson,const px_char payload[],const px_char text[]);
说明	修改一个 string 数据类型为目标值
参数	Mp 内存池 payload 数据访问代码,例如访问对象 A 中的 B 则为 A.B,如 B 为数组则为 A.B[数组索引] text 需要修改成的文本类型
返回值	如果成功返回 PX_TRUE,否则返回 PX_FALSE

修改 Number 数据类型的数据

函数名	px_bool PX_JsonSetNumber(PX_Json *pjson,const px_char payload[],const px_double number);
说明	修改一个 Number 数据类型为目标值
参数	Mp 内存池 payload 数据访问代码 text 需要修改成的文本类型
返回值	如果成功返回 PX_TRUE,否则返回 PX_FALSE

修改 Boolean 数据类型的数据

函数名	px_bool PX_JsonSetBoolean(PX_Json *pjson,const px_char payload[],const px_bool b);
说明	修改一个 Boolean 数据类型为目标值
参数	Mp 内存池 payload 数据访问代码 b 需要修改成的文本类型
返回值	如果成功返回 PX_TRUE,否则返回 PX_FALSE

向一个 Object 数据类型中插入数据

函数名	px_bool PX_JsonObjectAddValue(PX_Json_Value *pObject,PX_Json_Value *value);
说明	向一个 Object 数据类型中插入数据
参数	pObject 目标 Object value 需要插入的数据
返回值	如果成功返回 PX_TRUE,否者返回 PX_FALSE

向一个 Array 数据类型中插入数据

函数名	px_bool PX_JsonArrayAddValue(PX_Json_Value *pArray,PX_Json_Value *value);
说明	向一个 Array 数据类型中插入数据
参数	pArray 目标 Array value 需要插入的数据
返回值	如果成功返回 PX_TRUE,否者返回 PX_FALSE

PainterEngine 基本对象

使用 PainterEngine 绘制对象

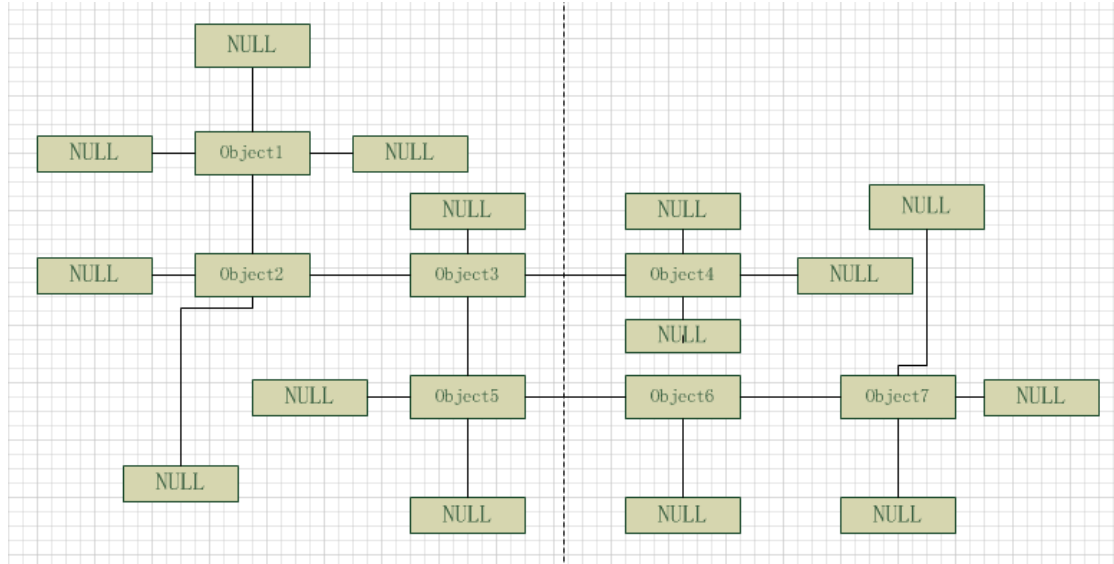
PainterEngine 的对象是由 PainterEngine 图形渲染函数进一步封装的结果，在 PainterEngine 中，所有的显示物都是以对象的形式存在的，对象是 PainterEngine 显示的基本形式，通过对对象的派生而衍生出不同的功能类型。

PainterEngine 对象机制

PainterEngine 的对象是所有控件的基础类型，一个对象当中包含了以下几种类型

1. 对象的父对象
2. 对象的前一个兄弟对象
3. 对象的后一个兄弟对象
4. 对象的子对象

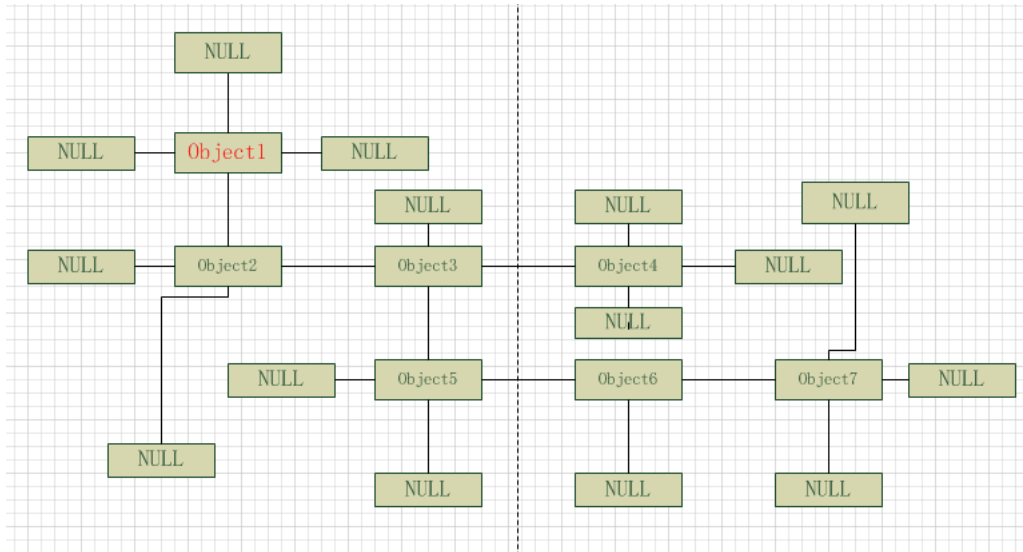
用下图表示对象之间的级联关系



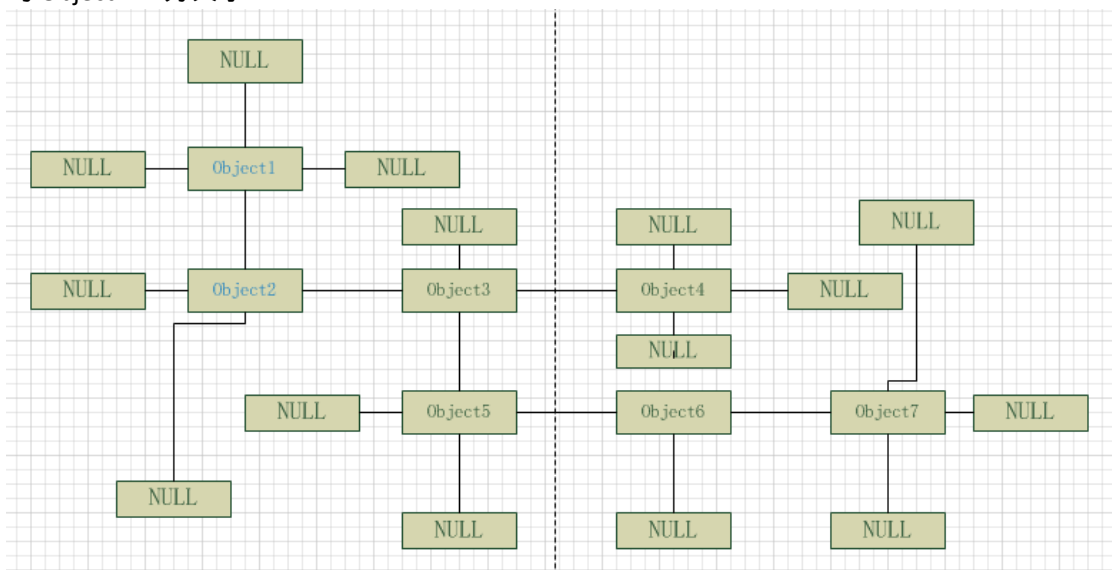
依照图所示，每一个 Object 都拥有 4 个链接点，向上方向指向该对象的父对象，向下指向该对象的子对象，向左指向该对象的前一个兄弟对象，向右指向改对象的后一个兄弟对象，当方向上不存在这个对象指向的对应节点的时候，这个节点将会也必须被赋值为 NULL 表示空。也就是，一个对象的四个方向节点，要么有效存在，要么为 NULL。

当父对象为 NULL 且只存在子对象的时候，我们称之为对“根对象”，

根对象不存在父对象，也不存在兄弟对象，例如下图当中，被红色标志的 Object 即为根对象



对象的级联关系中，我们将上下级联的对象称之为“父子对象”，如下图所示，Object1 与 Object2 互为父子



Object1 是 Object2 的父对象，Object2 是 Object1 的子对象

同时可以从上图看到 Object2 拥有 2 个兄弟对象，分别是 Object3 与 Object4，在广义上 Object3 与 Object4 也是 Object1 的子对象，但是，Object3 与 Object4 的父向量并不指向 Object1 而指向了 NULL，这也意味着无法通过 Object3 或 Object4 直接寻找到其父对象而必须经过 Object2。这也就意味着 Object2 是对象的入口，在同一层级的对象中，有且仅有一个入口对象，在上图中可以看到，Object2 是 Object1 子对象的入口对象，Object5 是 Object3 子对象的入口对象

