Sample Utilities Reference

PlayStation®Vita Programmer Tool Runtime Library Release 3.50

Table of Contents

Controller Utility API **6**
Data Types 7
CtrlUtilData 7
Functions 8
controllerUtilGetLeftStickX 8
controllerUtilGetLeftStickY 9
controllerUtilGetRightStickX 10
controllerUtilGetRightStickY 11
controllerUtilInit 12
controllerUtilInitDefaults 13
controllerUtilIsButtonDown 14
controllerUtilIsButtonPressed 15
controllerUtilIsButtonReleased 16
controllerUtilIsButtonUp 17
controllerUtilSetButtonRepeat 18
controllerUtilShutdown 19
controllerUtilUpdateState 20
Graphics Utility API **21**
Data Types 22
GraphicsUtilConfigParams 22
GraphicsUtilContextData 24
GraphicsUtilHeapSizes 26
GraphicsUtilHeapType 27
Functions 28
graphicsUtilAlloc 28
graphicsUtilClear64BitTarget 29
graphicsUtilClearScreen 30
graphicsUtilCreateRenderTarget 31
graphicsUtilDestroyRenderTarget 32
graphicsUtilFree 33
graphicsUtilInit 34
graphicsUtilSaveDisplayAsBmp 35
graphicsUtilSetDefaultParams 36
graphicsUtilShutdown 37
graphicsUtilUpdateDisplayQueue 38
Callback Functions 39
GraphicsUtilDisplayCallback 39
Constants 40
Define Summary 40
Loader Utility API **42**
Data Types 43
MeshLoaderUtilData 43
MeshLoaderUtilHeapType 44
MeshLoaderUtilSourceMap 45

Functions ... 46
meshLoaderUtilCheckAttributeAvailability ... 46
meshLoaderUtilInit ... 47
meshLoaderUtilInitDefaults ... 48
meshLoaderUtilSetDataBuffer ... 49
meshLoaderUtilShutdown ... 50
Constants ... 51
Define Summary ... 51
Timer Utility API ... 52
Data Types ... 53
TimerUtilData ... 53
Functions ... 54
timerUtilGetTimeDeltaDouble ... 54
timerUtilGetTimeDeltaFloat ... 55
timerUtilGetTotalTimeDouble ... 56
timerUtilGetTotalTimeFloat ... 57
timerUtilInit ... 58
timerUtilReset ... 59
timerUtilStart ... 60
timerUtilStop ... 61
timerUtilUpdate ... 62
Heap Utility API ... 63
Data Types ... 64
HeapUtilContext ... 64
HeapUtilMemBlock ... 65
Functions ... 66
heapUtilAlloc ... 66
heapUtilAllocWithOffset ... 67
heapUtilExtend ... 68
heapUtilFree ... 69
heapUtilInitialize ... 70
heapUtilTerminate ... 71
Font Utility API ... 72
Data Types ... 73
FontUtilFontProperty ... 73
FontUtilImageBuffer ... 74
FontUtilFontSize ... 75
FontUtilImageBufferType ... 76
FontUtilAllocFunc ... 77
FontUtilReallocFunc ... 78
FontUtilFreeFunc ... 79
Functions ... 80
fontUtilInit ... 80
fontUtilShutdown ... 81
fontUtilSetFontSize ... 82
fontUtilGetFontSize ... 83
fontUtilSetFontColor ... 84

fontUtilGetFontColor 85
fontUtilCreateImageBuffer....86
fontUtilFreeImageBuffer 87
fontUtilClearImageBuffer....88
fontUtilPrintUtf8 89
fontUtilPrintUcs2 90
fontUtilGetPrintSizeUtf8 91
fontUtilGetPrintSizeUcs2 92

Sound Utility API 93

Data Types 94
SoundUtilResources 94
SoundUtilVoiceInitParams 95
SoundUtilBussInitParams 96
SoundUtilUpdateThreadInfo 97
SoundUtilSoundInfo 98
SoundUtilWavInfo 99
SoundUtilVolumeInfo....100
SoundUtilEnvelopeInfo 101
SoundUtilNoiseInfo 102
SoundUtilStreamingInfo 103
SoundUtilMalloc 104
SoundUtilFree 105
Functions....106
soundUtilSetMemoryFunc....106
soundUtilInit 107
soundUtilExit 108
soundUtilProcess 109
soundUtilGetWavInfo 110
soundUtilLoadData.... 111
soundUtilUnloadData 112
soundUtilVoiceOpen....113
soundUtilVoicePlay 114
soundUtilVoiceKeyOff 115
soundUtilVoiceClose 116
soundUtilVoicePause 117
soundUtilVoiceResume 118
soundUtilVoiceSetEnvelope....119
soundUtilVoiceGetState 120
soundUtilVoiceGetModuleState 121
soundUtilVoiceSetBuss....122
soundUtilBussCreateReverb....123
soundUtilBussDestroyReverb 124
soundUtilEnableBgmPort....125
soundUtilGetBgmBuffer 126
soundUtilSetBgmData....127
Callback Functions....128
SoundUtilRenderHandler 128

Debug Menu Utility API.... 129

Data Types 130
DebugMenuUtilItemStatus 130
DebugMenuUtilItemTextAttrib 131
DebugMenuUtilItemType 132
DebugMenuUtilItemValueBool 133
DebugMenuUtilItemValueCallback 134
DebugMenuUtilItemValueFloat 135
DebugMenuUtilItemValueInt 136
Functions 137
debugMenuUtilAddItem 137
debugMenuUtilAddStringValueToItem 138
debugMenuUtilCreateMenu 139
debugMenuUtilDecrementItemValue 140
debugMenuUtilIncrementItemValue 141
debugMenuUtilInitCursorPosition 142
debugMenuUtilInitDefaults 143
debugMenuUtilRenderMenu 144
debugMenuUtilSelectDownItem 145
debugMenuUtilSelectUpItem 146
debugMenuUtilShutdown 147
Callback Functions 148
DebugMenuUtilItemCallback 148

Controller Utility API

Data Types

CtrlUtilData

The data structure containing controller information.

Definition

```
#include <sample_utilities/controller_utility.h>
typedef struct CtrlUtilData {
   SceCtrlData *currentCtrlData;
   uint32_t pressedButtonData;
   uint32_t releasedButtonData;
   float deadZone;
   uint32_t buttonRepeatDelay;
   uint32_t port;
   uint32_t numBufs;
   float leftStickXYValues[2];
   float rightStickXYValues[2];
} CtrlUtilData;
```

Members

currentCtrlData	The current frame controller data.
pressedButtonData	The “Pressed” button event data.
releasedButtonData	The “Released” button event data.
deadZone	The controller deadzone variable.
buttonRepeatDelay	The cycle delay between button repeats.
port	The controller port number. This is used internally.
numBufs	The number of buffers that will receive controller data (1 to 64).
leftStickXYValues	The left stick analog X, Y values.
rightStickXYValues	The right stick analog X, Y values.

Description

The data structure containing controller information. This structure is used for initialization and run-time processing of the Controller Utility. The utility functions require this data structure to track button event state, and also to store constants and analog stick values after deadzone adjustments have been made.

Functions

controllerUtilGetLeftStickX

Retrieves left analog stick data in the X plane.

Definition

```
#include <sample_utilities/controller_utility.h>
float controllerUtilGetLeftStickX(
    const CtrlUtilData *pData
);
```

Arguments

[in] *pData*	A pointer to a CtrlUtilData data structure, which contains controller state information.

Return Values

The float value of the left analog stick in the X plane.

Description

Retrieves left analog stick data in the X plane. The data is in float form (-1 < x < +1).

Notes

This function takes deadzone into account.

controllerUtilGetLeftStickY

Retrieves left analog stick data in the Y plane.

Definition

```
#include <sample_utilities/controller_utility.h>
float controllerUtilGetLeftStickY(
    const CtrlUtilData *pData
);
```

Arguments

[in] *pData*	A pointer to a CtrlUtilData data structure, which contains controller state information.

Return Values

The float value of the left analog stick in the Y plane.

Description

Retrieves left analog stick data in the Y plane. The data is in float form (-1 < y < +1).

Notes

This function takes deadzone into account.

controllerUtilGetRightStickX

Retrieves right analog stick data in the X plane.

Definition

```
#include <sample_utilities/controller_utility.h>
float controllerUtilGetRightStickX(
    const CtrlUtilData *pData
);
```

Arguments

pData	A pointer to a CtrlUtilData data structure, which contains controller state information.

Return Values

The float value of the right analog stick in the X plane.

Description

Retrieves right analog stick data in the X plane. The data is in float form (-1 < x < +1).

Notes

This function takes deadzone into account.

controllerUtilGetRightStickY

Retrieves right analog stick data in the Y plane.

Definition

```
#include <sample_utilities/controller_utility.h>
float controllerUtilGetRightStickY(
    const CtrlUtilData *pData
);
```

Arguments

pData	A pointer to a CtrlUtilData data structure, which contains controller state information.

Return Values

The float value of the right analog stick in the Y plane.

Description

Retrieves right analog stick data in the Y plane. The data is in float form (-1 < y < +1).

Notes

This function takes deadzone into account.

controllerUtilInit

Initializes the controller utility.

Definition

```
#include <sample_utilities/controller_utility.h>
int32_t controllerUtilInit(
   CtrlUtilData *pData
);
```

Arguments

[in] *pData*	A pointer to a CtrlUtilData data structure, which contains controller state information.

Return Values

Value	Description
SCE_OK	The operation was successful.
SCE_ERROR_ERRNO_EFAULT	The operation failed because the value of *pData* was NULL, or because an error occurred during controller initialization.

Description

Initializes the controller utility.

controllerUtilInitDefaults

Initializes the CtrlUtilData controller utility structure with default values.

Definition

```
#include <sample_utilities/controller_utility.h>
int32_t controllerUtilInitDefaults(
    CtrlUtilData *pData,
    uint32_t numBufs
);
```

Arguments

[in] *pData*	A pointer to a CtrlUtilData data structure, which contains controller state information.
[in] *numBufs*	The number of buffers that will receive controller state data.

Return Values

Value	Description
`SCE_OK`	The operation was successful.
`SCE_ERROR_ERRNO_EFAULT`	The operation failed because the value of *pData* was `NULL`, or because *numBufs* is out of limits.

Description

Initializes the CtrlUtilData controller utility structure with default values.

controllerUtilIsButtonDown

Queries the controller event state to check the button down state for a particular button.

Definition

```
#include <sample_utilities/controller_utility.h>
bool controllerUtilIsButtonDown(
    const CtrlUtilData *pData,
    uint32_t button
);
```

Arguments

[in] *pData*	A pointer to a `CtrlUtilData` data structure, which contains controller state information.
[in] *button*	An SCE controller button ID from the Controller library.

Return Values

Value	Description
`true`	The button referenced by *button* is down.
`false`	The button referenced by *button* is not down.

Description

Queries the controller event state to check the button down state for a particular button.

controllerUtilIsButtonPressed

Queries the controller event state to see if there are any button pressed events for a particular button.

Definition

```
#include <sample_utilities/controller_utility.h>
bool controllerUtilIsButtonPressed(
    const CtrlUtilData *pData,
    uint32_t button
);
```

Arguments

[in] *pData*	A pointer to a `CtrlUtilData` data structure, which contains controller state information.
[in] *button*	An SCE controller button ID from the Controller library.

Return Values

Value	Description
`True`	The button referenced by *button* is pressed.
`False`	The button referenced by *button* is not pressed.

Description

Queries the controller event state to see if there are any button pressed events for a particular button.

controllerUtilIsButtonReleased

Queries the controller event state to see if there are any button released events for a particular button.

Definition

```
#include <sample_utilities/controller_utility.h>
bool controllerUtilIsButtonReleased(
    const CtrlUtilData *pData,
    uint32_t button
);
```

Arguments

[in] *pData*	A pointer to a CtrlUtilData data structure, which contains controller state information.
[in] *button*	An SCE controller button ID from the Controller library.

Return Values

Value	Description
`true`	The button referenced by *button* is released.
`false`	The button referenced by *button* is not released.

Description

Queries the controller event state to see if there are any button released events for a particular button.

controllerUtilIsButtonUp

Queries the controller event state to check the button up state for a particular button.

Definition

```
#include <sample_utilities/controller_utility.h>
bool controllerUtilIsButtonUp(
    const CtrlUtilData *pData,
    uint32_t button
);
```

Arguments

[in] *pData*	A pointer to a CtrlUtilData data structure, which contains controller state information.
[in] *button*	An SCE controller button ID from the Controller library.

Return Values

Value	Description
`true`	The button referenced by *button* is up.
`false`	The button referenced by *button* is not up.

Description

Queries the controller event state to check the button up state for a particular button.

controllerUtilSetButtonRepeat

Sets the button repeat for a particular button to be on or off.

Definition

```
#include <sample_utilities/controller_utility.h>
int32_t controllerUtilSetButtonRepeat(
   CtrlUtilData *pData,
   uint32_t button,
   bool repeat
);
```

Arguments

[in] *pData*	A pointer to a `CtrlUtilData` data structure, which contains controller state information.
[in] *button*	A SCE controller button ID from the Controller library.
[in] *repeat*	A flag that determines whether to turn button repeat on or off for the specified button.

Return Values

Value	Description
`SCE_OK`	The operation was successful.
`SCE_ERROR_ERRNO_EFAULT`	The operation failed because the value of *pData* was `NULL`, or because an error occurred during button repeat setup.

Description

Sets the button repeat for a particular button to be on or off. Uses the rapid fire functionality of the controller service as a key repeat function.

controllerUtilShutdown

Shuts down the controller utility.