PainterEngine 提供了以下几个函数，用于对象的操作与使用

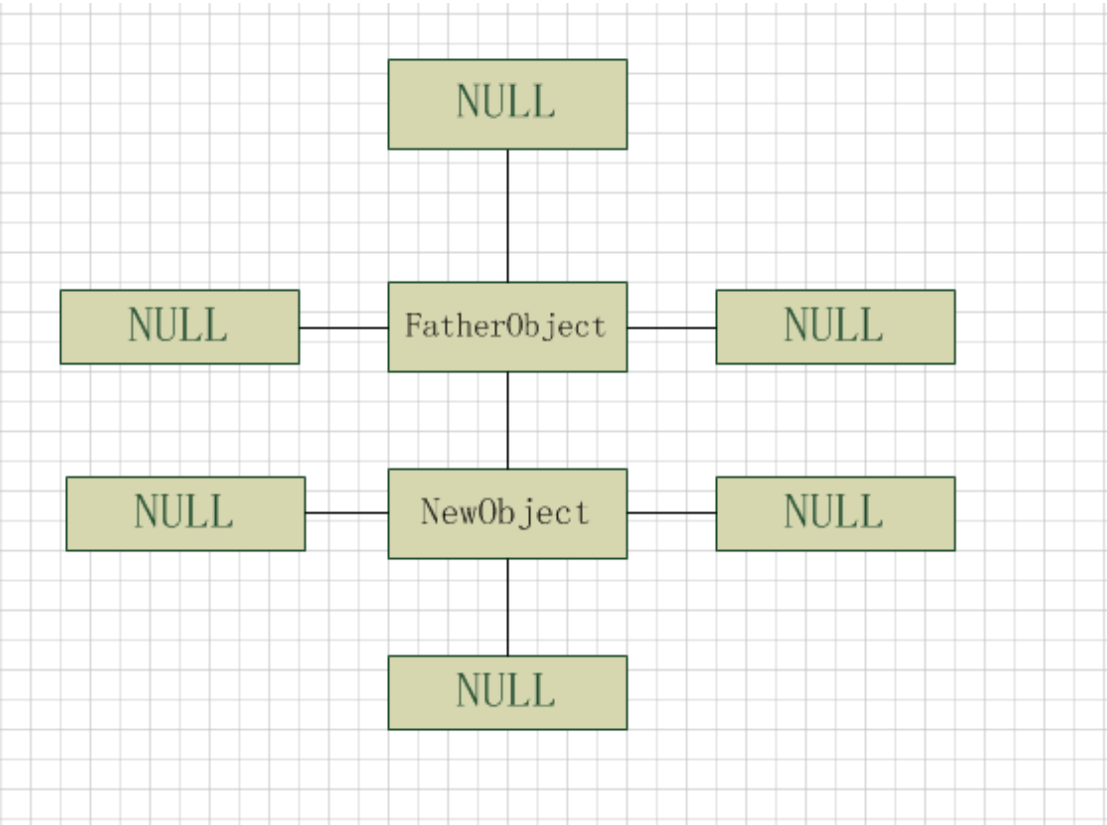
```
PX_Object *PX_ObjectCreate(PX_Object *Parent,int x,int y,int Width,int Height);
```

函数名	PX_ObjectCreate
功能	创建一个对象
参数 Parent	创建对象的父对象
参数 x	对象的左上角坐标 x
参数 y	对象的左上角坐标 y
参数 width	对象的宽度
参数 height	对象的高度
返回值 PX_ObjectCreate *	创建对象的指针

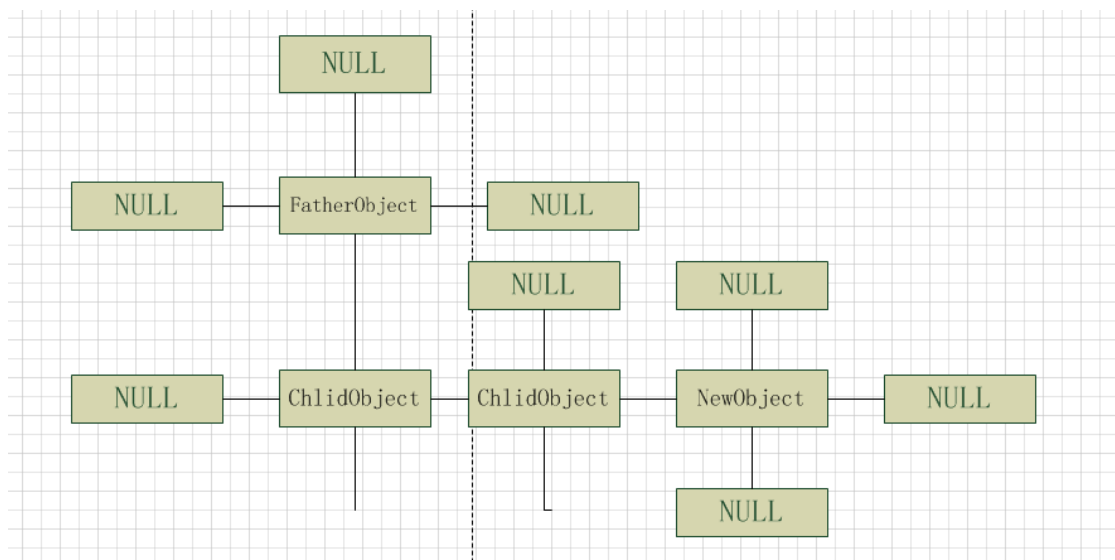
当 Parent 为 NULL 的时候，表示这个对象是一个根对象，当对象不为 NULL 的时候，这个对象创建如下

创建对象的连接规则

- 1.当父对象没有子对象的时候，创建的对象将作为其子对象



- 2.当父对象拥有子对象的时候，这个对象将被插入到父对象的子对象层次的最后一个节点



void PX_ObjectInit(PX_Object *Object,PX_Object *Parent,int x,int y,int Width,int Height);

函数名	PX_ObjectInit
功能	初始换一个对象
参数 Object	需要初始化的对象
参数 Parent	初始化该对象的父对象
参数 x	对象的左上角坐标 x
参数 y	对象的左上角坐标 y
参数 width	对象的宽度
参数 height	对象的高度

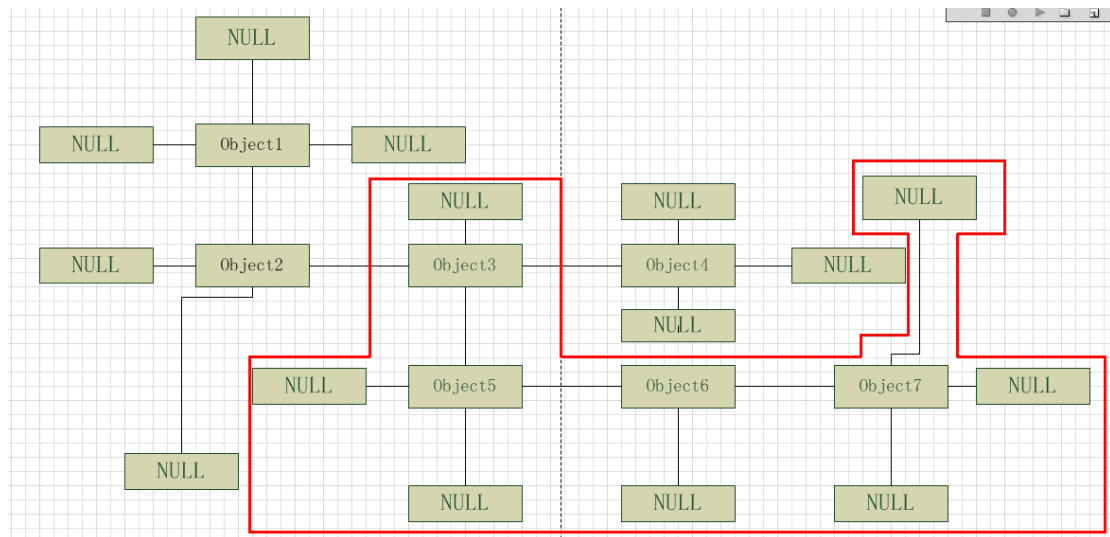
这个函数用于初始化自定义的一个对象，并将对象连接到父对象当中，连接的规则和创建对象的连接规则一样。

void PX_ObjectDelete(PX_Object *pObject);

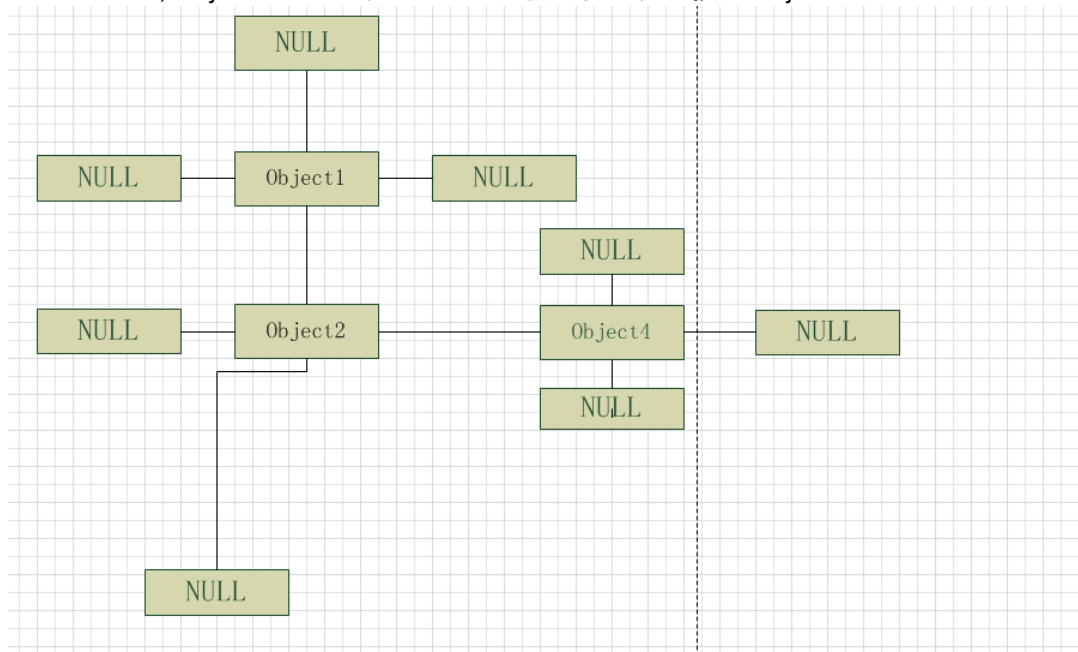
函数名	PX_ObjectDelete
功能	删除一个对象
参数 Object	指向该对象的指针