Definition

```
#include <sample_utilities/controller_utility.h>
int32_t controllerUtilShutdown(
    CtrlUtilData *pData
);
```

Arguments

[in] *pData*	A pointer to a CtrlUtilData data structure, which contains controller state information.

Return Values

Value	Description
SCE_OK	The operation was successful.
SCE_ERROR_ERRNO_EFAULT	The operation failed because the value of *pData* was NULL.

Description

Shuts down the controller utility.

controllerUtilUpdateState

Updates the controller state.

Definition

```
#include <sample_utilities/controller_utility.h>
int32_t controllerUtilUpdateState(
    CtrlUtilData *pData
);
```

Arguments

[in] *pData*	A pointer to a CtrlUtilData data structure, which contains controller state information.

Return Values

Value	Description
SCE_OK	The operation was successful.
SCE_ERROR_ERRNO_EFAULT	The operation failed because the value of *pData* was NULL.

Description

Updates the controller state. Using the CtrlUtilData data structure, which is passed as an argument, this function reads the current controller state in terms of button up/down and analog stick values. It also checks the data against the previous state to record button events such as pressed/released events. It then adjusts the recorded analog stick values and adjusts for the deadzone setting.

Graphics Utility API

Data Types

GraphicsUtilConfigParams

The data structure containing parameters/variables used to initialize the libgxm graphics API.

Definition

```
#include <sample_utilities/graphics_utility.h>
typedef struct GraphicsUtilConfigParams {
    SceGxmContextParams ctxParams;
    SceGxmShaderPatcherParams patcherParams;
    SceGxmInitializeParams initializeParams;
    GraphicsUtilHeapSizes usageHeapSizes;
    uint32_t usageFlags;
    SceGxmColorFormat displayColorFormat;
    uint32_t displayPixelFormat;
    uint32_t displayDbgFontFormat;
    uint32_t displayBufferCount;
    uint32_t displayBufferSize;
    SceGxmMultisampleMode msaaMode;
    SceGxmOutputRegisterSize gxmOutputRegSize;
    uint32_t displayWidth;
    uint32_t displayHeight;
    uint32_t displayStrideInPixels;
} GraphicsUtilConfigParams;
```

Members

ctxParams	An instance of a `SceGxmContextParams` structure.
patcherParams	An instance of a `SceGxmShaderPatcherParams` structure.
initializeParams	An instance of a `SceGxmInitializeParams` structure.
usageHeapSizes	An instance of a `GraphicsUtilHeapSizes` structure to store different heap sizes.
usageFlags	A variable for clearscreen and display buffer usage flags.
displayColorFormat	An instance of a `SceGxmColorFormat` structure.
displayPixelFormat	An unsigned integer containing the display pixel format used by the utility display setup.
displayDbgFontFormat	An unsigned integer containing the debug font pixel format used by the utility display setup.
displayBufferCount	An unsigned integer containing the number of display buffers used by the utility display setup.
displayBufferSize	An unsigned integer containing the size of the display buffer used by the utility display setup.
msaaMode	An instance of a `SceGxmMultisampleMode` structure containing multisampling mode data.
gxmOutputRegSize	A member of the `SceGxmOutputRegisterSize` enum which specifies the output register size used by the color surface.
displayWidth	An unsigned integer containing the width of the display buffer in pixels.
displayHeight	An unsigned integer containing the height of the display buffer in pixels.
displayStrideInPixels	An unsigned integer containing the stride of the display buffer in pixels.

Description

The data structure containing parameters/variables used to initialize the libgxm graphics API. Its purpose is to store essential data information used for initializing graphics resources.

GraphicsUtilContextData

The data structure containing context data used within the libgxm graphics API.

Definition

```
#include <sample_utilities/graphics_utility.h>
typedef struct GraphicsUtilContextData {
    SceGxmContext *pContext;
    SceGxmShaderPatcher *pShaderPatcher;
    HeapUtilContext *pContextHeapMem;
    SceUID lpddrHeapMemoryUid;
    SceUID cdramHeapMemoryUid;
    SceGxmColorSurface displaySurface
    [GRAPHICS_UTIL_DEFAULT_DISPLAY_BUFFER_COUNT];
    int32_t displayBufferUid[GRAPHICS_UTIL_DEFAULT_DISPLAY_BUFFER_COUNT];
    SceGxmSyncObject *pDisplayBufferSync
    [GRAPHICS_UTIL_DEFAULT_DISPLAY_BUFFER_COUNT];
    void *pDisplayBufferData[GRAPHICS_UTIL_DEFAULT_DISPLAY_BUFFER_COUNT];
    uint32_t displayBackBufferIndex;
    uint32_t displayFrontBufferIndex;
    void *pDisplayDepthBufferData;
    SceUID displayDepthBufferUid;
    SceGxmDepthStencilSurface displayDepthSurface;
    SceGxmVertexProgram *pClearVertexProgram;
    SceGxmFragmentProgram *pClearFragmentProgram;
    SceGxmFragmentProgram *pClearFragmentProgram64bpp;
    SceGxmShaderPatcherId clearVertexProgramId;
    SceGxmShaderPatcherId clearFragmentProgramId;
    SceUID clearVerticesUid;
    SceUID clearIndicesUid;
    float *pClearVertices;
    uint16_t *pClearIndices;
    const SceGxmProgramParameter *pClearColorParam;
    GraphicsUtilConfigParams configParams;
} GraphicsUtilContextData;
```

Members

pContext	A pointer to a `SceGxmContext` structure.
pShaderPatcher	A pointer to a `SceGxmShaderPatcher` structure.
pContextHeapMem	A pointer to a `HeapUtilContext` structure.
lpddrHeapMemoryUid	The UID of the heap memory blocks to be allocated from the main memory (LPDDR).
cdramHeapMemoryUid	The UID of the heap memory blocks to be allocated from the graphics memory (CDRAM).
displaySurface	An array of `SceGxmColorSurface` structures used in the display buffer setup.
displayBufferUid	An array of integer memory IDs used in the display buffer setup.
pDisplayBufferSync	An array of `SceGxmSyncObject` pointers used in the display buffer setup.
pDisplayBufferData	An array of void pointers that point to the memory used in the display buffer setup.
displayBackBufferIndex	An unsigned integer containing the back buffer index used by the utility display setup.
displayFrontBufferIndex	An unsigned integer containing the front buffer index used by the utility display setup.

pDisplayDepthBufferData	A void pointer that points to the memory used in the depth buffer setup for the display surface.
displayDepthBufferUid	An integer memory ID used in the depth buffer setup for the display surface.
displayDepthSurface	A `SceGxmDepthStencilSurface` structure used in the depth buffer setup for the display surface.
pClearVertexProgram	A vertex shader used for clearing the screen.
pClearFragmentProgram	A fragment shader used for clearing the screen.
pClearFragmentProgram64bpp	A fragment shader used for clearing the 64 bit surface.
clearVertexProgramId	A vertex program ID used for clearing the screen.
clearFragmentProgramId	A fragment program ID used for clearing the screen.
clearVerticesUid	The UID of the vertex data used for clearing the screen.
clearIndicesUid	The UID of the indices data used for clearing the screen.
pClearVertices	The vertex data used for clearing the screen.
pClearIndices	The index data used for clearing the screen.
pClearColorParam	The shader parameter used for clearing the screen.
configParams	A copy of a `GraphicsUtilConfigParams` structure that stores parameters used for initializing the libgxm graphics API.

Description

The data structure containing context data used within the libgxm graphics API. Its purpose is to encapsulate the essential data structures used in graphics processing. The values of this structure are set and modified by the graphics util API. Other functions should not change the values.

GraphicsUtilHeapSizes

The data structure containing members in which are stored the set up sizes for the various heaps used for graphics resources.

Definition

```
#include <sample_utilities/graphics_utility.h>
typedef struct GraphicsUtilHeapSizes {
    uint32_t heapLpddrReadSize;
    uint32_t heapLpddrReadWriteSize;
    uint32_t heapCdramReadWriteSize;
    uint32_t heapVertexUsseSize;
    uint32_t heapFragmentUsseSize;
} GraphicsUtilHeapSizes;
```

Members

heapLpddrReadSize	The size of the heap, on the main memory (uncached), allocated for read only data.
heapLpddrReadWriteSize	The size of the heap, on the main memory (uncached), allocated for read/write data.
heapCdramReadWriteSize	The size of the heap, on the graphics memory, allocated for read/write data.
heapVertexUsseSize	The size of the heap, on the main memory, used for managing vertex USSE allocations.
heapFragmentUsseSize	The size of the heap, on the main memory, used for managing fragment USSE allocations.

Description

The data structure containing members in which are stored the set up sizes for the various heaps used for graphics resources.

GraphicsUtilHeapType

An enumeration which represents the different types of heap that can be created in the Graphics Utility.

Definition

```
#include <sample_utilities/graphics_utility.h>
typedef enum GraphicsUtilHeapType {
   GRAPHICS_UTIL_HEAP_TYPE_LPDDR_R,
   GRAPHICS_UTIL_HEAP_TYPE_LPDDR_RW,
   GRAPHICS_UTIL_HEAP_TYPE_CDRAM_RW,
   GRAPHICS_UTIL_HEAP_TYPE_VERTEX_USSE,
   GRAPHICS_UTIL_HEAP_TYPE_FRAGMENT_USSE
} GraphicsUtilHeapType;
```

Enumeration Values

Macro	Description
`GRAPHICS_UTIL_HEAP_TYPE_LPDDR_R`	Represents a heap, on the main memory, which is used for read purposes only.
`GRAPHICS_UTIL_HEAP_TYPE_LPDDR_RW`	Represents a heap, on the main memory, which is used for both read and write purposes.
`GRAPHICS_UTIL_HEAP_TYPE_CDRAM_RW`	Represents a heap, on the graphics memory, which is used for both read and write purposes.
`GRAPHICS_UTIL_HEAP_TYPE_VERTEX_USSE`	Represents a heap, on the main memory, which is used for managing vertex USSE allocations.
`GRAPHICS_UTIL_HEAP_TYPE_FRAGMENT_USSE`	Represents a heap, on the main memory, which is used for managing fragment USSE allocations.

Description

An enumeration which represents the different types of heap that can be created in the Graphics Utility.

Functions

graphicsUtilAlloc

Allocates graphics memory using the heap utility.

Definition

```
#include <sample_utilities/graphics_utility.h>
void *graphicsUtilAlloc(
    GraphicsUtilContextData *pData,
    GraphicsUtilHeapType type,
    uint32_t size,
    uint32_t alignment
);
```

Arguments

[in] *pData*	A pointer to a `GraphicsUtilContextData` data structure.
[in] *type*	The memory block type to be allocated.
[in] *size*	The size in bytes of the memory to be allocated.
[in] *alignment*	The desired alignment of the allocation.

Return Values

Value	Description
`void*`	A pointer to the allocated memory.
`NULL`	The function returns `NULL` if the memory allocation failed.

Description

Allocates graphics memory using the heap utility.

graphicsUtilClear64BitTarget

Clears the 64 bit surface using the libgxm API.

Definition

```
#include <sample_utilities/graphics_utility.h>
int32_t graphicsUtilClear64BitTarget(
    GraphicsUtilContextData *pData,
    uint32_t color
);
```

Arguments

[in] *pData*	A pointer to a GraphicsUtilContextData data structure.
[in] *color*	An unsigned integer containing color information to clear the screen with.

Return Values

Value	Description
SCE_OK	The operation was successful.
SCE_ERROR_ERRNO_EFAULT	The operation failed because the value of *pData* or *pData->pContext* was NULL.

Description

Clears the 64 bit surface using the libgxm API.

graphicsUtilClearScreen

Clears the screen using the libgxm API.

Definition

```
#include <sample_utilities/graphics_utility.h>
int32_t graphicsUtilClearScreen(
    GraphicsUtilContextData *pData,
    uint32_t color
);
```

Arguments

[in] *pData*	A pointer to a GraphicsUtilContextData data structure.
[in] *color*	An unsigned integer containing color information to clear the screen with.

Return Values

Value	Description
SCE_OK	The operation was successful.
SCE_ERROR_ERRNO_EFAULT	The operation failed because the value of *pData* or *pData->pContext* was NULL.

Description

Clears the screen using the libgxm API.

graphicsUtilCreateRenderTarget

Creates a render target and returns pointer to `SceGxmRenderTarget`.

Definition

```
#include <sample_utilities/graphics_utility.h>
SceGxmRenderTarget *graphicsUtilCreateRenderTarget(
    GraphicsUtilContextData *pData,
    uint32_t width,
    uint32_t height,
    SceGxmMultisampleMode msaaMode
);
```

Arguments

[in] *pData*	A pointer to a `GraphicsUtilContextData` data structure.
[in] *width*	The width of the render target.
[in] *height*	The height of the render target.
[in] *msaaMode*	The multisampling mode data.

Return Values

Value	Description
`SceGxmRenderTarget*`	A valid pointer to `SceGxmRenderTarget` if the operation was successful; otherwise `NULL` is returned.

Description

Creates a render target and returns pointer to `SceGxmRenderTarget`.

graphicsUtilDestroyRenderTarget

Destroys a render target.

Definition

```
#include <sample_utilities/graphics_utility.h>
int32_t graphicsUtilDestroyRenderTarget(
    GraphicsUtilContextData *pData,
    SceGxmRenderTarget *pRenderTarget
);
```

Arguments

[in] *pData*	A pointer to a GraphicsUtilContextData data structure.
[in] *pRenderTarget*	A pointer to SceGxmRenderTarget data structure

Return Values

Value	Description
SCE_OK	The operation was successful.
SCE_ERROR_ERRNO_EFAULT	The operation failed because the value of *pData* or *pRenderTarget* was NULL.