使用该函数将会将这个对象断开原有的对象链并删除，同时，这个对象所有的子对象都会被删除

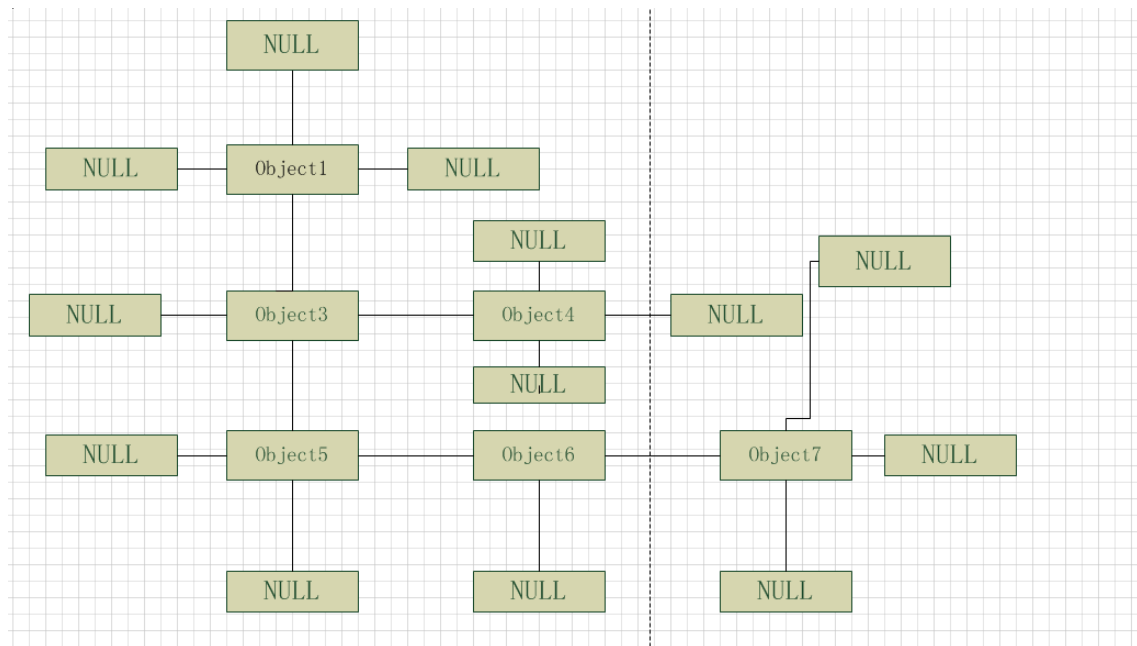
如下图所示，删除 Object 将导致如下对象都被删除



删除完成后，Object2 的下一个兄弟向量将被重新定位连接到 Object4



如果删除的是 Object2，则 Object1 的子对象将被重新定位到 Object3，而 Object3 将作为 Object1 子对象的入口对象



void PX_DeleteLinkerObject(PX_Object **ppObject);

函数名	PX_ObjectLinkerDelete
功能	删除该对象和该对象后续的所有兄弟对象
参数 Object	指向该对象的二级指针

void PX_ObjectSetPosition(PX_Object *Object,int x,int y);

函数名	PX_ObjectSetPosition
功能	设置对象的位置
参数 Object	该对象指针
参数 x	对象的新横坐标 x
参数 y	对象的新纵坐标 y

void PX_ObjectSetSize(PX_Object *Object,int Width,int Height);

函数名	PX_ObjectSetSize
功能	设置对象的大小
参数 Object	该对象的指针
参数 width	设置对象的新宽度
参数 height	设置对象的新高度

void PX_ObjectFree(PX_Object *Object);

函数名	PX_ObjectSetVisible
功能	设置该对象可见性 (将影响其子对象)
参数 Object	该对象指针
参数 Visible	可见性, 不为 0 表示可见, 为 0 不可见

void PX_ObjectSetFocus(PX_Object *Object);

函数名	PX_ObjectSetFocus
功能	设置该对象为焦点对象
参数 Object	该对象指针
备注	焦点对象会独占该对象树结构中的消息处理事件直到该对象的焦点属性被清除,通过设置 LostFocusReleaseEvent,可销毁失去焦点事件,注意,清除焦点对象将会一并清除该对象所有父级对象的焦点属性

void PX_ObjectClearFocus(PX_Object *Object);

函数名	PX_ObjectClearFocus
功能	取消对象的焦点属性
参数 Object	该对象指针
备注	焦点对象会独占该对象树结构中的消息处理事件直到该对象的焦点属性被清除,通过设置 LostFocusReleaseEvent,可销毁失去焦点事件,注意,如果该对象的子对象中仍然有焦点对象,那么该对象的焦点标志不会被清除

创建对象

非根对象是除根对象的附属对象，就如同对象机制中描述的那样，非根对象必须被链接到一个根对象当中，同时，可以进一步指派该对象的长宽高坐标位置等具体属性类型。

PX_Object *PX_ObjectCreate(px_memorypool *mp,PX_Object *Parent,px_float x,px_float y,px_float z,px_float Width,px_float Height,px_float Lenght);

函数名	PX_ObjectCreate
功能	创建一个对象
mp	对象使用的内存池
parent	父对象(如果为 NULL 表示这是一个根对象)
x,y,z	坐标信息,注意 z 坐标信息,z 的值越大,这个对象在渲染时将被最先绘制,这也就意味着 z 值小的对象将会覆盖在 z 值较大的对象中
Width,Heigth,Lenght	宽度,高度,长度
返回值	返回创建的对象指针

注意,其它的非根对象需要被链接到一个根对象中,如果该对象是一个实际绘制对象,可以在创建完根对象后对其的具体属性进行进一步的赋值操作。

设置对象的 usercode

```
px_void PX_ObjectSetUserCode(PX_Object *pObject,px_int user);  
px_void PX_ObjectSetUserPointer(PX_Object *pObject,px_void *user_ptr);
```

函数名	PX_ObjectSetUserCode PX_ObjectSetUserPointer
功能	设置对象的 user 字段,这个字段为用户自定义的类型,可以将这个值赋值为需要的数据类型,在对象更新渲染或者是事件回调时,利用该函数获取额外信息
Object	需要设置的对象指针
user_int user_ptr	设置的值,注意,每个对象的 user_int 和 User_ptr 为共享内存的数据类型.建议仅能设置其中一个

删除对象

```
px_void PX_ObjectDelete(PX_Object *pObject);
```

函数名	PX_ObjectDelete
功能	删除一个对象,注意,当需要删除对象的时候不能够使用 MP_Free 函数,必须调用该函数来删除一个对象,该函数将会断开对象与父对象与兄弟对象的链接,整理对象关系
Object	需要删除的对象指针

```
px_void PX_ObjectDeleteChilds( PX_Object *pObject );
```

函数名	PX_ObjectDeleteChilds
功能	删除一个对象的所有子对象
Object	需要删除子对象的对象指针

对象属性设置

```
px_void PX_ObjectSetPosition(PX_Object *Object,px_float x,px_float y);
```

函数名	PX_ObjectSetPosition
功能	设置一个对象的坐标
Object	需要设置的对象指针
x,y,z	三维坐标

px_void PX_ObjectSetSize(PX_Object *Object,px_float Width,px_float Height,px_float Length);

函数名	PX_ObjectSetSize
功能	设置一个对象的宽度和高度
Object	需要设置的对象指针
Width,Height,Length	宽,高,长

px_void PX_ObjectSetVisible(PX_Object *Object,px_bool visible);

函数名	PX_ObjectSetVisible
功能	设置一个对象的可见性
Object	需要设置的对象指针
visible	可见性,如果为 FALSE 这个对象将不被渲染

px_float PX_ObjectGetHeight(PX_Object *Object);

px_float PX_ObjectGetWidth(PX_Object *Object);

函数名	PX_ObjectGetHeight PX_ObjectGetWidth
功能	取得对象的高宽
Object	对象指针
返回值	对象的高宽

PainterEngine 更新/绘制流水线

PainterEngine 的绘制流程按照

先父对象，后子对象，先左兄弟对象，后右兄弟对象的流程进行绘制

PainterEngine 提供两个对象绘制函数进行对象的绘制，分别是

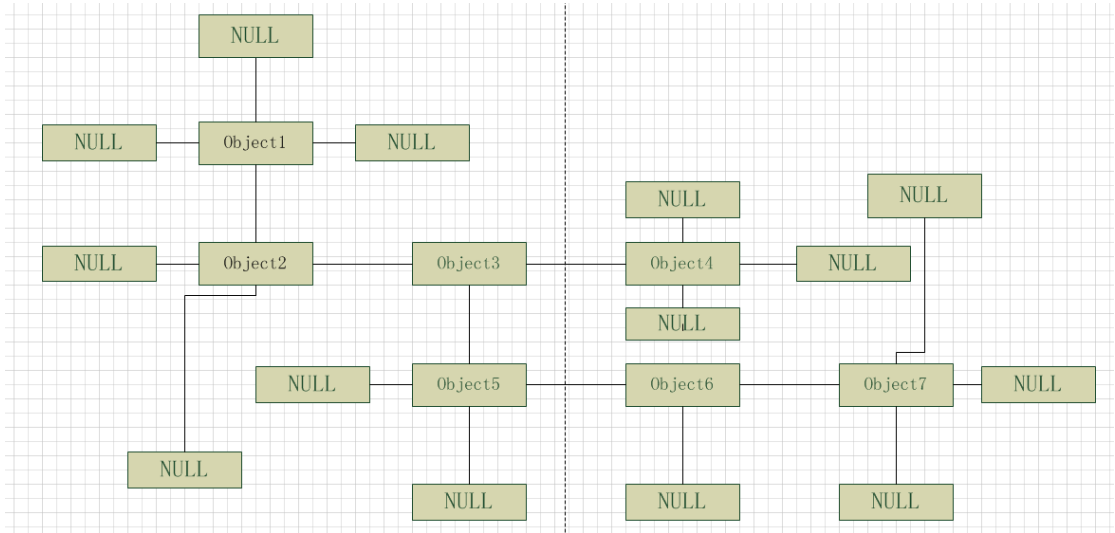
void PX_ObjectUpdate(struct _PX_Object *,px_uint elapsed);

函数名	ObjectUpdate
功能	更新该对象及该对象的所有子对象
参数 Object	需要更新的对象指针
参数 elapsed	上一次更新经过的时间

```
void PX_ObjectRender(px_surface *psurface,struct _PX_Object *,px_uint elapsed);
```

函数名	PX_ObjectRender
功能	绘制该对象及该对象的所有子对象
参数 psurface	将对象渲染到的目标表面
参数 Object	需要绘制的对象指针
参数 elapsed	上一次更新经过的时间

依照下图所示更新/绘制的绘制流程如下



Start→Object1→Object2→Object3→object5→object6→Object7→
Object4→End;

对象事件机制

事件类型

在 PainterEngine 中,每个对象都可以为其注册触发事件,当使用 PX_ObjectPostEvent 函数对一个对象进行事件投递后,该对象及其子对象都会收到该事件信息,如果这个对象有注册这个事件的处理函数,将会回调该处理函数.

注意,如果对象树结构中有一个焦点对象,那么该焦点对象将会独占所有的消息事件处理的优先权并决定事件是销毁还是继续投递

事件定义	事件响应
------	------

PX_OBJECT_EVENT_ANY	任意事件
PX_OBJECT_EVENT_CUSORMOVE	收到指针移动事件,例如鼠标移动触发
PX_OBJECT_EVENT_CURSORUP	收到指针抬起事件, 例如鼠标左键点击抬起, 或者触摸屏触摸抬起
PX_OBJECT_EVENT_CUSORRDOWN	收到指针按下事件, 例如鼠标右键点击按下
PX_OBJECT_EVENT_CUSORDOWN	收到指针按下事件, 例如鼠标左键点击按下, 或者触摸按下
PX_OBJECT_EVENT_CUSORRUP	收到指针右键抬起事件, 例如鼠标右键抬起
PX_OBJECT_EVENT_CURSOROVER	指针移进时触发一次
PX_OBJECT_EVENT_CURSOROUT	指针移出时触发一次
PX_OBJECT_EVENT_CURSORWHEEL	指针滚动,例如鼠标中键滚动或者触摸屏滑动
PX_OBJECT_EVENT_CURSORCLICK	指针单机,例如鼠标单击或触摸屏单击, 这需要按下抬起都在同一个位置
PX_OBJECT_EVENT_CUSORDRAG	指针拖拽,例如鼠标按下拖动或者触摸屏滑动
PX_OBJECT_EVENT_STRING	输入法输入了一个字符串
PX_OBJECT_EVENT_EXECUTE	对象被触发,例如按钮被点击触发(按钮触发有别于 Click,click 指按下抬起都在同一个位置)
PX_OBJECT_EVENT_VALUECHANGED	对象值改变,例如 sliderbar 或者 list 选择改变
PX_OBJECT_EVENT_DRAGFILE	外部拖入了一个文件进来
PX_OBJECT_EVENT_KEYDOWN	键盘某个按键被按下
PX_OBJECT_EVENT_IMPACT	两个对象两两碰撞(这个事件尽在 PainterEngine World 中生效)
PX_OBJECT_EVENT_ONFOCUSCHANGED	焦点改变
PX_OBJECT_EVENT_SCALE	缩放事件
PX_OBJECT_EVENT_WINDOWRESIZE	窗口的大小发生了变化
PX_OBJECT_EVENT_ONFOCUS	取得焦点
PX_OBJECT_EVENT_LOSTFOCUS	失去焦点
PX_OBJECT_EVENT_CANCEL	对象被取消(例如 explorer 控件点击取消)
PX_OBJECT_EVENT_CLOSE	对象被关闭(例如 widget 被关闭)

响应处理函数

响应处理函数是对象接收到对应事件调用的回调函数
其定义为

px_void (*ProcessFunc)(PX_Object *,PX_Object_Event e,px_void *ptr);

函数名	ProcessFunc
功能	事件回调函数
参数 Object	响应对象指针

参数 Event	响应的事件类型
参数 ptr	用户传入的指针

注册对象事件

px_int PX_ObjectRegisterEvent(PX_Object *Object,px_uint Event,px_void (*ProcessFunc)(PX_Object *,PX_Object_Event e),px_void *ptr);

函数名	PX_ObjectRegisterEvent
功能	为一个对象注册响应事件
参数 Object	需要绘制的对象指针
参数 Event	响应的事件类型
ProcessFunc	响应处理函数
ptr	用户指针

派分事件

px_void PX_ObjectPostEvent(PX_Object *pPost,PX_Object_Event Event);

函数名	ObjectPostEvent
功能	派分事件,该事件将会被派分到该对象及其子对象当中,如果该对象注册了该事件,将会执行该事件的回调函数,如果派分的对象树中有焦点对象,焦点会独占所有的事件处理
参数 Object	派分的对象指针
参数 Event	响应的事件

构造事件

函数原型	PX_Object_Event PX_OBJECT_BUILD_EVENT(px_uint Event);
功能	构造一个事件
参数 Event	事件类型
返回值	构造的事件

构造字符串事件

函数原型	PX_Object_Event PX_OBJECT_BUILD_EVENT_STRING(px_uint Event,const px_char *content);
功能	构造一个字符串事件
参数 Event	事件类型
参数 content	需要传入的字符串
返回值	构造的事件

指针事件偏移

函数原型	PX_Object_Event PX_Object_Event_CursorOffset(PX_Object_Event e,px_point offset);
功能	将一个指针类型的坐标偏移一个位置
参数 e	传入事件
参数 offset	偏移量
返回值	构造的事件

取得指针事件的 X 坐标

函数原型	px_float PX_Object_Event_GetCursorX(PX_Object_Event e);
功能	取得指针事件的 X 坐标
参数 e	传入事件
返回值	X 坐标

取得指针事件的 Y 坐标

函数原型	px_float PX_Object_Event_GetCursorY(PX_Object_Event e);
------	---

功能	取得指针事件的 Y 坐标
参数 e	传入事件
返回值	Y 坐标

取得指针事件的 Z 坐标

函数原型	px_float PX_Object_Event_GetCursorZ(PX_Object_Event e);
功能	取得指针事件的 Z 坐标
参数 e	传入事件
返回值	Z 坐标

设置指针事件的 X 坐标

函数原型	px_void PX_Object_Event_SetCursorX(PX_Object_Event *e,px_float x);
功能	设置指针事件的 X 坐标
参数 e	需要修改的传入事件
参数 x	需要设置的值
返回值	

设置指针事件的 Y 坐标

函数原型	px_void PX_Object_Event_SetCursorY(PX_Object_Event *e,px_float y);
功能	设置指针事件的 Y 坐标
参数 e	需要修改的传入事件
参数 y	需要设置的值
返回值	

设置指针事件的 X 坐标

函数原型	px_void PX_Object_Event_SetCursorX(PX_Object_Event *e,px_float z);
功能	设置指针事件的 z 坐标
参数 e	需要修改的传入事件
参数 z	需要设置的值
返回值	

取得键盘事件的输入码

函数原型	px_uint PX_Object_Event_GetKeyDown(PX_Object_Event e);
功能	取得键盘事件的输入码
参数 e	传入事件
返回值	键盘输入码

取得输入法事件的输入字符串

函数原型	px_char* PX_Object_Event_GetStringPtr(PX_Object_Event e);
功能	取得输入法事件的输入字符串
参数 e	传入事件
返回值	输入字符串指针

PainterEngine UI 对象

PainterEngine UI 是基于 PainterEngine Object 开发的默认 UI 控件,包含按钮,文本,图片,进度条,拖动框,菜单,下拉框.....等一系列平台无关的交互控件

静态文本框

创建文本框

函数名	PX_Object * PX_Object_LabelCreate(px_memorypool *mp,PX_Object *Parent,px_int x,px_int y,px_int Width,px_int Height,const px_char *Text,PX_FontModule *fm,px_color Color);
功能	创建一个静态文本框
mp	使用内存池
Parent	父对象
x,y	二维坐标 xy
width,height	宽度,高度
text	文本
fm	字模,如果为 PX_NULL 表示默认字模
Color	文本颜色
返回值	如果创建成功返回对象指针,否者返回 PX_NULL

取得对象数据

函数名	PX_Object_Label * PX_Object_GetLabel(PX_Object *Object);
功能	取得静态文本框对象数据
Object	对象指针
返回值	如果成功返回对象数据,否者返回 PX_NULL

取得静态文本框文本

函数名	px_char * PX_Object_LabelGetText(PX_Object *Label);
功能	取得静态文本框的文本

Object	对象指针
返回值	如果成功返回文本框文本,否则返回PX_NULL

设置静态文本框文本

函数名	px_void PX_Object_LabelSetText(PX_Object *pLabel,px_char *Text);
功能	设置静态文本框的文本
Object	对象指针
Text	文本指针
返回值	-

设置文本框背景颜色

函数名	px_void PX_Object_LabelSetBackgroundColor(PX_Object *pLabel,px_color Color);
功能	设置文本框的背景颜色
Object	对象指针
color	颜色
返回值	-

设置文本框文本对齐方式

函数名	px_void PX_Object_LabelSetAlign(PX_Object *pLabel,PX_FONT_ALIGN Align);
功能	设置文本框的文本对齐方式
Object	对象指针
color	颜色
返回值	-

进度条

创建进度条

函数名	PX_Object * PX_Object_ProcessBarCreate(px_memorypool *mp,PX_Object *Parent,px_int x,px_int y,px_int Width,px_int Height);
功能	创建进度条
mp	内存池
Parent	父对象
x,y	平面坐标
width,height	宽度,高度
	如果创建成功返回对象指针,否者返回 PX_NULL

取得对象数据

函数名	PX_Object_ProcessBar *PX_Object_GetProcessBar(PX_Object *Object);
功能	取得进度条对象数据
Object	对象指针
返回值	如果成功返回对象数据指针,否者返回 PX_NULL

设置进度条颜色

函数名	px_void PX_Object_ProcessBarSetColor(PX_Object *pProcessBar,px_color Color);
功能	设置进度条颜色
Object	对象指针
color	颜色

返回值	-
-----	---

设置进度条当前值

函数名	px_void PX_Object_ProcessBarSetValue(PX_Object *pProcessBar,px_int Value);
功能	设置进度条最大值
Object	对象指针
value	当前值
返回值	-

设置进度条最大值

函数名	px_void PX_Object_ProcessBarSetMax(PX_Object *pProcessBar,px_int Max);
功能	设置进度条最大值
Object	对象指针
max	最大值
返回值	-

取得进度条当前值

函数名	px_int PX_Object_ProcessBarGetValue(PX_Object *pProcessBar);
功能	取得进度条当前值
Object	对象指针
返回值	进度条当前值

图片框

创建图片框

函数名	PX_Object *PX_Object_ImageCreate(px_memorypool *mp,PX_Object *Parent,px_int x,px_int y,px_texture *ptex);
功能	创建图片框
mp	内存池
Parent	父对象
x,y	平面坐标
width,height	宽度,高度
pTex	纹理指针
	如果创建成功返回对象指针,否者返回 PX_NULL

取得对象数据

函数名	PX_Object_Image *PX_Object_GetImage(PX_Object *Object);
功能	取得图片框对象数据
Object	对象指针
返回值	如果成功返回对象数据指针,否者返回 PX_NULL

滑动框

创建滑动框

函数名	PX_Object *PX_Object_SliderBarCreate(px_memorypool *mp,PX_Object *Parent,px_int x,px_int
-----	--

	y,px_int Width,px_int Height,enum PX_OBJECT_SLIDERBAR_TYPE Type,enum PX_OBJECT_SLIDERBAR_STYLE style);
功能	创建滑动框
mp	内存池
Parent	父对象
x,y	平面坐标
width,height	宽度,高度
PX_OBJECT_SLIDERBAR_TYPE	滑动框类型(水平,垂直)
	如果创建成功返回对象指针,否者返回 PX_NULL

取得对象数据

函数名	PX_Object_SliderBar *PX_Object_GetSliderBar(PX_Object *Object);
功能	取得滑动框对象数据
Object	对象指针
返回值	如果成功返回对象数据指针,否者返回 PX_NULL