Description

Destroys a render target.

graphicsUtilFree

Frees graphics memory using the heap utility.

Definition

```
#include <sample_utilities/graphics_utility.h>
int32_t graphicsUtilFree(
    GraphicsUtilContextData *pData,
    void *pAddr
);
```

Arguments

[in] *pData*	A pointer to a GraphicsUtilContextData data structure.
[in] *pAddr*	The base address of the memory to be freed.

Return Values

Value	Description
`SCE_OK`	The operation was successful.
`SCE_ERROR_ERRNO_EFAULT`	The operation failed because the value of *pData* was `NULL`, or because an error occurred during memory deallocation.

Description

Frees graphics memory using the heap utility.

graphicsUtilInit

Initializes the libgxm API using variables initialized in a GraphicsUtilConfigParams data structure.

Definition

```
#include <sample_utilities/graphics_utility.h>
int32_t graphicsUtilInit(
    GraphicsUtilContextData *pData,
    GraphicsUtilDisplayCallback pDisplayCallback,
    uint32_t callbackDataSize,
    const GraphicsUtilConfigParams *pConfig
);
```

Arguments

[in] *pData*	A pointer to a GraphicsUtilContextData data structure.
[in] *pDisplayCallback*	A GraphicsUtilDisplayCallback function pointer for the display callback.
[in] *callbackDataSize*	The size of the data that needs to be passed to the callback function.
[in] *pConfig*	A GraphicsUtilConfigParams data structure used for initializing the GraphicsUtilContextData data structure. If this argument is set to NULL, the default setting is applied.

Return Values

Value	Description
SCE_OK	The operation was successful.
SCE_ERROR_ERRNO_EFAULT	The operation failed because the value of *pData* or *pDisplayCallback* was NULL, or because an error occurred during the graphics API initialization.

Description

Initializes the libgxm API using variables initialized in a GraphicsUtilConfigParams data structure.

graphicsUtilSaveDisplayAsBmp

Saves the main display buffer as a `.bmp` file at the specified path.

Definition

```
#include <sample_utilities/graphics_utility.h>
int32_t graphicsUtilSaveDisplayAsBmp(
    GraphicsUtilContextData *pData,
    const char *path
);
```

Arguments

[in] *`pData`*	A pointer to a `GraphicsUtilContextData` data structure.
[in] *`path`*	The path/filename to which the buffer is to be saved (for example, "`app0:filename.bmp`").

Return Values

Value	Description
`SCE_OK`	The operation was successful.
`SCE_ERROR_ERRNO_EFAULT`	The operation failed because the value of *`pData`* or *`path`* was `NULL`, or because an error occurred when saving display as a `.bmp` file.

Description

Saves the main display buffer as a `.bmp` file at the specified path.

graphicsUtilSetDefaultParams

Initializes a `GraphicsUtilConfigParams` data structure with default values.

Definition

```
#include <sample_utilities/graphics_utility.h>
int32_t graphicsUtilSetDefaultParams(
    GraphicsUtilConfigParams *pConfig
);
```

Arguments

[in] *pConfig*	A pointer to a `GraphicsUtilConfigParams` data structure.

Return Values

Value	Description
`SCE_OK`	The operation was successful.
`SCE_ERROR_ERRNO_EFAULT`	The operation failed because the value of *pConfig* was `NULL`.

Description

Initializes a `GraphicsUtilConfigParams` data structure with default values.

graphicsUtilShutdown

Shuts down the libgxm API using variables initialized in a GraphicsUtilContextData data structure.

Definition

```
#include <sample_utilities/graphics_utility.h>
int32_t graphicsUtilShutdown(
    GraphicsUtilContextData *pData
);
```

Arguments

[in] *pData*	A pointer to a GraphicsUtilContextData data structure.

Return Values

Value	Description
SCE_OK	The operation was successful.
SCE_ERROR_ERRNO_EFAULT	The operation failed because the value of *pData* was NULL, or because an error occurred during the graphics API shutdown.

Description

Shuts down the libgxm API using variables initialized in a GraphicsUtilContextData data structure.

graphicsUtilUpdateDisplayQueue

Swaps the front/back buffers using the libgxm API.

Definition

```
#include <sample_utilities/graphics_utility.h>
int32_t graphicsUtilUpdateDisplayQueue(
    GraphicsUtilContextData *pData,
    const void *displayData
);
```

Arguments

[in] *pData*	A pointer to a GraphicsUtilContextData data structure.
[in] *displayData*	A pointer to the data structure to be passed to the display callback function.

Return Values

Value	Description
SCE_OK	The operation was successful.
SCE_ERROR_ERRNO_EFAULT	The operation failed because the value of *pData* was NULL, or because an error occurred during the display queue update.

Description

Swaps the front/back buffers using the libgxm API.

Callback Functions

GraphicsUtilDisplayCallback

The graphics utility callback function that is used to handle the display of a buffer.

Definition

```
#include <sample_utilities/graphics_utility.h>
typedef void (*GraphicsUtilDisplayCallback)(
    const void *pCallbackData
);
```

Arguments

pCallbackData	A pointer to a data structure containing the data to pass to the callback function.

Return Values

None

Description

The graphics utility callback function that is used to handle the display of a buffer.

Constants

Define Summary

Define	Value	Description
GRAPHICS_UTIL_DEFAULT_DISPLAY_BUFFER_COUNT	3	The default number of display buffers.
GRAPHICS_UTIL_DEFAULT_DISPLAY_BUFFER_SIZE	(1*1024*1024)	The default display buffer size.
GRAPHICS_UTIL_DEFAULT_DISPLAY_COLOR_FORMAT	SCE_GXM_COLOR_FORMAT_A8B8G8R8	The default color format.
GRAPHICS_UTIL_DEFAULT_DISPLAY_DBGFONT_FORMAT	SCE_DBGFONT_PIXELFORMAT_A8B8G8R8	The default font pixel format.
GRAPHICS_UTIL_DEFAULT_DISPLAY_HEIGHT	544	The default screen height.
GRAPHICS_UTIL_DEFAULT_DISPLAY_MAX_PENDING_SWAPS	2	The default maximum number of pending display swaps to be allowed.
GRAPHICS_UTIL_DEFAULT_DISPLAY_PIXEL_FORMAT	SCE_DISPLAY_PIXELFORMAT_A8B8G8R8	The default pixel format.
GRAPHICS_UTIL_DEFAULT_DISPLAY_STRIDE_IN_PIXELS	1024	The default display stride.
GRAPHICS_UTIL_DEFAULT_DISPLAY_WIDTH	960	The default screen width.
GRAPHICS_UTIL_HEAP_SIZE_CDRAM_RW	(32*1024*1024)	The default size of the heap, on the graphics memory, which is allocated for both read and write purposes.
GRAPHICS_UTIL_HEAP_SIZE_FRAGMENT_USSE	(8*1024*1024)	The default size of the heap, on the main memory, which is used to manage fragment USSE allocations.
GRAPHICS_UTIL_HEAP_SIZE_LPDDR_R	(64*1024*1024)	The default size of the heap, on the main memory, which is allocated for read purposes only.
GRAPHICS_UTIL_HEAP_SIZE_LPDDR_RW	(32*1024*1024)	The default size of the heap, on the main memory, which is allocated for both read and write purposes.

Define	Value	Description
GRAPHICS_UTIL_HEAP_SIZE_VERTEX_USSE	(8*1024*1024)	The default size of the heap, on the main memory, which is used to manage vertex USSE allocations.
GRAPHICS_UTIL_USE_CLEAR	0x01	A flag to determine whether to use clear screen functionality.
GRAPHICS_UTIL_USE_DISPLAY_BUFFERS	0x02	A flag to determine whether to use display buffer functionality.

Loader Utility API

Data Types

MeshLoaderUtilData

The data structure containing mesh data used for rendering.

Definition

```
#include <sample_utilities/loader_utility.h>
typedef struct MeshLoaderUtilData {
   int meshHandle;
   int meshSize;
   int numVerts;
   float boundingSphereRadius;
   float boundingOrigin[3];
   float boundingBoxMin[3];
   float boundingBoxMax[3];
   float *pMeshVertices;
   uint16_t *pMeshIndices;
   char textureName[MAX_STRING_LENGTH];
   uint8_t *pMeshTextureData;
   SceGxmTexture meshTexture;
} MeshLoaderUtilData;
```

Members

meshHandle	The handle for the loaded mesh.
meshSize	The size of the loaded mesh.
numVerts	The number of vertices in the loaded mesh.
boundingSphereRadius	The radius of the sphere bounding the mesh.
boundingOrigin	The center of the bounding box.
boundingBoxMin	The minimum point included in the bounding box.
boundingBoxMax	The maximum point included in the bounding box.
pMeshVertices	A pointer to the array containing vertex data in an interleaved fashion.
pMeshIndices	A pointer to the array containing index data.
textureName	The texture name to be used for the loaded mesh.
pMeshTextureData	A pointer to texture data stored in memory.
meshTexture	An instance of a `SceGxmTexture` structure.

Description

The data structure containing mesh data used for rendering. This structure is used to store data for the geometry, which includes vertex data, index data and texture data. The vertex data is stored as an interleaved array. The bounding box for the geometry is also contained in the structure.

MeshLoaderUtilHeapType

An enumeration which represents the different types of heap that can be created in the MeshLoader Utility.

Definition

```
#include <sample_utilities/loader_utility.h>
typedef enum MeshLoaderUtilHeapType {
   MESH_LOADER_UTIL_HEAP_TYPE_LPDDR_R
} MeshLoaderUtilHeapType;
```

Enumeration Values

Macro	Description
MESH_LOADER_UTIL_HEAP_TYPE_LPDDR_R	Represents a heap, on the main memory, which is used for read purposes only.

Description

An enumeration which represents the different types of heap that can be created in the MeshLoader Utility.

MeshLoaderUtilSourceMap

An enumeration to represent the different types of source streams available in the COLLADA mesh.

Definition

```
#include <sample_utilities/loader_utility.h>
typedef enum MeshLoaderUtilSourceMap {
    SOURCEMAPPING_VERTEX,
    SOURCEMAPPING_NORMAL,
    SOURCEMAPPING_TEXCOORD,
    SOURCEMAPPING_COLOR,
    SOURCEMAPPING_TANGENT,
    SOURCEMAPPING_BINORMAL,
    SOURCEMAPPING_TEXTANGENT,
    SOURCEMAPPING_TEXBINORMAL,
    NUM_ATTRIBUTE_TYPES
} MeshLoaderUtilSourceMap;
```

Enumeration Values

Macro	Description
SOURCEMAPPING_VERTEX	Represents a vertex position type source stream.
SOURCEMAPPING_NORMAL	Represents a normal type source stream.
SOURCEMAPPING_TEXCOORD	Represents a texture coordinate type source stream.
SOURCEMAPPING_COLOR	Represents a color type source stream.
SOURCEMAPPING_TANGENT	Represents a tangent type source stream.
SOURCEMAPPING_BINORMAL	Represents a binormal type source stream.
SOURCEMAPPING_TEXTANGENT	Represents a texture space tangent type source stream.
SOURCEMAPPING_TEXBINORMAL	Represents a texture space binormal type source stream.
NUM_ATTRIBUTE_TYPES	The number of source stream types.

Description

An enumeration to represent the different types of source streams available in the COLLADA mesh.

Functions

meshLoaderUtilCheckAttributeAvailability

Finds out if a vertex attribute is available in the mesh being loaded.

Definition

```
#include <sample_utilities/loader_utility.h>
int32_t meshLoaderUtilCheckAttributeAvailability(
    MeshLoaderUtilData *pMesh,
    int attrType
);
```

Arguments

[in] *pMesh*	A pointer to a MeshLoaderUtilData data structure.
[in] *attrType*	The type of attribute whose availability is to be checked in the COLLADA file.

Return Values

Value	Description
SCE_OK	The operation was successful and the attribute specified by *attrType* is available.
SCE_ERROR_ERRNO_EFAULT	The operation failed because the value of *pMesh* was NULL, or because the attribute *attrType* is either not defined or is not available.

Description

Finds out if a vertex attribute is available in the mesh being loaded.

meshLoaderUtilInit

Parses the mesh from the file and stores the data in the variables within a MeshLoaderUtilData data structure.

Definition

```
#include <sample_utilities/loader_utility.h>
int32_t meshLoaderUtilInit(
    MeshLoaderUtilData *pMesh,
    const char *fileName
);
```

Arguments

[in] *pMesh*	A pointer to a MeshLoaderUtilData data structure.
[in] *fileName*	A pointer to store the filename to be loaded.

Return Values

Value	Description
SCE_OK	The operation was successful.
SCE_ERROR_ERRNO_EFAULT	The operation failed because the value of *pMesh* or *fileName* was NULL, or because an error occurred when loading the mesh data from the file.

Description

Parses the mesh from the file and stores the data in the variables within a MeshLoaderUtilData data structure.

meshLoaderUtilInitDefaults

Initializes a MeshLoaderUtilData data structure with default values.

Definition

```
#include <sample_utilities/loader_utility.h>
int32_t meshLoaderUtilInitDefaults(
    MeshLoaderUtilData *pMesh
);
```

Arguments

[in] *pMesh*	A pointer to a MeshLoaderUtilData data structure.

Return Values

Value	Description
SCE_OK	The operation was successful.
SCE_ERROR_ERRNO_EFAULT	The operation failed because the value of *pMesh* was NULL.

Description

Initializes a MeshLoaderUtilData data structure with default values.

meshLoaderUtilSetDataBuffer

Sets the vertex streams and index streams corresponding to the parsed geometry data.

Definition

```
#include <sample_utilities/loader_utility.h>
int32_t meshLoaderUtilSetDataBuffer(
    MeshLoaderUtilData *pMesh
);
```

Arguments

[in] *pMesh*	A pointer to a `MeshLoaderUtilData` data structure.