设置滑动框当前值

函数名	px_void PX_Object_SliderBarSetValue(PX_Object *pSliderBar,px_int Value);
功能	设置滑动框当前值
Object	对象指针
value	当前值
返回值	-

设置滑动框值域

函数名	px_void PX_Object_SliderBarSetRange(PX_Object
-----	--

	<code>*pSliderBar,px_int Min,px_int Max);</code>
功能	设置滑动框值域
Object	对象指针
Min	最小值
Max	最大值
返回值	-

读取滑动框值

函数名	<code>px_int PX_Object_SliderBarGetValue(PX_Object *pSliderBar);</code>
功能	读取滑动框值
Object	对象指针
返回值	当前滑动框的值

设置滑动框背景颜色

函数名	<code>px_void PX_Object_SliderBarSetBackgroundColor(PX_Object *pSliderBar,px_color color);</code>
功能	设置滑动框背景颜色
Object	对象指针
color	背景颜色
返回值	当前滑动框的值

设置滑动框颜色

函数名	<code>px_void PX_Object_SliderBarSetColor(PX_Object *pSliderBar,px_color color);</code>
功能	设置滑动框颜色
Object	对象指针
color	颜色
返回值	

设置滑动框按钮大小

函数名	px_void PX_Object_SliderBarSetSliderButtonLength(PX_Object *pSliderBar,px_int length);
功能	取得滑动框当前值
Object	对象指针
length	按钮长度
返回值	

按钮

创建按钮

函数名	PX_Object *PX_Object_PushButtonCreate(px_memorypool *mp,PX_Object *Parent,px_int x,px_int y,px_int Width,px_int Height,const px_char *Text,PX_FontModule *fontmodule);
功能	创建按钮
mp	内存池
Parent	父对象
x,y	平面坐标
width,height	宽度,高度
text	按钮文本
fontmodule	字模
	如果创建成功返回对象指针,否者返回PX_NULL

取得对象数据

函数名	PX_Object_PushButton * PX_Object_GetPushButton(PX_Object
-----	---

	*Object);
功能	取得按钮对象数据
Object	对象指针
返回值	如果成功返回对象数据指针,否者返回 PX_NULL

取得按钮文本

函数名	px_char PX_Object_PushButtonGetText(PX_Object *PushButton);
功能	取得按钮文本
Object	对象指针
返回值	返回文本指针

设置按钮文本

函数名	px_void PX_Object_PushButtonSetText(PX_Object *pObject,const px_char *Text);
功能	设置按钮文本
Object	对象指针
Text	想要设置的文本
返回值	-

设置按钮背景颜色

函数名	px_void PX_Object_PushButtonSetBackgroundColor(PX_Object *pObject,px_color Color);
功能	设置按钮背景颜色
Object	对象指针
color	想要设置背景颜色
返回值	-

设置按钮指针颜色

函数名	px_void PX_Object_PushButtonSetCursorColor(PX_Object *pObject,px_color Color);
功能	设置按钮背景指针颜色(当鼠标移动到按钮上显示的背景颜色)
Object	对象指针
color	想要设置指针颜色
返回值	-

设置按钮风格

函数名	px_void PX_Object_PushButtonSetStyle(PX_Object *pObject,PX_OBJECT_PUSHBUTTON_STYLE style);
功能	设置按钮风格
Object	对象指针
style	想要设置的风格(矩形按钮,圆角按钮)
返回值	-

设置按钮按下颜色

函数名	px_void PX_Object_PushButtonSetPushColor(PX_Object *pObject,px_color Color);
功能	设置按钮按下颜色
Object	对象指针
color	想要设置的按下颜色
返回值	-

设置按钮边框颜色

函数名	px_void PX_Object_PushButtonSetBorderColor(PX_Object *pObject,px_color Color);
功能	设置按钮边框颜色
Object	对象指针
color	想要设置的边框颜色
返回值	-

设置按钮边框是否显示边框

函数名	px_void PX_Object_PushButtonSetBorder(PX_Object *Object,px_bool Border);
功能	设置按钮是否显示边框
Object	对象指针
Border	PX_TRUE 表示显示,否者(PX_FALSE)表示不显示
返回值	-

设置按钮文本颜色

函数名	px_void PX_Object_PushButtonSetTextColor(PX_Object *pObject,px_color Color);
功能	设置按钮文本颜色
Object	对象指针
color	想要设置的文本颜色
返回值	-

设置按钮背景图片

函数名	px_void
-----	---------

	PX_Object_PushButtonSetTexture(PX_Object *pObject,px_texture *texture);
功能	设置按钮背景图片(这个图片会居中显示)
Object	对象指针
texture	纹理指针
返回值	-

设置按钮背景轮廓

函数名	px_void PX_Object_PushButtonSetShape(PX_Object *pObject,px_shape *pshape);
功能	设置按钮背景轮廓(这个轮廓会居中显示)
Object	对象指针
pShape	轮廓指针
返回值	-

文本编辑框

创建文本编辑框

函数名	PX_Object* PX_Object_EditCreate(px_memorypool *mp, PX_Object *Parent,px_int x,px_int y,px_int Width,px_int Height,PX_FontModule *fontModule);
功能	创建文本编辑框
mp	内存池
Parent	父对象
x,y	平面坐标
width,height	宽度,高度
fontmodule	字模
	如果创建成功返回对象指针,否者返回 PX_NULL

取得对象数据

函数名	PX_Object_Edit PX_Object_GetEdit(PX_Object *Object);
功能	取得文本编辑框对象数据
Object	对象指针
返回值	如果成功返回对象数据指针, 否者返回 PX_NULL

取得文本编辑框文本

函数名	px_char * PX_Object_EditGetText(PX_Object *pObject);
功能	取得文本编辑框当前文本
Object	对象指针
返回值	返回文本字符串指针

取得文本编辑框最大(字节)长度

函数名	px_void PX_Object_EditSetMaxTextLength(PX_Object *pObject,px_int max_length);
功能	取得文本编辑框最大长度
Object	对象指针
返回值	-

设置文本编辑框文本

函数名	px_void PX_Object_EditSetText(PX_Object *pObject,const px_char *Text);
功能	取得文本编辑框文本
Object	对象指针
Text	文本
返回值	-

追加文本编辑框文本

函数名	px_void PX_Object_EditAppendText(PX_Object *pObject,const px_char *Text);
功能	追加文本编辑框文本
Object	对象指针
Text	文本
返回值	-

激活文本编辑框

函数名	px_void PX_Object_EditSetFocus(PX_Object *pObject,px_bool OnFocus);
功能	激活文本编辑框
Object	对象指针
OnFocus	PX_TRUE 表示激活,否者不激活
返回值	-

文本编辑框密码模式

函数名	px_void PX_Object_EditSetPasswordStyle(PX_Object *pObject,px_uchar Enabled);
功能	文本编辑框密码模式
Object	对象指针
Enabled	PX_TRUE 表示启用,否者不启用
返回值	-

文本编辑框背景颜色

函数名	px_void PX_Object_EditSetBackgroundColor(PX_Object *pObject,px_color Color);
-----	--

功能	文本编辑框背景颜色
Object	对象指针
color	背景颜色
返回值	-

文本编辑框边框颜色

函数名	px_void PX_Object_EditSetBorderColor(PX_Object *pObject,px_color Color);
功能	文本编辑框边框颜色
Object	对象指针
color	边框颜色
返回值	-

文本编辑框指针颜色

函数名	px_void PX_Object_EditSetCursorColor(PX_Object *pObject,px_color Color);
功能	文本编辑框指针颜色(鼠标移动到编辑框时的背景颜色)
Object	对象指针
color	指针颜色
返回值	-

文本编辑框文本颜色

函数名	px_void PX_Object_EditSetTextColor(PX_Object *pObject,px_color Color);
功能	文本编辑框文本颜色
Object	对象指针
color	文本颜色
返回值	-

文本编辑框过滤器

函数名	px_void PX_Object_EditSetLimit(PX_Object *pObject,const px_char *Limit);
功能	文本编辑框过滤器(仅允许输入 Limit 中包含的 ANSI 字符)
Object	对象指针
Limit	过滤器字符串,包含所有允许输入的 ANSI 字符
返回值	-

文本编辑框显示边框

函数名	px_void PX_Object_EditSetBorder(PX_Object *pObj,px_bool Border);
功能	文本编辑框显示边框
Object	对象指针
Border	PX_TRUE 表示显示,PX_FALSE 表示不显示
返回值	-

自动换行

函数名	px_void PX_Object_EditAutoNewLine(PX_Object *pObject,px_bool b,px_int AutoNewLineSpacing);
功能	是否允许文本编辑框自动换行(当字符超过文本编辑框宽度的时候)
Object	对象指针
b	PX_TRUE 表示允许,PX_FALSE 表示不允许
AutoNewLineSpacing	距离最右边框多少像素距离时换行
返回值	-

列表框

创建列表框

函数名	PX_Object * PX_Object_ListCreate(px_memorypool *mp, PX_Object *Parent,px_int x,px_int y,px_int Width,px_int Height,px_int ItemHeight,PX_Object_ListItemOnCreate _CreateFunc);
功能	创建列表框
mp	内存池
Parent	父对象
x,y	平面坐标
width,height	宽度,高度
ItemHeight	每个列表条目的高度
CreateFunc	列表条目的创建函数 List 自动创建了一个类型为 PX_Object_ListItem 的对象, 你需要在这个创建函数中,设定这个对象的 更新/渲染/释放 函数,你可以通过这个对象 类型的 pData 获取对应条目的信息数据
	如果创建成功返回对象指针,否则返回 PX_NULL

取得对象数据

函数名	PX_Object_ListItem * PX_Object_GetListItem(PX_Object *Object);
功能	取得列表框对象数据
Object	对象指针
返回值	如果成功返回对象数据指针,否则返回 PX_NULL

清空列表框

函数名	px_void PX_Object_ListClear(PX_Object *pListObj);
功能	清空列表框
Object	对象指针
返回值	-

列表框添加一个条目数据

函数名	px_void PX_Object_ListAdd(PX_Object *pListObj,px_void *ptr);
功能	列表框添加一个条目数据
Object	对象指针
ptr	条目数据的指针,这个条目数据必须是一个生存期有效的数据指针
返回值	-

取得条目数据

函数名	px_void *PX_Object_ListGetItem(PX_Object *pListObject,px_int index);
功能	取得条目数据
Object	对象指针
index	条目索引
返回值	条目数据指针

删除条目(数据)

函数名	px_void PX_Object_ListRemoveItem(PX_Object *pListObject,px_int index);
功能	删除条目数据
Object	对象指针

index	条目索引
返回值	-

设置背景颜色

函数名	px_void PX_Object_ListSetBackgroundColor(PX_Object *pListObject,px_color color);
功能	设置背景颜色
Object	对象指针
index	要设置的背景颜色
返回值	-

设置边框颜色

函数名	px_void PX_Object_ListSetBorderColor(PX_Object *pListObject,px_color color);
功能	设置边框颜色
Object	对象指针
index	要设置的边框颜色
返回值	-

虚拟键盘

创建全键虚拟键盘

函数名	PX_Object* PX_Object_VirtualKeyBoardCreate(px_memorypool *mp, PX_Object *Parent,px_int x,px_int y,px_int width,px_int height);
功能	创建虚拟键盘
mp	内存池

Parent	父对象
x,y	平面坐标
width,height	宽度,高度
	如果创建成功返回对象指针,否者返回 PX_NULL

设置全键虚拟键盘背景颜色

函数名	px_void PX_Object_VirtualKeyBoardSetBackgroundColor(PX_Object *pObject,px_color Color);
功能	设置虚拟键盘背景颜色
Object	对象指针
color	背景颜色
返回值	-

设置全键虚拟键盘边框颜色

函数名	px_void PX_Object_VirtualKeyBoardSetBorderColor(PX_Object *pObject,px_color Color);
功能	设置虚拟键盘边框颜色
Object	对象指针
color	边框颜色
返回值	-

设置全键虚拟键盘指针颜色

函数名	px_void PX_Object_VirtualKeyBoardCursorColor(PX_Object *pObject,px_color Color);
功能	设置虚拟键盘指针颜色
Object	对象指针
color	指针颜色
返回值	-

设置全键虚拟键盘按下颜色

函数名	px_void PX_Object_VirtualKeyBoardPushColor(PX_Object *pObject,px_color Color);
功能	设置虚拟键盘按下颜色
Object	对象指针
color	按下颜色
返回值	-

键盘返回值

函数名	px_char PX_Object_VirtualKeyBoardGetCode(PX_Object *pObject);
功能	取得键盘返回值
Object	对象指针
返回值	如果虚拟键盘有被按键,那么将会返回一个非0的 ANSI 字符(或控制符)

创建数字虚拟键盘

函数名	PX_Object* PX_Object_VirtualNumberKeyBoardCreate(px_memorypool *mp, PX_Object *Parent,px_int x,px_int y,px_int width,px_int height);
功能	创建数字虚拟键盘
mp	内存池
Parent	父对象
x,y	平面坐标
width,height	宽度,高度
	如果创建成功返回对象指针,否者返回 PX_NULL

设置数字虚拟键盘背景颜色

函数名	px_void PX_Object_VirtualNumberKeyBoardSetBackgroundColor(PX_Object *pObject,px_color Color);
功能	设置虚拟键盘背景颜色
Object	对象指针
color	颜色
返回值	-

设置数字虚拟键盘边框颜色

函数名	px_void PX_Object_VirtualNumberKeyBoardSetBorderColor(PX_Object *pObject,px_color Color);
功能	设置虚拟键盘边框颜色
Object	对象指针
color	颜色
返回值	-

设置数字虚拟键盘指针颜色

函数名	px_void PX_Object_VirtualNumberKeyBoardCursorColor(PX_Object *pObject,px_color Color);
功能	设置虚拟键盘指针颜色
Object	对象指针
color	颜色
返回值	-

设置数字虚拟键盘按下颜色

函数名	px_void PX_Object_VirtualNumberKeyBoardPushColor(PX_Object
-----	--

	<code>*pObject,px_color Color);</code>
功能	设置虚拟键盘按下颜色
Object	对象指针
color	颜色
返回值	-

键盘返回值

函数名	<code>px_char PX_Object_VirtualNumberKeyBoardGetCode(PX_Object *pObject);</code>
功能	取得键盘返回值
Object	对象指针
返回值	如果虚拟键盘有被按键,那么将会返回一个非 0 的 ANSI 字符(或控制符)

勾选框

创建勾选框

函数名	<code>PX_Object * PX_Object_CheckBoxCreate(px_memorypool *mp, PX_Object *Parent,px_int x,px_int y,px_int Width,px_int Height,const char text[],PX_FontModule *fm);</code>
功能	创建勾选框
mp	内存池
Parent	父对象
x,y	平面坐标
width,height	宽度,高度
text	勾选框文本
fm	字模
	如果创建成功返回对象指针,否者返回 PX_NULL

取得对象数据

函数名	PX_Object_CheckBox *PX_Object_GetCheckBox(PX_Object *Object);
功能	取得勾选框对象数据
Object	对象指针
返回值	如果成功返回对象数据指针,否者返回 PX_NULL

取得勾选框勾选状态

函数名	px_bool PX_Object_CheckBoxGetCheck(PX_Object *Object);
功能	取得勾选框勾选状态
Object	对象指针
返回值	如果被勾选状态返回 PX_TRUE,否者返回 PX_FALSE

设置背景颜色

函数名	px_void PX_Object_CheckBoxSetBackgroundColor(PX_Object *Object,px_color clr);
功能	设置背景颜色
Object	对象指针
clr	背景颜色
返回值	

设置边框颜色

函数名	px_void PX_Object_CheckBoxSetBorderColor(PX_Object
-----	---

	<code>*Object,px_color clr);</code>
功能	设置背景颜色
Object	对象指针
clr	背景颜色
返回值	

设置按下颜色

函数名	<code>px_void PX_Object_CheckBoxSetPushColor(PX_Object *Object,px_color clr);</code>
功能	设置按下颜色
Object	对象指针
clr	按下颜色
返回值	

设置指针颜色

函数名	<code>px_void PX_Object_CheckBoxSetCursorColor(PX_Object *Object,px_color clr);</code>
功能	设置指针颜色
Object	对象指针
clr	指针颜色
返回值	

设置文本

函数名	<code>px_void PX_Object_CheckBoxSetText(PX_Object *Object,const px_char text[]);</code>
功能	设置文本
Object	对象指针
text	文本
返回值	

设置文本颜色

函数名	px_void PX_Object_CheckBoxSetTextColor(PX_Object *Object,px_color clr);
功能	设置文本颜色
Object	对象指针
clr	文本颜色
返回值	

设置勾选状态

函数名	px_void PX_Object_CheckBoxSetCheck(PX_Object *Object,px_bool check);
功能	设置勾选状态
Object	对象指针
check	PX_TRUE 表示勾选,PX_FALSE 表示非勾选
返回值	

菜单

创建菜单

函数名	PX_Object * PX_Object_MenuCreate(px_memorypool *mp,PX_Object *Parent,px_int x,int y,px_int width,PX_FontModule *fontmodule);
功能	创建菜单
mp	内存池
Parent	父对象
x,y	平面坐标
width,height	宽度,高度

fm	字模
	如果创建成功返回对象指针,否者返回 PX_NULL

取得菜单根节点

函数名	PX_Object_Menu_Item * PX_Object_MenuGetRootItem(PX_Object *pMenuObject);
功能	取得菜单根节点
pMenuObject	对象指针
返回值	菜单的根节点指针

添加菜单节点

函数名	PX_Object_Menu_Item * PX_Object_MenuAddItem(PX_Object *pMenuObject,PX_Object_Menu_Item *parent,const px_char Text[],PX_MenuExecuteFunc _callback,px_void *ptr);
功能	添加菜单节点
pMenuObject	对象指针
parent	父节点指针
Text	节点文本
_callback	回调函数,当菜单这个节点被点击时会执行该回调函数
ptr	回调函数指针
返回值	新创建的节点指针

下拉框

创建下拉选择框

函数名	PX_Object * PX_Object_SelectBarCreate(px_memorypool
-----	--

	<code>*mp,PX_Object *Parent,px_int x,int y,px_int width,px_int height,PX_FontModule *fontmodule);</code>
功能	创建下拉框
mp	内存池
Parent	父对象
x,y	平面坐标
width,height	宽度,高度
fm	字模
	如果创建成功返回对象指针,否者返回 PX_NULL

取得对象数据

函数名	<code>PX_Object_SelectBar *PX_Object_GetSelectBar(PX_Object *pSelectBar);</code>
功能	取得对象数据
pSelecrBar	对象指针
返回值	如果成功返回对象数据指针,否者返回 PX_NULL

添加条目

函数名	<code>px_int PX_Object_SelectBarAddItem(PX_Object *PX_Object_SelectBar,const px_char Text[]);</code>
功能	添加一个条目
pSelecrBar	对象指针
Text	条目文本
返回值	如果成功返回条目索引,否者返回-1

移除条目

函数名	<code>px_void PX_Object_SelectBarRemoveltem(PX_Object *PX_Object_SelectBar,px_int index);</code>
功能	移除一个条目
pSelecrBar	对象指针
index	条目索引

返回值	-
-----	---

取得条目索引

函数名	px_int PX_Object_SelectBarGetItemIndexByText(PX_Object *pObject,const px_char Text[]);
功能	通过条目文本搜索一个条目索引
pSelecrBar	对象指针
index	条目索引
返回值	如果找到返回条目索引,否者返回-1,如果有多个同名条目仅会返回最先的索引

设置最大显示条目格式

函数名	px_void PX_Object_SelectBarSetDisplayCount(PX_Object *pObject,px_int count);
功能	设置最大显示条目格式,超过条目个数下拉框将会以滚动条的形式进行调整
pSelecrBar	对象指针
count	最大显示条目个数
返回值	-

设置背景颜色

函数名	px_void PX_Object_SelectBarSetBackgroundColor(PX_Object *pObject,px_color color);
功能	设置背景颜色
Object	对象指针
clr	背景颜色
返回值	

设置边框颜色

函数名	px_void PX_Object_SelectBarSetBorderColor(PX_Object *pObject,px_color color);
功能	设置背景颜色
Object	对象指针
clr	背景颜色
返回值	

设置文本颜色

函数名	px_void PX_Object_SelectBarSetFontColor(PX_Object *pObject,px_color color);
功能	设置文本颜色
Object	对象指针
clr	文本颜色
返回值	

设置指针颜色

函数名	px_void PX_Object_SelectBarSetCursorColor(PX_Object *pObject,px_color color);
功能	设置指针颜色
Object	对象指针
clr	指针颜色
返回值	

单选框

创建单选框

函数名	PX_Object * PX_Object_RadioButtonCreate(px_memorypool *mp, PX_Object *Parent,px_int x,px_int y,px_int Width,px_int Height,const char text[],PX_FontModule *fm);
功能	创建单选框
mp	内存池
Parent	父对象
x,y	平面坐标
width,height	宽度,高度
text	单选框文本
fm	字模
	如果创建成功返回对象指针,否者返回 PX_NULL