Return Values

Value	Description
`SCE_OK`	The operation was successful.
`SCE_ERROR_ERRNO_EFAULT`	The operation failed because the value of *pMesh* was `NULL`, or because an error occurred during the setup of the mesh data buffer.

Description

Sets the vertex streams and index streams corresponding to the parsed geometry data.

meshLoaderUtilShutdown

Shuts down the loader utility.

Definition

```
#include <sample_utilities/loader_utility.h>
int32_t meshLoaderUtilShutdown(
    MeshLoaderUtilData *pMesh
);
```

Arguments

[in] *pMesh*	A pointer to a MeshLoaderUtilData data structure.

Return Values

Value	Description
SCE_OK	The operation was successful.
SCE_ERROR_ERRNO_EFAULT	The operation failed because the value of *pMesh* was NULL, or because an error occurred during mesh loader shutdown.

Description

Shuts down the loader utility. This function clears the memory allocations for all variables in the MeshLoaderUtilData data structure.

Constants

Define Summary

Define	Value	Description
MESH_LOADER_UTIL_HEAP_SIZE_LPDDR_R	(16*1024*1024)	The default size of the heap, on the main memory, which is allocated for mesh data (read purposes only).

Timer Utility API

Data Types

TimerUtilData

The data structure containing timer information.

Definition

```
#include <sample_utilities/timer_utility.h>
typedef struct TimerUtilData {
    uint64_t initialTimeValue;
    uint64_t lastTimeValue;
    uint64_t currentTimeValue;
    bool isRunning;
} TimerUtilData;
```

Members

initialTimeValue	The time stamp when the timer was started.
lastTimeValue	The last time value from the last call to `timerUtilUpdate`.
currentTimeValue	The current time value after `timerUtilUpdate`.
isRunning	A flag that determines whether to process `timerUtilUpdate`.

Description

The data structure containing timer information. Used for initialization and run-time processing of the timer utility, including initial, previous and current time values.

Functions

timerUtilGetTimeDeltaDouble

Returns the time as a double value elapsed since the last call to timerUtilUpdate.

Definition

```
#include <sample_utilities/timer_utility.h>
double timerUtilGetTimeDeltaDouble(
    TimerUtilData *pTimer,
    double resolution
);
```

Arguments

[in] *pTimer*	A pointer to an instance of TimerUtilData containing time variables.
[in] *resolution*	A variable to determine the timer resolution for the function to return. Valid values are TIMER_UTIL_RESOLUTION_SECONDS, TIMER_UTIL_RESOLUTION_MILLI_SECONDS, or TIMER_UTIL_RESOLUTION_MICRO_SECONDS.

Return Values

The time delta value as a double.

Description

Returns the time as a double value elapsed since the last call to timerUtilUpdate.

timerUtilGetTimeDeltaFloat

Returns the time as a float value elapsed since the last call to `timerUtilUpdate`.

Definition

```
#include <sample_utilities/timer_utility.h>
float timerUtilGetTimeDeltaFloat(
    TimerUtilData *pTimer,
    double resolution
);
```

Arguments

[in] *pTimer*	A pointer to an instance of `TimerUtilData` containing time variables.
[in] *resolution*	A variable to determine the timer resolution for the function to return. Valid values are `TIMER_UTIL_RESOLUTION_SECONDS`, `TIMER_UTIL_RESOLUTION_MILLI_SECONDS`, or `TIMER_UTIL_RESOLUTION_MICRO_SECONDS`.

Return Values

The time delta value as a float.

Description

Returns the time as a float value elapsed since the last call to `timerUtilUpdate`.

timerUtilGetTotalTimeDouble

Returns the total time as a double value elapsed since the timer was started.

Definition

```
#include <sample_utilities/timer_utility.h>
double timerUtilGetTotalTimeDouble(
    TimerUtilData *pTimer,
    double resolution
);
```

Arguments

[in] *pTimer*	A pointer to an instance of `TimerUtilData` containing time variables.
[in] *resolution*	A variable to determine the timer resolution for the function to return. Valid values are `TIMER_UTIL_RESOLUTION_SECONDS`, `TIMER_UTIL_RESOLUTION_MILLI_SECONDS`, or `TIMER_UTIL_RESOLUTION_MICRO_SECONDS`.

Return Values

The time value as a double.

Description

Returns the total time as a double value elapsed since the timer was started.

timerUtilGetTotalTimeFloat

Returns the total time as a float value elapsed since the timer was started.

Definition

```
#include <sample_utilities/timer_utility.h>
float timerUtilGetTotalTimeFloat(
    TimerUtilData *pTimer,
    double resolution
);
```

Arguments

[in] *pTimer*	A pointer to an instance of TimerUtilData containing time variables.
[in] *resolution*	A variable to determine the timer resolution for the function to return. Valid values are `TIMER_UTIL_RESOLUTION_SECONDS`, `TIMER_UTIL_RESOLUTION_MILLI_SECONDS`, or `TIMER_UTIL_RESOLUTION_MICRO_SECONDS`.

Return Values

The time value as a float.

Description

Returns the total time as a float value elapsed since the timer was started.

timerUtilInit

Initializes the timer utility.

Definition

```
#include <sample_utilities/timer_utility.h>
int32_t timerUtilInit(
    TimerUtilData *pTimer
);
```

Arguments

[in] *pTimer*	A pointer to an instance of TimerUtilData containing time variables.

Return Values

Value	Description
SCE_OK	The operation was successful.
SCE_ERROR_ERRNO_EFAULT	The operation failed because the value of *pTimer* was NULL.

Description

Initializes the timer utility.

timerUtilReset

Resets the timer utility.

Definition

```
#include <sample_utilities/timer_utility.h>
int32_t timerUtilReset(
    TimerUtilData *pTimer
);
```

Arguments

[in] *pTimer*	A pointer to an instance of TimerUtilData containing time variables.

Return Values

Value	Description
SCE_OK	The operation was successful.
SCE_ERROR_ERRNO_EFAULT	The operation failed because the value of *pTimer* was NULL.

Description

Resets the timer utility.

timerUtilStart

Starts the timer utility.

Definition

```
#include <sample_utilities/timer_utility.h>
int32_t timerUtilStart(
    TimerUtilData *pTimer
);
```

Arguments

[in] *pTimer*	A pointer to an instance of TimerUtilData containing time variables.

Return Values

Value	Description
SCE_OK	The operation was successful.
SCE_ERROR_ERRNO_EFAULT	The operation failed because the value of *pTimer* was NULL.

Description

Starts the timer utility.

timerUtilStop

Stops the timer utility.

Definition

```
#include <sample_utilities/timer_utility.h>
int32_t timerUtilStop(
    TimerUtilData *pTimer
);
```

Arguments

[in] *pTimer*	A pointer to an instance of TimerUtilData containing time variables.

Return Values

Value	Description
SCE_OK	The operation was successful.
SCE_ERROR_ERRNO_EFAULT	The operation failed because the value of *pTimer* was NULL.

Description

Stops the timer utility.

timerUtilUpdate

Updates the timer state in the TimerUtilData data structure.

Definition

```
#include <sample_utilities/timer_utility.h>
int32_t timerUtilUpdate(
    TimerUtilData *pTimer
);
```

Arguments

[in] *pTimer*	A pointer to an instance of TimerUtilData containing time variables.

Return Values

Value	Description
SCE_OK	The operation was successful.
SCE_ERROR_ERRNO_EFAULT	The operation failed because the value of *pTimer* was NULL.

Description

Updates the timer state in the TimerUtilData data structure.

Heap Utility API

Data Types

HeapUtilContext

The data structure containing a list of allocated and free memory blocks in a heap.

Definition

```
#include <heap_utility.h>
typedef struct HeapUtilContext {
   HeapUtilMemBlock *allocList;
   HeapUtilMemBlock *freeList;
} HeapUtilContext;
```

Members

allocList	A list of allocated memory blocks. This is not sorted and functions on a LIFO basis.
freeList	A list of free memory blocks. This is sorted by base address and is always fully merged.

Description

The data structure containing a list of allocated and free memory blocks in a heap. This structure is used during the initialization of a heap's context.

HeapUtilMemBlock

The data structure containing details of a memory block in a heap.

Definition

```
#include <heap_utility.h>
typedef struct HeapUtilMemBlock {
   HeapUtilMemBlock *next;
   int32_t type;
   uintptr_t base;
   uint32_t offset;
   uint32_t size;
} HeapUtilMemBlock;
```

Members

next	A pointer to the next memory block in the heap.
type	An arbitrary value denoting the type of memory. This is used later with `heapUtilAlloc()` to allocate memory of this type.
base	The base address of the block.
offset	A USSE offset to track with this block. This can be set to zero if USSE offsets are not used.
size	The size of the block.

Description

The data structure containing details of a memory block in a heap. This structure is used for allocating a memory block in a heap.

Functions

heapUtilAlloc

Allocates memory from the heap.

Definition

```
#include <sample_utilities/heap_utility.h>
void *heapUtilAlloc(
    HeapUtilContext *pCtx,
    int32_t type,
    uint32_t size,
    uint32_t alignment
);
```

Arguments

[in] *pCtx*	A pointer to the heap context in use.
[in] *type*	The block type to allocate from. This should match a block type added earlier with `heapUtilExtend()`.
[in] *size*	The size in bytes of the allocation.
[in] *alignment*	The alignment in bytes of the start of the allocation.

Return Values

The address of the memory allocated. `NULL` is returned if no allocation could be made.

Description

Allocates memory from the heap.

heapUtilAllocWithOffset

Allocates memory from the heap with a USSE offset.

Definition

```
#include <sample_utilities/heap_utility.h>
void *heapUtilAllocWithOffset(
    HeapUtilContext *pCtx,
    int32_t type,
    uint32_t size,
    uint32_t alignment,
    uint32_t *offset
);
```

Arguments

[in] *pCtx*	A pointer to the heap context in use.
[in] *type*	The block type to allocate from. This should match a block type added earlier with `heapUtilExtend()`.
[in] *size*	The size in bytes of the allocation.
[in] *alignment*	The alignment in bytes of the start of the allocation.
[in] *offset*	A USSE offset to track with this block. This can be set to zero if USSE offsets are not used.

Return Values

The address of the memory allocated. `NULL` is returned if no allocation could be made.

Description

Allocates memory from the heap with a USSE offset.

heapUtilExtend

Adds a block of memory to the heap.

Definition

```
#include <sample_utilities/heap_utility.h>
int32_t heapUtilExtend(
    HeapUtilContext *pCtx,
    int32_t type,
    void *base,
    uint32_t size,
    uint32_t offset
);
```

Arguments

[in] *pCtx*	A pointer to the heap context in use.
[in] *type*	An arbitrary value denoting the type of memory contained in the block. Use this later with `heapUtilAlloc()` and `heapUtilAllocWithOffset()` to allocate memory of this type.
[in] *base*	The base address of the block.
[in] *size*	The size of the block.
[in] *offset*	A USSE offset to track with this block. This can be set to zero if USSE offsets are not used.

Return Values

Value	Description
`SCE_OK`	The operation was successful.
`SCE_ERROR_ERRNO_EFAULT`	The operation failed to add the block of memory to the heap.

Description

Adds a block of memory to the heap. The block must not overlap with any existing block.

heapUtilFree

Frees memory back to the heap.

Definition

```
#include <sample_utilities/heap_utility.h>
int32_t heapUtilFree(
    HeapUtilContext *pCtx,
    void *addr
);
```

Arguments

[in] *pCtx*	A pointer to the heap context in use.
[in] *addr*	The address of the allocation to free or `NULL`. If an address is supplied, it should match a previous return value from `heapUtilAlloc()` or `heapUtilAllocWithOffset()`.

Return Values

Value	Description
`SCE_OK`	The operation was successful.
`SCE_ERROR_ERRNO_EFAULT`	The operation failed to free the memory back into the heap.

Description

Frees memory back to the heap.

heapUtilInitialize

Initializes an empty heap.

Definition

```
#include <sample_utilities/heap_utility.h>
int32_t heapUtilInitialize(
    HeapUtilContext *pCtx
);
```

Arguments

[in] *pCtx*	A pointer to the heap context in use.

Return Values

Value	Description
SCE_OK	The operation was successful.
SCE_ERROR_ERRNO_EFAULT	The operation failed to initialize the heap utility.

Description

Initializes an empty heap. Use heapUtilExtend() to add free blocks to the heap before allocating memory using heapUtilAlloc() and heapUtilAllocWithOffset().

heapUtilTerminate

Destroys a heap.

Definition

```
#include <sample_utilities/heap_utility.h>
int32_t heapUtilTerminate(
    HeapUtilContext *pCtx
);
```

Arguments

[in] *pCtx*	A pointer to the heap context in use.

Return Values

Value	Description
`SCE_OK`	The operation was successful.
`SCE_ERROR_ERRNO_EFAULT`	The operation failed to terminate the heap utility.

Description

Destroys a heap.

Font Utility API

Data Types

FontUtilFontProperty

The data structure that stores the font information.