单选框选中状态

函数名	px_bool PX_Object_RadioButtonGetCheck(PX_Object *Object);
功能	取得单选框的选中状态
Object	对象指针
	如果选中返回 PX_TRUE,否者 PX_FALSE

设置单选框背景颜色

函数名	px_void PX_Object_RadioButtonSetBackgroundColor(PX_Object *Object,px_color clr);
功能	设置单选框背景颜色
Object	对象指针
clr	颜色

设置单选框边框颜色

函数名	px_void PX_Object_RadioButtonSetBorderColor(PX_Object *Object,px_color clr);
功能	设置单选框边框颜色
Object	对象指针
clr	颜色

设置单选框按下颜色

函数名	px_void PX_Object_RadioButtonSetPushColor(PX_Object *Object,px_color clr);
功能	设置单选框按下颜色
Object	对象指针
clr	颜色

设置单选框指针颜色

函数名	px_void PX_Object_RadioButtonSetCursorColor(PX_Object *Object,px_color clr);
功能	设置单选框指针颜色
Object	对象指针
clr	颜色

设置单选框文本

函数名	px_void PX_Object_RadioButtonSetText(PX_Object *Object,const px_char text[]);
功能	设置单选框文本
Object	对象指针
text	文本

设置单选框文本颜色

函数名	px_void PX_Object_RadioButtonSetTextColor(PX_Object *Object,px_color clr);
功能	设置单选框文本颜色
Object	对象指针
clr	文本颜色

设置单选框选中状态

函数名	px_void PX_Object_RadioButtonSetCheck(PX_Object *Object,px_bool check);
功能	设置单选框选中状态
Object	对象指针
check	选中状态

文件浏览器

创建文件浏览器

函数名	<pre>PX_Object PX_Object_ExplorerCreate(px_memorypool *mp, PX_Object *Parent,px_int x,px_int y,px_int Width,px_int Height,PX_FontModule *fm, PX_ExplorerGetPathFolderCount _func_gpfdc, PX_ExplorerGetPathFileCount _func_gpfec, PX_ExplorerGetPathFolderName _func_gpfdn, PX_ExplorerGetPathFileName _func_gpfcn, const px_char path[260]);</pre>
功能	创建文件浏览器
mp	内存池
Parent	父对象
x,y	平面坐标
width,height	宽度,高度
fm	字模
_func_gpfdc	读路径文件夹个数的回调函数
_func_gpfec	读路径文件夹文件个数的回调函数
_func_gpfdn	读路径文件夹名称的回调函数
_func_gpfcn	读路径文件夹文件名的回调函数
	如果创建成功返回对象指针,否者返回 PX_NULL

设置文件浏览器边框颜色

函数名	<pre>px_void PX_Object_ExplorerSetBorderColor(PX_Object *Object,px_color clr);</pre>
功能	设置文件浏览器边框颜色
Object	对象指针
clr	颜色

设置文件浏览器按下颜色

函数名	px_void PX_Object_ExplorerSetPushColor(PX_Object *Object,px_color clr);
功能	设置文件浏览器按下颜色
Object	对象指针
clr	颜色

设置文件浏览器指针颜色

函数名	px_void PX_Object_ExplorerSetCursorColor(PX_Object *Object,px_color clr);
功能	设置文件浏览器指针颜色
Object	对象指针
clr	颜色

设置文件浏览器文本颜色

函数名	px_void PX_Object_ExplorerSetTextColor(PX_Object *Object,px_color clr);
功能	设置文件浏览器指针颜色
Object	对象指针
clr	颜色

刷新文件浏览器

函数名	px_void PX_Object_ExplorerRefresh(PX_Object *Object);
功能	刷新文件浏览器
Object	对象指针

--	--

文件浏览器选中个数

函数名	px_int PX_Object_ExplorerGetSelectedCount(PX_Object *Object);
功能	取得文件浏览器选中个数
Object	对象指针
返回值	选中个数

文件浏览器路径

函数名	px_void PX_Object_ExplorerGetPath(PX_Object *Object,px_char path[PX_EXPLORER_MAX_PATH_LEN],px_int index);
功能	取得文件浏览器选中路径
Object	对象指针
index	如果文件浏览器选中多个文件,则编码为 0-N-1
path	输出路径
返回值	

打开文件浏览器

函数名	px_void PX_Object_ExplorerOpen(PX_Object *Object);
功能	打开文件浏览器
Object	对象指针
返回值	

关闭文件浏览器

函数名	px_void PX_Object_ExplorerClose(PX_Object *Object);
功能	关闭文件浏览器
Object	对象指针
返回值	

设置文件浏览器的过滤参数

函数名	px_void PX_Object_ExplorerSetFilter(PX_Object *Object,const px_char *filter);
功能	设置文件浏览器的过滤参数
Object	对象指针
filter	过滤参数,例如 "*.png\0*.jpg\0"以\0 隔开
返回值	

设置文件浏览器的最大选中大小

函数名	px_void PX_Object_ExplorerSetMaxSelectCount(PX_Object *Object,int selectCount);
功能	设置文件浏览器的最大选中大小
Object	对象指针
selectCount	最大选中大小,默认为 1,多选则为多值
返回值	

子窗口

创建子窗口

函数名	PX_Object *
-----	-------------

	PX_Object_WidgetCreate(px_memorypool *mp,PX_Object *Parent,int x,int y,int width,int height,const px_char title[],PX_FontModule *fontmodule);
功能	创建子窗口
mp	内存池
Parent	父对象
x,y	平面坐标
width,height	宽度,高度
title	标题
fm	字模
	如果创建成功返回对象指针,否者返回 PX_NULL

显示子窗口

函数名	px_void PX_Object_WidgetShow(PX_Object *pObject);
功能	显示子窗口
pObject	对象指针

隐藏子窗口

函数名	px_void PX_Object_WidgetHide(PX_Object *pObject);
功能	隐藏子窗口
pObject	对象指针

隐藏/显示子窗口关闭按钮

函数名	px_void PX_Object_WidgetShowHideCloseButton(PX_Object *pObject,px_bool show);
功能	隐藏/显示子窗口关闭按钮

pObject	对象指针
show	是否显示

模态子窗口

函数名	px_void PX_Object_WidgetSetModel(PX_Object *Object,px_bool model);
功能	设置是否为模态子窗口
pObject	对象指针
show	是否模态窗口

取得子窗口的对象指针

函数名	PX_Object * PX_Object_WidgetGetRoot(PX_Object *pObject);
功能	取得子窗口的对象指针
pObject	对象指针

设置子窗体边框颜色

函数名	px_void PX_Object_WidgetSetBorderColor(PX_Object *pObject,px_color clr);
功能	设置子窗体边框颜色
pObject	对象指针
clr	要设置的颜色

设置子窗体拖动条颜色

函数名	px_void PX_Object_WidgetSetBarColor(PX_Object *pObject,px_color clr);
功能	设置子窗体拖动条颜色
pObject	对象指针
clr	要设置的颜色

设置子窗体背景颜色

函数名	px_void PX_Object_WidgetSetBackgroundColor(PX_Object *pObject,px_color clr);
功能	设置子窗体背景颜色
pObject	对象指针
clr	要设置的颜色

设置子窗体焦点颜色

函数名	px_void PX_Object_WidgetSetFocusColor(PX_Object *pObject,px_color clr);
功能	设置子窗体焦点颜色
pObject	对象指针
clr	要设置的颜色

设置子窗体标题文本颜色

函数名	px_void PX_Object_WidgetSetFontColor(PX_Object *pObject,px_color clr);
功能	设置子窗体标题文本颜色

pObject	对象指针
clr	要设置的颜色

设置子窗体标题文本

函数名	px_void PX_Object_WidgetSetTitle(PX_Object *pObject,const px_char title[])
功能	设置子窗体标题文本
pObject	对象指针
title	窗体文本

消息对话框

创建消息对话框

函数名	PX_Object * PX_Object_MessageBoxCreate(px_memorypool *mp,PX_Object *parent,PX_FontModule *fontmodule);
功能	创建消息对话框
mp	内存池
Parent	父对象
fontmodule	字模
	如果创建成功返回对象指针,否则返回 PX_NULL

关闭消息对话框

函数名	px_void PX_Object_MessageBoxClose(PX_Object *pObject);
功能	关闭消息对话框
pObject	对话框对象

确定消息对话框

函数名	px_void PX_Object_MessageBoxAlertOk(PX_Object *pObject,const px_char *message,PX_Object_MessageBoxCallBack func_callback,px_void *ptr);
功能	弹出一个确定消息对话框
pObject	消息对话框对象
message	显示文本
func_callback	点击确定时回调函数
ptr	用户指针

纯消息对话框

函数名	px_void PX_Object_MessageBoxAlert(PX_Object *pObject,const px_char *message);
功能	弹出一个仅显示文本消息对话框,这个对话框需要手动关闭
pObject	消息对话框对象
message	显示文本

确定/取消消息对话框

函数名	px_void PX_Object_MessageBoxAlertYesNo(PX_Object *pObject,const char *Message,PX_Object_MessageBoxCallBack func_yncallback,px_void *yesptr,PX_Object_MessageBoxCallBack func_nocallback,px_void *noptr);
功能	弹出一个确定/取消消息对话框
pObject	消息对话框对象
message	显示文本
func_yncallback	点击确定时的回调函数

yesptr	确定的用户指针
func_nocallback	点击取消的回调函数
nopt	取消的用户指针

输入消息对话框

函数名	px_void PX_Object_MessageBoxInputDialog(PX_Object *pObject,const char *Message,PX_Object_MessageBoxCallBack func_ynescallback,px_void *yesptr,PX_Object_MessageBoxCallBack func_cancelcallback,px_void *cancelptr);
功能	弹出一个输入消息对话框,允许用户输入一行的文本数据
pObject	消息对话框对象
message	显示文本
func_ynescallback	点击确定时的回调函数
yesptr	确定的用户指针
func_nocallback	点击取消的回调函数
nopt	取消的用户指针

取得输入消息对话框的文本数据

函数名	px_char * PX_Object_MessageBoxGetInput(PX_Object *pObject);
功能	取得输入消息对话框的文本数据
pObject	消息对话框对象
message	显示文本
返回值	文本指针

PainterEngine UI 设计框架

基本数据类型

PainterEngine 可以通过定义 PX_UI 从 JSON 文件中加载一个 UI 布局,其 JSON 的格式规范如下
“控件类型” //必选参数,控件类型

```
{
  "id": "控件 id", //可选参数,控件 id
  "x": "控件的 x 坐标", //可选参数,控件 x 坐标,默认为 0
  "y": "控件的 y 坐标", // 可选参数,控件 y 坐标,默认为 0
  "width": "控件宽度", //可选参数,控件宽度,默认为 0
  "height": "控件高度", //可选参数,控件高度,默认为 0
  "length": "控件长度", //可选参数,控件长度,默认为 0
  "halign": "水平对齐方式", //可选参数,水平对齐方式
  "valign": "水平垂直方式", //可选参数,水平垂直方式
}
```