Definition

```
#include <sample_utilities/font_utility.h>
typedef struct FontUtilFontProperty{
   void *userData;
   FontUtilAllocFunc allocFunc;
   FontUtilReallocFunc reallocFunc;
   FontUtilFreeFunc freeFunc;
   float fontSize;
   uint32_t fontColor;
   void *libId;
   void *fontId0;
   void *fontId1;
   ScePvf_t_fontInfoSceFont_t_fontInfo fontInfo0;
   ScePvf_t_fontInfoSceFont_t_fontInfo fontInfo1;
   int32_t maxAscender;
   int maxGlyphBitmapHeight;

#ifdef __USE_BUILT_IN_KOREAN_FONT__
   SceFont_t_u16 languageCode;
#endif

   SceCesUcsContext cesContext;
} FontUtilFontProperty;
```

Members

userData	A pointer to the user data that is used by libpvf internally.
allocFunc	The memory allocation function that is used by libpvf internally.
reallocFunc	The memory re-allocation function that is used by libpvf internally.
freeFunc	The memory free function that is used by libpvf internally.
fontSize	The font size.
fontColor	The font color.
libId	The font library ID that is used by libpvf internally.
fontId0	The font ID of Font 0 that is used by libpvf internally.
fontId1	The font ID of Font 1 that is used by libpvf internally.
fontInfo0	The font information for Font 0 that is used by libpvf internally.
fontInfo1	The font information for Font 1 that is used by libpvf internally.
maxAscender	The maximum ascender value contained in the font data.
maxGlyphBitmapHeight	The maximum bitmap height value contained in the font data.
languageCode	The font language code.
cesContext	The context structure that is used by libces internally.

Description

This data structure that stores the font information. It must be initialized using `fontUtilInit()` before use.

FontUtilImageBuffer

The data structure that stores an image to be rendered.

Definition

```
#include <sample_utilities/font_utility.h>
typedef struct FontUtilImageBuffer{
    void *data;
    uint32_t width;
    uint32_t height;
    uint32_t byteStride;
    FontUtilImageBufferType imageBufferType;
    FontUtilImageState state;
} FontUtilImageBuffer;
```

Members

data	A pointer to the image storage destination.
width	The image width.
height	The image height.
byteStride	The image stride in bytes.
imageBufferType	The type of the image buffer.
state	The `FontUtilImageState` structure that is used internally.

Description

This data structure that stores an image to be rendered.

FontUtilFontSize

An enumeration to represent the different font sizes.

Definition

```
#include <sample_utilities/font_utility.h>
typedef enum FontUtilFontSize {
   FONTUTIL_FONTSIZE_INVALID = 0,
   FONTUTIL_FONTSIZE_X1,
   FONTUTIL_FONTSIZE_X2,
} FontUtilFontSize;
```

Enumeration Values

Macro	Description
`FONTUTIL_FONTSIZE_INVALID`	An invalid font size.
`FONTUTIL_FONTSIZE_X1`	The standard font size obtained from libpvf.
`FONTUTIL_FONTSIZE_X2`	A font size that is twice the size of that obtained from libgpf.

Description

An enumeration to represent the different font sizes.

FontUtilImageBufferType

An enumeration to represent the type of the image buffer.

Definition

```
#include <sample_utilities/font_utility.h>
typedef enum FontUtilImageBufferType {
   FONTUTIL_IMAGEBUFFER_TYPE_INVALID = 0,
   FONTUTIL_IMAGEBUFFER_TYPE_GRAY,
   FONTUTIL_IMAGEBUFFER_TYPE_COLOR_ABGR,
} FontUtilImageBufferType;
```

Enumeration Values

Macro	Description
`FONTUTIL_IMAGEBUFFER_TYPE_INVALID`	An invalid buffer type.
`FONTUTIL_IMAGEBUFFER_TYPE_GRAY`	A grey buffer type. A buffer of this type ignores color values set using `fontUtilSetFontColor()`.
`FONTUTIL_IMAGEBUFFER_TYPE_COLOR_ABGR`	An ABGR color buffer type.

Description

An enumeration to represent the type of the image buffer.

FontUtilAllocFunc

The typedef of the memory allocation function pointer used in libpvf.

Definition

```
#include <sample_utilities/font_utility.h>
typedef void* (*FontUtilAllocFunc)(
    void *userData,
    uint32_t size);
```

Members

userData	The user data.
size	The size of the memory to allocate.

Return Values

A pointer to the allocated memory.

Description

The typedef of the memory allocation function pointer used by `scePvfNewLib()` in libpvf.

Please refer to the *libpvf Reference* document for more details.

FontUtilReallocFunc

The typedef of the memory reallocation function pointer used in libpvf.

Definition

```
#include <sample_utilities/font_utility.h>
typedef void* (*FontUtilReallocFunc)(
   void *userData,
   void *addr,
   uint32_t size);
```

Members

userData	The user data.
addr	The address of the memory to reallocate.
size	The new size of the memory.

Return Values

A pointer to the reallocated memory.

Description

The typedef of the memory reallocation function pointer used by `scePvfNewLib()` in libpvf.

Please refer to the *libpvf Reference* document for more details.

FontUtilFreeFunc

The typedef of the memory deallocation function pointer used in libpvf.

Definition

```
#include <sample_utilities/font_utility.h>
typedef void (*FontUtilFreeFunc)(
    void *userData,
    void *addr);
```

Members

userData	The user data.
addr	The address of the memory to free.

Description

The typedef of the memory deallocation function pointer used by `scePvfNewLib()` in libpvf.

Please refer to the *libpvf Reference* document for more details.

Functions

fontUtilInit

Initializes the font utility.

Definition

```
#include <sample_utilities/font_utility.h>
int  fontUtilInit (
    FontUtilFontProperty *fontProperty,
    char *fileName,
    uint16_t languageCode,
    uint32_t accessMode,
    void *userData,
    FontUtilAllocFunc allocFunc,
    FontUtilReallocFunc reallocFunc,
    FontUtilFreeFunc freeFunc
);
```

Arguments

[in] *fontProperty*	A pointer to the `FontUtilFontProperty` structure to initialize.
[in] *fileName*	The font file name (built-in fonts are used when this is not specified).
[in] *languageCode*	Specifies the `SceFontLanguageCode` (refer to the *libpvf Reference* document).
[in] *accessMode*	Specifies the `SceFontDataAccessMode` (refer to the *libpvf Reference* document).
[in] *userData*	A pointer to user data.
[in] *allocFunc*	A pointer to the memory allocation function that is used by libpvf (refer to the *libpvf Reference* document).
[in] *reallocFunc*	A pointer to the memory reallocation function that is used by libpvf (refer to the *libpvf Reference* document).
[in] *freeFunc*	A pointer to memory free function that is used by libpvf (refer to the *libpvf Reference* document).

Return Values

Value	Result
SCE_OK	The operation was successful.
FONTUTIL_ERROR_INVALID_ARG	There was an error in one of the arguments.
FONTUTIL_ERROR_ALREADY_INITIALIZED	The operation has already been executed.
FONTUTIL_ERROR_INITIALIZE	The operation failed to initialize the font utility.

Description

Initializes the font utility.

fontUtilShutdown

Shuts down the font utility.

Definition

```
#include <sample_utilities/font_utility.h>
int  fontUtilShutdown (
    FontUtilFontProperty *fontProperty
);
```

Arguments

[in] *fontProperty*	A pointer to a `FontUtilFontProperty` structure that was initialized by `fontUtilInit()`.

Return Values

Value	Result
`SCE_OK`	The operation was successful.
`FONTUTIL_ERROR_INVALID_ARG`	There was an error in one of the arguments.
Others	An error code of libpvf.

Description

Shuts down the font utility. The function also frees the resources that are allocated by libpvf.

fontUtilSetFontSize

Sets the font size to use for string rendering.

Definition

```
#include <sample_utilities/font_utility.h>
int  fontUtilSetFontSize (
    FontUtilFontProperty *fontProperty,
    float fontSize
);
```

Arguments

[in] *fontProperty*	A pointer to the `FontUtilFontProperty` structure to use.
[in] *fontSize*	The font size. Please refer to `scePvfSetCharSize()` in the *libpvf Reference* document.

Return Values

Value	Result
`SCE_OK`	The operation was successful.
`FONTUTIL_ERROR_INVALID_ARG`	There was an error in one of the arguments.
Others	An error code of libpvf.

Description

Sets the font size to use for string rendering. This value is parsed to `scePvfSetCharSize()`. For more information please refer to the *libpvf Reference* document.

fontUtilGetFontSize

Obtains the font size that is currently set.

Definition

```
#include <sample_utilities/font_utility.h>
int fontUtilGetFontSize (
    FontUtilFontProperty *fontProperty,
    float *fontSize
);
```

Arguments

[in] *fontProperty*	A pointer to the FontUtilFontProperty structure to use.
[out] *fontSize*	Receives the font size.

Return Values

Value	Description
SCE_OK	The operation was successful.
FONTUTIL_ERROR_INVALID_ARG	There was an error in one of the arguments.

Description

Obtains the font size that is currently set.

fontUtilSetFontColor

Sets the font color used for string rendering.

Definition

```
#include <sample_utilities/font_utility.h>
int  fontUtilSetFontColor (
    FontUtilFontProperty *fontProperty,
    uint32_t color
);
```

Arguments

[in] *fontProperty*	A pointer to the FontUtilFontProperty structure to use.
[in] *color*	The font color in ARGB format.

Return Values

Value	Description
SCE_OK	The operation was successful.
FONTUTIL_ERROR_INVALID_ARG	There was an error in one of the arguments.

Description

Sets the font color used for string rendering.

fontUtilGetFontColor

Obtains the font color set.

Definition

```
#include <sample_utilities/font_utility.h>
int  fontUtilGetFontColor (
    FontUtilFontProperty *fontProperty,
    uint32_t *color
);
```

Arguments

[in] *fontProperty*	A pointer to the FontUtilFontProperty structure to use.
[out] *color*	Receives the font color in the ARGB format set.

Return Values

Value	Description
SCE_OK	The operation was successful.
FONTUTIL_ERROR_INVALID_ARG	There was an error in one of the arguments.

Description

Obtains the font color set.

fontUtilCreateImageBuffer

Creates/initializes an image buffer.

Definition

```
#include <sample_utilities/font_utility.h>
int  fontUtilCreateImageBuffer (
    FontUtilImageBuffer *imageBuffer,
    void *data,
    uint32_t width,
    uint32_t height,
    uint32_t byteStride,
    FontUtilImageBufferType imageBufferType
);
```

Arguments

[in] *imageBuffer*	A pointer to a FontUtilImageBuffer structure.
[out] *data*	Receives the created image.
[in] *width*	The width of the image to be generated and initialized.
[in] *height*	The height of the image to be generated and initialized.
[in] *byteStride*	The stride in bytes of the image to be generated and initialized.
[in] *imageBufferType*	The image type.

Return Values

Value	Description
SCE_OK	The operation was successful.
FONTUTIL_ERROR_NO_IMAGEBUFFER	The FontUtilImageBuffer structure was not created/initialized successfully.
FONTUTIL_ERROR_ALLOC_WORKMEM	The operation failed to allocate the working memory for the image buffer.

Description

Creates/initializes an image buffer.

FONTUTIL_IMAGEBUFFER_TYPE_GRAY or FONTUTIL_IMAGEBUFFER_TYPE_COLOR_ABGR can be specified for the *imageBufferType*. When FONTUTIL_IMAGEBUFFER_TYPE_GRAY is specified to *imageBufferType*, the ABGR value that is set by fontUtilSetFontColor() will be ignored. When FONTUTIL_IMAGEBUFFER_TYPE_COLOR_ABGR is specified to *imageBufferType*, a font image is created based on the ABGR value set by fontUtilSetFontColor().

When the value of the character glyph image obtained from libpvf is anything other than 0x00, the value set by fontUtilSetFontColor() will be reflected to the BGR value as is. When the character glyph image value is 0x00, then 0x00 is set to the BGR value. If you wish to set the value of the character glyph image to A, set A to 0xFF using fontUtilSetFontColor().

fontUtilFreeImageBuffer

Frees an image buffer.

Definition

```
#include <sample_utilities/font_utility.h>
int  fontUtilFreeImageBuffer (
    FontUtilImageBuffer *imageBuffer,
);
```

Arguments

[in] *imageBuffer*	A pointer to a FontUtilImageBuffer structure.

Return Values

Value	Description
SCE_OK	The operation was successful.
FONTUTIL_ERROR_NO_IMAGEBUFFER	The pointer to the FontUtilImageBuffer structure was NULL or the structure had not been initialized.

Description

Frees an image buffer.

fontUtilClearImageBuffer

Clears an image buffer.

Definition

```
#include <sample_utilities/font_utility.h>
int  fontUtilClearImageBuffer (
    FontUtilImageBuffer *imageBuffer,
);
```

Arguments

[in] *imageBuffer*	A pointer to the `FontUtilImageBuffer` structure to be cleared.

Return Values

Value	Description
`SCE_OK`	The operation was successful.
`FONTUTIL_ERROR_NO_IMAGEBUFFER`	The pointer to the `FontUtilImageBuffer` structure was `NULL` or the structure had not been initialized.

Description

Clears an image buffer. All of the image buffer memory will be set to 0, but it will remain allocated. The dimensions of the buffer will remain the same.

fontUtilPrintUtf8

Renders a UTF8 string to the specified image buffer.

Definition

```
#include <sample_utilities/font_utility.h>
int  fontUtilPrintUtf8(
    FontUtilFontProperty *fontProperty,
    const uint8_t *utf8str,
    const uint32_t  maxStringsLen,
    int x,
    int y
);
```

Arguments

[in] *fontProperty*	A pointer to the FontUtilFontProperty structure to use.
[out] *utf8str*	Receives the rendered UTF8 string.
[in] *maxStringsLen*	The maximum length of the string to be rendered.
[in] *x*	The rendering start point X.
[in] *y*	The rendering start point Y.

Return Values

Values	Description
SCE_OK	The operation was successful.
FONTUTIL_ERROR_INVALID_ARG	There was an error in one of the arguments.
Others	An error code of libpvf.

Description

Renders a UTF8 string to the specified image buffer.

fontUtilPrintUcs2

Renders a UCS2 string to the specified image buffer.

Definition

```
#include <sample_utilities/font_utility.h>
int  fontUtilPrintUcs2(
    FontUtilFontProperty *fontProperty,
    const uint16_t *ucs2str,
    const uint32_t  maxStringsLen,
    int x,
    int y
);
```

Arguments

[in] *fontProperty*	A pointer to the FontUtilFontProperty structure to use.
[out] *ucs2str*	Receives the rendered UCS2 string.
[in] *maxStringsLen*	The maximum length of the string to be rendered.
[in] *x*	The rendering start point X.
[in] *y*	The rendering start point Y.

Return Values

Values	Description
SCE_OK	The operation was successful.
FONTUTIL_ERROR_INVALID_ARG	There was an error in one of the arguments.
Others	An error code of libpvf.

Description

Renders a UCS2 string to the specified image buffer.

fontUtilGetPrintSizeUtf8

Obtains the frame size of a string to be rendered by fontUtilPrintUtf8().

Definition

```
#include <sample_utilities/font_utility.h>
int  fontUtilGetPrintSizeUtf8 (
    FontUtilFontProperty *fontProperty,
    FontUtilImageBuffer *imageBuffer,
    const uint8_t *utf8str,
    const uint32_t  maxStringsLen,
    uint32_t *width,
    uint32_t *height
);
```

Arguments

[in] *fontProperty*	A pointer to the FontUtilFontProperty structure to use.
[in] *imageBuffer*	A pointer to FontUtilImageBuffer structure to be rendered.
[in] *utf8str*	A pointer to the storage destination to which the UTF8 string will be rendered.
[in] *maxStringsLen*	The maximum length of the string to be rendered.
[out] *width*	Receives the width of the frame to be rendered.
[out] *height*	Receives the height of the frame to be rendered.

Return Values

Value	Description
SCE_OK	The operation was successful.
FONTUTIL_ERROR_INVALID_ARG	There was an error in one of the arguments.
FONTUTIL_ERROR_NO_IMAGEBUFFER	The pointer to the FontUtilImageBuffer structure was NULL or the structure had not been initialized.
Others	An error code of libpvf.

Description

Obtains the frame size of a string to be rendered by fontUtilPrintUtf8().

This function only performs computation of the frame size and does not perform rendering.

fontUtilGetPrintSizeUcs2

Obtains the frame size of a string to be rendered by fontUtilPrintUcs2().

Definition

```
#include <sample_utilities/font_utility.h>
int  fontUtilGetPrintSizeUcs2 (
    FontUtilFontProperty *fontProperty,
    FontUtilImageBuffer *imageBuffer,
    const uint16_t *ucs2str,
    const uint32_t  maxStringsLen,
    uint32_t *width,
    uint32_t *height
);
```

Arguments

[in] *fontProperty*	A pointer to the FontUtilFontProperty structure to use.
[in] *imageBuffer*	A pointer to FontUtilImageBuffer structure to be rendered.
[in] *ucs2str*	A pointer to the storage destination to which the UCS2 string will be rendered.
[in] *maxStringsLen*	The maximum length of the string to be rendered.
[out] *width*	Receives the width of the frame to be rendered.
[out] *height*	Receives the height of the frame to be rendered.

Return Values

Value	Description
`SCE_OK`	The operation was successful.
`FONTUTIL_ERROR_INVALID_ARG`	There was an error in one of the arguments.
`FONTUTIL_ERROR_NO_IMAGEBUFFER`	The pointer to the FontUtilImageBuffer structure was `NULL` or the structure had not been initialized.
Others	An error code of libpvf.

Description

Obtains the frame size of a string to be rendered by fontUtilPrintUcs2().

This function only performs computation of the frame size and does not perform rendering.

Sound Utility API

Data Types

SoundUtilResources

Defines the resource to be used within the sound utility.

Definition

```
#include <sample_utilities/sound_utility.h>
typedef struct SoundUtilResources {
   ...
}  SoundUtilResources;
```

Members

Do not directly access the members of this structure from the application.

Description

Defines the resource to be used within the sound utility.

Do not directly access the members of this structure from the application.

SoundUtilVoiceInitParams

Defines the number of voices that can be used within the sound utility.

Definition

```
#include <sample_utilities/sound_utility.h>
typedef struct SoundUtilVoiceInitParams {
   SceUInt32 stereoVoiceNum;
   SceUInt32 monoVoiceNum;
} SoundUtilVoiceInitParams;
```

Members

stereoVoiceNum	The maximum number of stereo voices that can be used concurrently.
monoVoiceNum	The maximum number of monaural voices that can be used concurrently.

Description

Defines the number of voices that can be used within the sound utility. These parameters are used when initializing the sound utility.

SoundUtilBussInitParams

Defines the number of busses that can be used within the sound utility.

Definition

```
#include <sample_utilities/sound_utility.h>
typedef struct SoundUtilBussInitParams {
   SceUInt32 reverbBussNum;
   SceUInt32 mixerBussNum;
} SoundUtilBussInitParams;
```

Members

reverbBussNum	The number of reverb busses that can be used concurrently.
mixerBussNum	The number of mixer busses that can be used concurrently.

Description

Defines the number of busses that can be used within the sound utility. These parameters are used when initializing the sound utility.

SoundUtilUpdateThreadInfo

Defines the update thread of the sound utility.

Definition

```
#include <sample_utilities/sound_utility.h>
typedef struct SoundUtilUpdateThreadInfo {
    SceUInt32 priority;
    SceSize stackSize;
    SoundUtilRenderHandler userFunction;
    void *userAttr;
    SceInt32 cpuAffinityMask;
} SoundUtilUpdateThreadInfo;
```

Members

priority	The priority of the update thread.
stackSize	The stack size of the update thread.
userFunction	The user defined function called within the update thread.
**userAttr*	The argument for the function defined in *userFunction*.
cpuAffinityMask	The affinity mask of the update thread.

Description

Defines the update thread of the sound utility. The update thread is generated within the sound utility based on this information.

SoundUtilSoundInfo

Contains sound data information used for memory playback.

Definition

```
#include <sample_utilities/sound_utility.h>
typedef struct SoundUtilSoundInfo {
   void *data;
   SceUInt32 numBytes;
   SceUInt32 numChannels;
   SceUInt32 sampleRate;
   SceUInt32 type;
   SceUInt32 isLoop;
   SceUInt32 loopStart;
   SceUInt32 loopEnd;
   SceUInt32 loopNum;
} SoundUtilSoundInfo;
```

Members

data	The start address of the sound data.
numBytes	The size of the sound data.
numChannels	The number of channels used for the sound data.
sampleRate	The sampling rate of the sound data.
type	The sound data type.
isLoop	Specifies whether the sound data should be played as a loop.
loopStart	The offset to the start of the loop in the sound data.
loopEnd	The offset to the end of the loop in the sound data.
loopNum	The number of times to play the loop.

Description

Contains sound data information used for memory playback.

SoundUtilWavInfo

Contains wav file information.

Definition

```
#include <sample_utilities/sound_utility.h>
typedef struct SoundUtilWavInfo {
   SceInt32SceUInt32 dataSize;
   SceInt32SceUInt32 dataOffset;
   SceInt32SceUInt32 numChannels;
   SceInt32SceUInt32 sampleRate;
   SceInt32SceUInt32 loopStart;
   SceInt32SceUInt32 loopEnd;
   SoundUtilAt9BufferInfo at9Info;
} SoundUtilWavInfo;
```

Members

dataSize	The size of the sound data.
dataOffset	The offset to the sound data.
numChannels	The number of channels used by the sound data.
sampleRate	The sampling rate of the sound data.
loopStart	The offset to the start of the loop in the sound data.
loopEnd	The offset to the end of the loop in the sound data.
at9Info	The format information of ATRAC9™.

Description

Contains wav file information.

SoundUtilVolumeInfo

This structure is used to set volume.

Definition

```
#include <sample_utilities/sound_utility.h>
typedef struct SoundUtilVolumeInfo {
   float leftVol;
   float rightVol;
} SoundUtilVolumeInfo;
```

Members

leftVol	The volume of the left channel.
rightVol	The volume of the right channel.

Description

This structure is used to set volume. When 1.0f or 0.5f is specified, the volume is adjusted to the same volume as the original volume or half the original volume respectively.

SoundUtilEnvelopeInfo

This structure is used to set an envelope.

Definition

```
#include <sample_utilities/sound_utility.h>
typedef struct SoundUtilEnvelopeInfo {
   SceNgsEnvelopePoint envelopePoints[SCE_NGS_ENVELOPE_MAX_POINTS];
   SceUInt32 releaseMsecs;
   SceUInt32 numPoints;
   SceUInt32 loopStart;
   SceInt32 loopEnd;
} SoundUtilEnvelopeInfo;
```

Members

envelopePoints	The points of the envelope.
releaseMsecs	The release rate (msec) of the envelope.
numPoints	The number of points in the envelope (1-4).
loopStart	The loop start number of the envelope (0-2).
loopEnd	The loop end number of the envelope (1-3, must be smaller than *loopStart*).

Description

This structure is used to set an envelope. Please refer to Chapter 6, “Amplitude Envelope DSP Effect Module Overview” in the *NGS Modules Overview*.

SoundUtilNoiseInfo

Represents noise data.

Definition

```
#include <sample_utilities/sound_utility.h>
typedef struct SoundUtilNoiseInfo {
   SceInt32 frequency;
   SceFloat32 amplitude;
   SceFloat32 pulseWidth;
   SceUInt32 sampleOffset;
   SceUInt32 phaseAngle;
   SceUInt32 type;
} SoundUtilNoiseInfo;
```

Members

frequency	The waveform playback frequency (in Hz).
amplitude	The waveform amplitude.
pulseWidth	The pulse width.
sampleOffset	The generation start offset in the waveform.
phaseAngle	The starting phase offset to use during sine wave generation.
type	The type of the waveform.

Description

Represents noise data. Each member is passed to a member of `SceNgsGeneratorSettings` in the Sound Utility as follows:

frequency -> *nFrequency*

amplitude -> *fAmplitude*

pulseWidth -> *fPulseWidth*

sampleOffset -> *uSampleOffset*

phaseAngle -> *uPhaseAngle*

type -> *eGeneratorMode*

Please see the *NGS Modules Overview* document for more details on noise behavior.

SoundUtilStreamingInfo

Represents streaming file data.

Definition

```
#include <sample_utilities/sound_utility.h>
typedef struct SoundUtilStreamingInfo {
    void* workBuffer;
    SceUInt32 workBufferSize;
    char* filePath;
    SceInt32 numBytes;
    SceInt32 numChannels;
    SceInt32 sampleRate;
    SceInt32 type;
    SceInt32 isLoop;
    SceInt32 loopStart;
    SceInt32 loopEnd;
    SceInt32 loopNum;
    SoundUtilAt9BufferInfo at9Info;
} SoundUtilStreamingInfo
```

Members

workBuffer	The work-buffer for the streaming buffer (not used in the current implementation).
workBufferSize	The size of the work-buffer for the streaming buffer (not used in the current implementation).
filePath	The file path to the streaming file.
numBytes	The size of the streaming file data.
numChannels	The number of channels within the streaming file.
sampleRate	The sampling frequency of the streaming file.
type	The data type of the streaming file. SOUND_UTIL_SOUND_TYPE_AT9 and SOUND_UTIL_SOUND_TYPE_WAV can be specified. SOUND_UTIL_SOUND_TYPE_AT9 is the preferred format because of file read load.
isLoop	Specifies whether to use loops within the streaming file: SOUND_UTIL_LOOP_NONE: The file should not be looped. SOUND_UTIL_LOOP_POINT: If both *loopStart* and *loopEnd* are specified, the file will loop according to that setting; however, if either of those two values are set to 0, the file will loop according to the looping setting in the file. SOUND_UTIL_LOOP_ALL: The whole file should be looped.
loopStart	The sample that represents the start of the loop.
loopEnd	The sample that represents the end of the loop.
loopNum	The number of loops to make. No loops are made if 0 is specified.
at9Info	ATRAC9™ buffer information. This should be specified if the format of the streaming file is ATRAC9™.

Description

Represents streaming file data. A pointer to a SoundUtilStreamingInfo object is passed to the *dataInfo* argument of soundUtilVoiceOpen().

SoundUtilMalloc

The typedef of the memory allocation function pointer used in the Sound Utility.

Definition

```
#include <sample_utilities/sound_utility.h>
typedef void* SoundUtilMalloc (
    int boundary,
    int size);
```

Members

boundary	The top address alignment of the requested memory.
size	The size of the requested memory.

Description

The typedef of the memory allocation function pointer used in the Sound Utility. It is required when using `soundUtilSetMemoryFunc()` to replace the memory allocation function currently used in the Sound Utility.

SoundUtilFree

The typedef of the memory deallocation function pointer used in the Sound Utility.

Definition

```
#include <sample_utilities/sound_utility.h>
typedef void SoundUtilFree (
    void* src);
```

Members

src	The top address of the memory to be released.

Description

The typedef of the memory deallocation function pointer used in the Sound Utility. It is required when using `soundUtilSetMemoryFunc()` to replace the memory deallocation function currently used in the Sound Utility.

Functions

soundUtilSetMemoryFunc

Sets the `malloc()` and `free()` functions to use within the sound utility.

Definition

```
#include <sample_utilities/sound_utility.h>
void  soundUtilSetMemoryFunc (
   SoundUtilResources *resources,
   SOUND_UTIL_MALLOC *userMalloc,
   SOUND_UTIL_FREE *userFree
);
```

Argument

[in] *resources*	A pointer to a `SoundUtilResources` structure.
[in] *userMalloc*	A pointer to the `malloc()` function to use.
[in] *userFree*	A pointer to the `free()` function to use.