其中,UI 中的基本参数为所有的 UI 控件都有的参数类型,包括以下

UI 参数名	类型	
id	string	控件 id,必须唯一
x	number	可选参数,控件 x 坐标,默认为 0
y	number	可选参数,控件 y 坐标,默认为 0
width	number	可选参数,控件宽度,默认为 0
height	number	可选参数,控件高度,默认为 0
length	number	可选参数,控件长度,默认为 0
halign	string	可选参数,水平对齐方式 left/mid/right
valign	string	可选参数,水平垂直方式 left/mid/right

UI 类型及对应属性

目前,PainterEngine UI 设计框架支持以下几种控件类型,同时其支持属性如下

label 静态文本框

扩展属性	类型	
------	----	--

bordercolor	number 数组	边框颜色,格式为[a,r,g,b]
style	string	风格,如果为 round 则为圆角类型
border	bool	是否有边框
align	string	文本对齐方式 lefttop 左上角对齐 leftmid 左居中对齐 leftbottom 左下对齐 midtop 居中置顶对齐 center 居中对齐 midbottom 居中置底对齐 righttop 右上角对齐 rightmid 靠右居中对齐 rightbottom 右下角对齐

progressbar 进度条

扩展属性	类型	
bordercolor	number 数组	边框颜色,格式为[a,r,g,b]
backgroundcolor	number 数组	背景颜色,格式为[a,r,g,b]
border	bool	是否有边框
max	number	最大值

image 图像框

扩展属性		
无		

sliderbar 滑动框

扩展属性	类型	
max	number	最大值
min	number	最小值
type	string	vertical 垂直滑动框 其它 水平滑动框

style	string	liner 线性滑动框类型 其它 方框滑动框
color	number 数组	颜色,格式为[a,r,g,b]
backgroundcolor	number 数组	背景颜色,格式为[a,r,g,b]
buttonlength	number	滑动块尺寸

pushbutton/cursorbutton 按钮

扩展属性	类型	
fontcolor	number 数组	文本颜色,格式为[a,r,g,b]
backgroundcolor	number 数组	背景颜色,格式为[a,r,g,b]
pushcolor	number 数组	按下颜色,格式为[a,r,g,b]
bordercolor	number 数组	边框颜色,格式为[a,r,g,b]
style	string	round 圆角按钮 其它 矩形按钮
border	bool	是否具有边框
text	string	按钮文本

edit 文本编辑框

扩展属性	类型	
fontcolor	number 数组	文本颜色,格式为[a,r,g,b]
backgroundcolor	number 数组	背景颜色,格式为[a,r,g,b]
cursorcolor	number 数组	指针颜色,格式为[a,r,g,b]
border	bool	是否具有边框
passwordstyle	bool	是否为密码样式,如果是,输入的文本都显示为*号
autonewline	bool	是否自动换行
limit	string	限制文本,仅允许输入限制文本中的字符
maxlength	number	最大长度
style	string	round 圆角编辑框 其它 矩形编辑框

scrollarea 滚动区域

扩展属性	类型	
bordercolor	number 数组	边框颜色,格式为[a,r,g,b]
border	bool	是否具有边框

autotext 自动换行静态文本框

扩展属性	类型	
text	string	文本
fontcolor	number 数组	文本颜色

virtualkeyboard 虚拟键盘

扩展属性	类型	
fontcolor	number 数组	文本颜色,格式为[a,r,g,b]
backgroundcolor	number 数组	背景颜色,格式为[a,r,g,b]
cursorcolor	number 数组	指针颜色,格式为[a,r,g,b]
pushcolor	number 数组	按下颜色,格式为[a,r,g,b]
border	bool	是否具有边框

virtualnumberkeyboard 虚拟数字键盘

扩展属性	类型	
fontcolor	number 数组	文本颜色,格式为[a,r,g,b]
backgroundcolor	number 数组	背景颜色,格式为[a,r,g,b]
cursorcolor	number 数组	指针颜色,格式为[a,r,g,b]
pushcolor	number 数组	按下颜色,格式为[a,r,g,b]
border	bool	是否具有边框

checkbox 选择框

扩展属性	类型	
fontcolor	number 数组	文本颜色,格式为[a,r,g,b]
backgroundcolor	number 数组	背景颜色,格式为[a,r,g,b]
cursorcolor	number 数组	指针颜色,格式为[a,r,g,b]
pushcolor	number 数组	按下颜色,格式为[a,r,g,b]
bordercolor	number 数组	边框颜色,格式为[a,r,g,b]
border	bool	是否具有边框
text	string	选择框文本

radiobox 单选框

扩展属性	类型	
fontcolor	number 数组	文本颜色,格式为[a,r,g,b]
backgroundcolor	number 数组	背景颜色,格式为[a,r,g,b]
cursorcolor	number 数组	指针颜色,格式为[a,r,g,b]
pushcolor	number 数组	按下颜色,格式为[a,r,g,b]
bordercolor	number 数组	边框颜色,格式为[a,r,g,b]

selectbar 下拉框

扩展属性	类型	
fontcolor	number 数组	文本颜色,格式为[a,r,g,b]
backgroundcolor	number 数组	背景颜色,格式为[a,r,g,b]
cursorcolor	number 数组	指针颜色,格式为[a,r,g,b]
bordercolor	number 数组	边框颜色,格式为[a,r,g,b]
style	string	round 圆角下拉框 其它 矩形下拉框
displaycount	number	下拉最大显示个数,超出个数将使用滚动滑块滚动显示
items	string 数组	条目文本,以 0 开始编号
currentindex	number	当前条目,如没有默认为 0

Widget 子窗体

扩展属性	类型	
cursorcolor	number 数组	指针颜色,格式为[a,r,g,b]
pushcolor	number 数组	按下颜色,格式为[a,r,g,b]
bordercolor	number 数组	边框颜色,格式为[a,r,g,b]

游戏世界框架

游戏世界初始化

函数名	<code>px_bool PX_WorldInitialize(px_memorypool *mp,PX_World *World,px_int world_width,px_int world_height,px_int surface_width,px_int surface_height,px_dword calcsize);</code>
功能	初始化一个游戏世界
mp	游戏世界内存池
World	世界实例
world_width	世界宽度(注意,超出世界宽度的 Object 将不再参与物理计算)
world_height	世界高度(注意,超出世界高度的 Object 将不再参与物理计算)
surface_width	视窗宽度大小
surface_height	视窗高度
calcsize	用于物理计算的内存大小(树优化,例如碰撞检测)
返回值	如果成功返回 PX_TRUE,否则 PX_FALSE

取得游戏世界 Object 的个数

函数名	<code>px_int PX_WorldGetCount(PX_World *World);</code>
功能	取得游戏世界 Object 的个数
World	世界实例

返回值	Object 的个数
-----	------------

删除 Object

函数名	px_void PX_WorldRemoveObject(PX_World *world,PX_Object *pObject);
功能	从游戏世界删除一个 Object
World	世界实例
pObject	要删除的指针

函数名	px_void PX_WorldRemoveObjectByIndex(PX_World *world,px_int i_index);
功能	从游戏世界删除一个 Object
World	世界实例
i_index	Object 的索引

在世界查找 Object

函数名	//LIMIT-Only used to ObjectUpdate Function _LIMIT px_int PX_WorldSearchRegion(PX_World *world,px_float centerX,px_float centerY,px_float radius,PX_Object *Object[],px_int MaxSearchCount,px_dword impact_test_type);
功能	划出一个圆形区域,查找区域内的所有 Object
World	世界实例
centerX	区域圆点 x 坐标
centerY	区域圆点 y 坐标
Object	返回数组指针
MaxSearchCount	最大搜索个数
impact_test_type	碰撞过滤器类型
注意	这个函数仅允许在 PX_ObjectUpdate 中被调用

添加 Object

函数名	px_bool PX_WorldAddObject(PX_World *World,PX_Object *pObject);
功能	在游戏世界中添加一个 Object
World	世界实例
pObject	要添加的 Object 指针,添加到游戏世界以后,该 Object 内存由游戏世界接管
返回值	如果成功返回 PX_TRUE,否者 PX_FALSE

更新游戏世界

函数名	px_void PX_WorldUpdate(PX_World *World,px_uint elapsed);
功能	更新游戏世界
World	世界实例
elapsed	上一次 update 经过的毫秒时间

渲染游戏世界

函数名	px_void PX_WorldRender(px_surface *psurface,PX_World *World,px_uint elapsed);
功能	渲染游戏世界
World	世界实例
elapsed	上一次 render 经过的毫秒时间

设置摄像机

函数名	px_void PX_WorldSetCamera(PX_World *World,px_point camera_center_point);
功能	设置摄像机的中心位置
World	世界实例
camera_center_point	摄像机指向的中心位置

设置游戏对象的碰撞类型

函数名	px_void PX_WorldSetImpact(PX_Object *pObj,px_dword type,px_dword impact);
功能	设置游戏对象的碰撞类型
pObj	需要设置的游戏对象
type	对象类型,该对象与 type&impact!=0 的对象发生碰撞
impact	碰撞类型, 该对象与 type&impact!=0 的对象发生碰撞
备注	当一个对象与另一个对象发生碰撞后,两个对象都会收到一个 PX_OBJECT_EVENT_IMPACT 事件, 可以通过 PX_Object_Event_GetPtr 来获取另一个与自己发生碰撞的对象指针

设置游戏世界辅助线间隔

函数名	px_void PX_WorldSetAuxiliaryXYSpacer(PX_World *pw,px_int x,px_int y);
功能	设置游戏世界辅助线
pw	游戏世界实例
x	水平间隔
y	垂直间隔

设置游戏世界辅助线

函数名	px_void PX_WorldEnableAuxiliaryLine(PX_World *pw,px_bool bline);
功能	设置游戏世界辅助线
pw	游戏世界实例
bline	是否显示世界辅助线,PX_FALSE 为不显示,否则显示

设置游戏世界辅助线的颜色

函数名	px_void PX_WorldSetAuxiliaryLineColor(PX_World *pw,px_color color);
功能	设置游戏世界辅助线
pw	游戏世界实例
color	辅助线颜色

世界坐标转换为屏幕坐标

函数名	px_point PX_WorldObjectXYtoScreenXY(PX_World *pw,px_float x,px_float y);
功能	世界坐标转换为屏幕坐标
x,y	游戏世界左边
返回值	屏幕坐标

释放游戏世界

函数名	px_void PX_WorldFree(PX_World *pw);
功能	释放游戏世界
pw	游戏世界实例