Description

Sets the `malloc()` and the `free()` functions to use within the sound utility.

soundUtilInit

Initializes the sound utility.

Definition

```
#include <sample_utilities/sound_utility.h>
int  soundUtilInit (
    SoundUtilResources *resources,
    SoundUtilVoiceInitParams *voiceParams,
    SoundUtilBussInitParams *bussParams,
    SoundUtilUpdateThreadInfo *threadInfo
);
```

Argument

[in] *resources*	A pointer to a SoundUtilResources structure.
[in] *voiceParams*	A pointer to a SoundUtilVoiceInitParams structure.
[in] *bussParams*	A pointer to a SoundUtilBussInitParams structure.
[in] *threadInfo*	A pointer to a SoundUtilUpdateThreadInfo structure.

Return Values

Value	Description
SCE_OK	The operation was successful.
SOUND_UTIL_ERROR_PARAM	There was an error in one of the arguments.
SOUND_UTIL_ERROR_FATAL	The operation failed to initialize the sound utility.
Others	An NGS error code.

Description

Initializes the sound utility.

soundUtilExit

Terminates the sound utility.

Definition

```
#include <sample_utilities/sound_utility.h>
int  soundUtilExit (
    SoundUtilResources *resources
);
```

Argument

[in] *resources*   A pointer to a SoundUtilResources structure.

Return Values

Value	Description
SCE_OK	The operation was successful.
SOUND_UTIL_ERROR_PARAM	There was an error in one of the arguments.
Others	An NGS error code.

Description

Terminates the sound utility.

soundUtilProcess

Updates the sound utility.

Definition

```
#include <sample_utilities/sound_utility.h>
void  soundUtilProcess (
    SoundUtilResources *resources,
    void *buffer
);
```

Argument

[in] *resources*	A pointer to a SoundUtilResources structure.
[in] *buffer*	The master buss output buffer.

Description

Updates the sound utility to reflect the playback and stop requests etc.

Although a function such as soundUtilVoicePlay() may have been called, the actual processing of the function is not executed until this function is called. Therefore this function must be called regularly at intervals of 5.3 msec by default.

If this function is called within the callback defined in the *userFunction* of the SoundUtilUpdateThreadInfo structure, it will be called at the intervals required by the system.

soundUtilGetWavInfo

Obtains wav file information.

Definition

```
#include <sample_utilities/sound_utility.h>
int  soundUtilGetWavInfo (
    SoundUtilResources *resources,
    const char *filePath,
    SoundUtilWavInfo *wavInfo
);
```

Argument

[in] *resources*	A pointer to a SoundUtilResources structure
[in] *filePath*	The path of the wave file.
[out] *wavInfo*	Receives the wav file information.

Return Values

Value	Description
SCE_OK	The operation was successful.
SOUND_UTIL_ERROR_PARAM	There was an error in one of the arguments.
SOUND_UTIL_ERROR_FATAL	The operation failed to obtain the wav file information.

Description

Obtains the wav file information for the file specified by *filePath*.

soundUtilLoadData

Loads sound data.

Definition

```
#include <sample_utilities/sound_utility.h>
int  soundUtilLoadData (
    SoundUtilResources *resources,
    const char *filePath,
    int *soundType,
    SoundUtilSoundInfo *soundInfo
);
```

Argument

resources	A pointer to a SoundUtilResources structure.
[in] *filePath*	The path to the wave file.
[in] *soundType*	The type of the sound.
[out] *soundInfo*	Receives the sound data.

Return Values

Value	Description
SCE_OK	The operation was successful.
SOUND_UTIL_ERROR_PARAM	There was an error in one of the arguments.
SOUND_UTIL_ERROR_FATAL	The operation failed to load the data.

Description

Loads the data from the file specified by *filePath* into *soundInfo*. Because this function allocates memory internally, soundUtilUnloadData() must be called when this data is no longer needed.

soundUtilUnloadData

Unloads sound data.

Definition

```
#include <sample_utilities/sound_utility.h>
int  soundUtilunloadData (
    SoundUtilResources *resources,
    SoundUtilSoundInfo *soundInfo
);
```

Argument

[in] *resources*	A pointer to a SoundUtilResources structure.
[in] *soundInfo*	The SoundUtilSoundInfo structure containing the data to unload.

Return Values

Value	Description
SCE_OK	The operation was successful.
SOUND_UTIL_ERROR_PARAM	There was an error in one of the arguments.

Description

Unloads the sound data in a SoundUtilSoundInfo structure. Any memory allocated when the sound was loaded is freed.

soundUtilVoiceOpen

Opens a voice.

Definition

```
#include <sample_utilities/sound_utility.h>
int  soundUtilVoiceOpen (
    SoundUtilResources *resources,
    int dataType,
    void *dataInfo,
    const SoundUtilVolumeInfo *defaultVol
);
```

Argument

[in] *resources*	A pointer to a SoundUtilResources structure.
[in] *dataType*	The type of the data (use either SOUND_UTIL_DATATYPE_SOUND, SOUND_UTIL_DATATYPE_NOISE or SOUND_UTIL_DATATYPE_STREAMING_SOUND).
[in] *dataInfo*	Depending on which datatype was specified, this can either be a SoundUtilSoundInfo, SoundUtilNoiseInfo or SoundUtilStreamingInfo structure.
[in] *defaultVol*	Specifies the default volume to play the voice at.

Return Values

Value	Description
Voice ID	The ID of the successfully created voice (0 or greater).
SOUND_UTIL_ERROR_PARAM	There was an error in one of the arguments.
SOUND_UTIL_ERROR_FATAL	The operation failed to open the voice.
Others	An NGS error code.

Description

Opens a voice. On successful completion of the operation, the voice ID can be used for voice related functions.

soundUtilVoicePlay

Plays a voice.

Definition

```
#include <sample_utilities/sound_utility.h>
int  soundUtilVoicePlay (
    SoundUtilResources *resources,
    const int voiceID,
    int autoClose
);
```

Argument

[in] *resources*	A pointer to a SoundUtilResources structure.
[in] *voiceID*	The ID of the voice to play.
[in] *autoClose*	Specifies whether to close the voice after playback has ended. A value of 1 means the voice will be closed after playback. A value of 0 means it will not.

Return Values

Value	Description
SCE_OK	The operation was successful.
SOUND_UTIL_ERROR_PARAM	There was an error in one of the arguments.
Others	An NGS error code.

Description

Plays a voice.

soundUtilVoiceKeyOff

Performs a key-off of the specified voice.

Definition

```
#include <sample_utilities/sound_utility.h>
int  soundUtilVoiceKeyOff (
    SoundUtilResources *resources,
    const int voiceID
);
```

Argument

[in] *resources*	A pointer to a SoundUtilResources structure.
[in] *voiceID*	The ID of the voice to key-off.

Return Values

Value	Description
SCE_OK	The operation was successful.
SOUND_UTIL_ERROR_PARAM	There was an error in one of the arguments.
Others	An NGS error code.

Description

Performs a key-off of the specified voice.

soundUtilVoiceClose

Closes a voice.

Definition

```
#include <sample_utilities/sound_utility.h>
int  soundUtilVoiceClose (
    SoundUtilResources *resources,
    const int voiceID
);
```

Argument

[in] *resources*	A pointer to a SoundUtilResources structure.
[in] *voiceID*	The voice ID.

Return Values

Value	Description
SCE_OK	The operation was successful.
SOUND_UTIL_ERROR_PARAM	There was an error in one of the arguments.

Description

Closes a voice. When this function is called for a voice that is still being played, the playback of the voice will be forcibly closed.

soundUtilVoicePause

Pauses voice playback.

Definition

```
#include <sample_utilities/sound_utility.h>
int  soundUtilVoicePause (
    SoundUtilResources *resources,
    const int voiceID
);
```

Argument

[in] *resources*	A pointer to a SoundUtilResources structure.
[in] *voiceID*	The voice ID.

Return Values

Value	Description
SCE_OK	The operation was successful.
SOUND_UTIL_ERROR_PARAM	There was an error in one of the arguments.
Others	An NGS error code.

Description

Pauses the playback of a voice.

soundUtilVoiceResume

Resumes voice playback.

Definition

```
#include <sample_utilities/sound_utility.h>
int  soundUtilVoiceResume (
    SoundUtilResources *resources,
    const int voiceID
);
```

Argument

[in] *resources*	A pointer to a SoundUtilResources structure.
[in] *voiceID*	The voice ID.

Return Values

Value	Description
SCE_OK	The operation was successful.
SOUND_UTIL_ERROR_PARAM	There was an error in one of the arguments.
Others	An NGS error code.

Description

Resumes voice playback after it has been paused.

soundUtilVoiceSetEnvelope

Sets the voice envelope information.

Definition

```
#include <sample_utilities/sound_utility.h>
int  soundUtilVoiceSetEnvelope (
    SoundUtilResources *resources,
    const int voiceID,
    SoundUtilEnvelopeInfo *envelopeInfo
);
```

Argument

[in] *resources*	A pointer to a SoundUtilResources structure.
[in] *voiceID*	The voice ID.
[in] *envelopeInfo*	The envelope information.

Return Values

Value	Description
SCE_OK	The operation was successful.
SOUND_UTIL_ERROR_PARAM	There was an error in one of the arguments.
Others	An NGS error code.

Description

Sets the voice envelope information.

soundUtilVoiceGetState

Gets the state of a voice.

Definition

```
#include <sample_utilities/sound_utility.h>
int  soundUtilVoiceGetState (
    SoundUtilResources *resources,
    const int voiceID,
);
```

Argument

[in] *resources*	A pointer to a SoundUtilResources structure.
[in] *voiceID*	The voice ID.

Return Values

Value	Description
0 or greater	The state of the voice. The operation was successful.
Others	An NGS error code.

Description

Gets the state of a voice. The state of the voice is assigned to each bit of the return value. SOUND_UTIL_STATE_AVAILABLE, SOUND_UTIL_STATE_ACTIVE, SOUND_UTIL_STATE_PAUSE, SOUND_UTIL_STATE_KEYOFF and SOUND_UTIL_STATE_PLAYING are declared as bits in sound_utility.h.

soundUtilVoiceGetModuleState

Gets the module information within a voice.

Definition

```
#include <sample_utilities/sound_utility.h>
int  soundUtilVoiceGetModuleState (
    SoundUtilResources *resources,
    const int voiceID,
    int module,
    void *mem,
    int memSize
);
```

Argument

[in] *resources*	A pointer to a SoundUtilResources structure.
[in] *voiceID*	The voice ID.
[in] *module*	The module index.
[out] *mem*	Receives the address of the memory that will receive the module information.
[in] *memSize*	The size of the module information.

Return Values

Value	Description
SCE_OK	The operation was successful.
SOUND_UTIL_ERROR_PARAM	There was an error in one of the arguments.
Others	An NGS error code.

Description

Obtains information about a module (such as the player or the envelope module) contained within a voice.

soundUtilVoiceSetBuss

Connects a voice and a buss.

Definition

```
#include <sample_utilities/sound_utility.h>
int  soundUtilVoiceSetBuss (
    SoundUtilResources *resources,
    const int voiceID,
    const int sendNo,
    const int bussID,
    const SoundUtilVolumeInfo *volumeInfo
);
```

Argument

[in] *resources*	A pointer to a SoundUtilResources structure.
[in] *voiceID*	The voice ID.
[in] *sendNo*	The output destination of the voice.
[in] *bussID*	The buss ID.
[in] *volumeInfo*	The volume to set the voice to. If `NULL` is used, the volume is set to `1.0f`.

Return Values

Value	Description
`SCE_OK`	The operation was successful.
`SOUND_UTIL_ERROR_PARAM`	There was an error in one of the arguments.
`SOUND_UTIL_ERROR_FATAL`	An error occurred connecting the buss.
Others	An NGS error code.

Description

Connects the output destination of a voice to a buss.

soundUtilBussCreateReverb

Creates a reverb buss.

Definition

```
#include <sample_utilities/sound_utility.h>
int  soundUtilBussCreateReverb (
    SoundUtilResources *resources,
    int reverbMode
);
```

Argument

[in] *resources*	A pointer to a SoundUtilResources structure.
[in] *reverbMode*	The reverb mode.

Return Values

Value	Description
`Reverb Buss ID`	The ID of the successfully created reverb buss (0 or greater).
`SOUND_UTIL_ERROR_PARAM`	There was an error in one of the arguments.
`SOUND_UTIL_ERROR_BUSY`	The reverb voice was unavailable.
Others	An NGS error code.

Description

Creates a reverb buss. By specifying the returned reverb buss ID to `soundUtilVoiceSetBuss()`, a voice and the buss can be connected.

soundUtilBussDestroyReverb

Destroys a reverb buss.

Definition

```
#include <sample_utilities/sound_utility.h>
int  soundUtilBussDestroyReverb (
    SoundUtilResources *resources,
    const int reverbBussID
);
```

Argument

[in] *resources*	A pointer to a SoundUtilResources structure.
[in] *reverbBussID*	The reverb buss ID.

Return Values

Value	Description
SCE_OK	The operation was successful.
SOUND_UTIL_ERROR_PARAM	There was an error in one of the arguments.
Others	An NGS error code.

Description

Destroys a reverb buss.

soundUtilEnableBgmPort

Enables or disables the BGM port.

Definition

```
#include <sample_utilities/sound_utility.h>
int  soundUtilEnableBgmPort (
    SoundUtilResources *resources,
    int flag
);
```

Argument

[in] *resources*	A pointer to a SoundUtilResources structure.
[in] *flag*	Specify either `SOUND_UTIL_BGM_PORT_ENABLE` or `SOUND_UTIL_BGM_PORT_DISABLE` depending on whether you want enable or disable the port.

Return Values

Value	Description
`SCE_OK`	The operation was successful.
`SOUND_UTIL_ERROR_PARAM`	There was an error in one of the arguments.
Others	An NGS error code.

Description

Enables or disables the BGM port.

soundUtilGetBgmBuffer

Obtains the buffer for BGM port output.

Definition

```
#include <sample_utilities/sound_utility.h>
int  soundUtilGetBgmBuffer (
    SoundUtilResources *resources,
    void** buffer,
    int* size
);
```

Argument

[in] *resources*	A pointer to a SoundUtilResources structure.
[out] *buffer*	Receives a pointer to the buffer for BGM port output.
[out] *size*	Receives the size of the buffer for BGM port output.

Return Values

Value	Description
SCE_OK	The operation was successful.
SOUND_UTIL_ERROR_PARAM	There was an error in one of the arguments.
SOUND_UTIL_ERROR_NOT_ACTIVE	The BGM port is not active.
Others	An NGS error code.

Description

Obtains the buffer for BGM port output. Sound data can be output from the BGM port by calling `soundUtilSetBgmData()` after calling this function. Call `soundUtilSetBgmData()` soon after calling this function if the data has been not been modified.

soundUtilSetBgmData

Notifies the sound utility that data has been set for the buffer obtained with `soundUtilGetBgmBuffer()`.

Definition

```
#include <sample_utilities/sound_utility.h>
int  soundUtilSetBgmData (
    SoundUtilResources *resources
);
```

Argument

[in] *resources*	A pointer to a `SoundUtilResources` structure.

Return Values

Value	Description
SCE_OK	The operation was successful.
SOUND_UTIL_ERROR_PARAM	There was an error in one of the arguments.
SOUND_UTIL_ERROR_NOT_ACTIVE	The BGM port is not active.
Others	An NGS error code.

Description

Notifies the sound utility that data has been set for the buffer obtained with `soundUtilGetBgmBuffer()`. Sound data generated before the next call to `soundUtilProcess()` can be output from the BGM port by calling this function.

Callback Functions

SoundUtilRenderHandler

A callback function which is called when the audio output is updated.

Definition

```
#include <sample_utilities/sound_utility.h>
typedef void (*SoundUtilRenderHandler)(
    void* buffer,
    void* userAttr);
```

Members

buffer	The next sound buffer to output.
userAttr	The user argument. This comes from the *userAttr* member of the *threadInfo* argument passed to soundUtilInit().

Description

A callback function which is called when the audio output is updated. The linear PCM data to output next must be inserted into the buffer. 512 samples x 2 channels of data is required for the buffer.

Debug Menu Utility API

Data Types

DebugMenuUtilItemStatus

An enumeration to represent the different status types that can be signified by a bool type menu item.

Definition

```
#include <sample_utilities/menu_utility.h>
typedef enum DebugMenuUtilItemStatus {
   DEBUG_MENU_UTIL_ITEM_TRUE_FALSE,
   DEBUG_MENU_UTIL_ITEM_ENABLE_DISABLE,
   DEBUG_MENU_UTIL_ITEM_ON_OFF
} DebugMenuUtilItemStatus;
```

Enumeration Values

Macro	Description
`DEBUG_MENU_UTIL_ITEM_TRUE_FALSE`	Used for bool values representing a `TRUE` or `FALSE` status.
`DEBUG_MENU_UTIL_ITEM_ENABLE_DISABLE`	Used for bool values representing an `ENABLE` or `DISABLE` status.
`DEBUG_MENU_UTIL_ITEM_ON_OFF`	Used for bool values representing an `ON` or `OFF` status.

Description

An enumeration to represent the different status types that can be signified by a bool type menu item.

DebugMenuUtilItemTextAttrib

A data structure that stores the attributes of the menu item text to be drawn.

Definition

```
#include <sample_utilities/menu_utility.h>
typedef struct DebugMenuUtilItemTextAttrib {
   float textXOffset;
   float textYOffset;
   float textScale;
   uint32_t textColor;
} DebugMenuUtilItemTextAttrib;
```

Members

textXOffset	The upper left x coordinate of the first character.
textYOffset	The upper left y coordinate of the first character.
textScale	The size of the font (>`0.0`).
textColor	The color of the font (`0x00000000-0xffffffff`).

Description

A data structure that stores the attributes of the menu item text to be drawn. The attributes are the starting position <x, y> in screen coordinates, and the size and color of the text.

DebugMenuUtilItemType

An enumeration to represent the different types of menu item that can be used.

Definition

```
#include <sample_utilities/menu_utility.h>
typedef enum DebugMenuUtilItemType {
    DEBUG_MENU_UTIL_ITEM_TYPE_LABEL,
    DEBUG_MENU_UTIL_ITEM_TYPE_FLOAT,
    DEBUG_MENU_UTIL_ITEM_TYPE_INTEGER,
    DEBUG_MENU_UTIL_ITEM_TYPE_BOOL,
    DEBUG_MENU_UTIL_ITEM_TYPE_CALLBACK,
    DEBUG_MENU_UTIL_ITEM_TYPE_COUNT
} DebugMenuUtilItemType;
```

Enumeration Values

Macro	Description
DEBUG_MENU_UTIL_ITEM_TYPE_LABEL	Used to represent a menu item with a simple label or a label with a string type value.
DEBUG_MENU_UTIL_ITEM_TYPE_FLOAT	Used to represent a menu item with a float type value.
DEBUG_MENU_UTIL_ITEM_TYPE_INTEGER	Used to represent a menu item with an integer type value.
DEBUG_MENU_UTIL_ITEM_TYPE_BOOL	Used to represent a menu item with a bool type value.
DEBUG_MENU_UTIL_ITEM_TYPE_CALLBACK	Used to represent a menu item that uses a callback function.
DEBUG_MENU_UTIL_ITEM_TYPE_COUNT	The number of menu item types that are available.

Description

An enumeration to represent the different types of menu item that can be used.

DebugMenuUtilItemValueBool

A data structure to store additional values for a bool type menu item.

Definition

```
#include <sample_utilities/menu_utility.h>
typedef struct DebugMenuUtilItemValueBool {
   DebugMenuUtilItemStatus toggleType;
   bool *pCurrentStatus;
} DebugMenuUtilItemValueBool;
```

Members

toggleType	This defines the toggle type of the menu item.
pCurrentStatus	A pointer to the status of the bool type item.

Description

A data structure to store additional values for a bool type menu item. This structure includes variables to set the status of the item and the toggle type (`TRUE`/`FALSE`, `ENABLE`/`DISABLE`, and `ON`/`OFF`) it is to handle.

DebugMenuUtilItemValueCallback

A data structure to store additional values for a callback type menu item.

Definition

```
#include <sample_utilities/menu_utility.h>
typedef struct DebugMenuUtilItemValueCallback {
    DebugMenuUtilItemCallback incrementCallback;
    DebugMenuUtilItemCallback decrementCallback;
} DebugMenuUtilItemValueCallback;
```

Members

incrementCallback	A pointer to the increment callback function.
decrementCallback	A pointer to the decrement callback function.

Description

A data structure to store additional values for a callback type menu item. This structure holds two pointers to functions that return void and take no parameters. These callback functions are used for the increment and decrement actions of the menu item.

DebugMenuUtilItemValueFloat

A data structure that stores additional values for a float type menu item.

Definition

```
#include <sample_utilities/menu_utility.h>
typedef struct DebugMenuUtilItemValueFloat {
   float *pCurrentValue;
   float minValue;
   float maxValue;
   float incStep;
} DebugMenuUtilItemValueFloat;
```

Members

pCurrentValue	A pointer to the current value for the float type item.
minValue	The minimum value of this menu item.
maxValue	The maximum value of this menu item.
incStep	The step value by which the current value is changed when the LEFT or RIGHT button on the pad is pressed.

Description

A data structure that stores additional values for a float type menu item. This structure includes variables in which to store the current value of the item, its minimum and maximum range, and the step by which the value can change.

DebugMenuUtilItemValueInt

A data structure to store additional values for an integer type menu item.

Definition

```
#include <sample_utilities/menu_utility.h>
typedef struct DebugMenuUtilItemValueInt {
   int32_t *pCurrentValue;
   int32_t minValue;
   int32_t maxValue;
   int32_t incStep;
} DebugMenuUtilItemValueInt;
```

Members

pCurrentValue	A pointer to the current value for the integer type item.
minValue	The minimum value of this menu item.
maxValue	The maximum value of this menu item.
incStep	The step value by which the current value is changed when the LEFT or RIGHT button on the pad is pressed.

Description

A data structure to store additional values for an integer type menu item. This structure includes variables in which to store the current value of the item, its minimum and maximum range, and the step by which the value can change.

Functions

debugMenuUtilAddItem

The function adds a menu item that is either a simple constant label or has a string type value.

Definition

```
#include <sample_utilities/menu_utility.h>
int32_t debugMenuUtilAddItem(
    int32_t menuHandle,
    const char *itemLabel,
    void *pItemValue,
    DebugMenuUtilItemType itemType,
    bool changeProperty
);
```

Arguments

[in] *menuHandle*	The handle for the menu in question.
[in] *itemLabel*	A simple descriptor for the menu item.
[in] *pItemValue*	A void type containing values to set and update any type of menu item.
[in] *itemType*	The type of the menu item to be added.
[in] *changeProperty*	A flag that determines whether this menu item can be changed or not.

Return Values

The menu item ID on success. SCE_ERROR_ERRNO_EFAULT is returned if an incorrect parameter is passed, or if an error occurred when adding the new menu item.

Description

The function adds a menu item that is either a simple constant label or has a string type value.

debugMenuUtilAddStringValueToItem

Adds a string identifier and value pair to a label menu item.

Definition

```
#include <sample_utilities/menu_utility.h>
int32_t debugMenuUtilAddStringValueToItem(
    int32_t menuHandle,
    int32_t labelItemHandle,
    const char *name,
    int32_t value
);
```

Arguments

[in] *menuHandle*	The handle for the menu in question.
[in] *labelItemHandle*	A handle for the label menu item, which is returned by `debugMenuUtilAddItem()` when the `DEBUG_MENU_UTIL_ITEM_TYPE_LABEL` type is specified.
[in] *name*	The string specifier to be rendered.
[in] *value*	The integer value to be paired with the string.

Return Values

The index of the string value item in the label menu item on success. `SCE_ERROR_ERRNO_EFAULT` is returned on failure.

Description

Adds a string identifier and value pair to a label menu item.

debugMenuUtilCreateMenu

Creates a menu structure for use with the menu utility.

Definition

```
#include <sample_utilities/menu_utility.h>
int32_t debugMenuUtilCreateMenu();
```

Arguments

None

Return Values

A handle to the created menu object on success. `SCE_ERROR_ERRNO_EFAULT` is returned on failure.

Description

Creates a menu structure for use with the menu utility.

debugMenuUtilDecrementItemValue

Decrements the value of the currently selected menu item.

Definition

```
#include <sample_utilities/menu_utility.h>
int32_t debugMenuUtilDecrementItemValue(
   int32_t menuHandle
);
```

Arguments

[in] `menuHandle`	The handle for the menu in question.

Return Values

The index of the selected menu item in the main array on success. `SCE_ERROR_ERRNO_EFAULT` is returned if no menu with value of *`menuHandle`* is registered, or because the decrement operation failed.

Description

Decrements the value of the currently selected menu item.

debugMenuUtilIncrementItemValue

Increments the value of currently selected menu item.

Definition

```
#include <sample_utilities/menu_utility.h>
int32_t debugMenuUtilIncrementItemValue(
   int32_t menuHandle
);
```

Arguments

[in] `menuHandle`	The handle for the menu in question.

Return Values

The index of the selected menu item in the main array on success. `SCE_ERROR_ERRNO_EFAULT` is returned if no menu with value of *`menuHandle`* is registered, or because the increment operation failed.

Description

Increments the value of currently selected menu item.

debugMenuUtilInitCursorPosition

Sets the cursor position (item ID) to the first available changeable menu item.

Definition

```
#include <sample_utilities/menu_utility.h>
int32_t debugMenuUtilInitCursorPosition(
   int32_t menuHandle
);
```

Arguments

[in] `menuHandle`	The handle for the menu in question.

Return Values

The index of the first changeable menu item on success. `SCE_ERROR_ERRNO_EFAULT` is returned if no menu with a value of *`menuHandle`* is registered.

Description

Sets the cursor position (item ID) to the first available changeable menu item.

debugMenuUtilInitDefaults

Sets up a menu with default values.

Definition

```
#include <sample_utilities/menu_utility.h>
int32_t debugMenuUtilInitDefaults(
    int32_t menuHandle,
    uint32_t selItemColor,
    DebugMenuUtilItemTextAttrib defaultAttr
);
```

Arguments

[in] *menuHandle*	The handle for the menu in question.
[in] *selItemColor*	The default color to use for highlighting the selected item.
[in] *defaultAttr*	A structure containing the default attributes.

Return Values

Value	Description
SCE_OK	The operation was successful.
SCE_ERROR_ERRNO_EFAULT	The operation failed because no menu with a value of *menuHandle* is registered.

Description

Sets up a menu with default values.

debugMenuUtilRenderMenu

Draws all the text items that are registered to the menu.

Definition

```
#include <sample_utilities/menu_utility.h>
int32_t debugMenuUtilRenderMenu(
    int32_t menuHandle
);
```

Arguments

[in] `menuHandle`	The handle for the menu in question.

Return Values

Value	Description
`SCE_OK`	The operation was successful.
`SCE_ERROR_ERRNO_EFAULT`	The operation failed because no menu with value of *`menuHandle`* is registered, or because an error occurred during a menu rendering operation.

Description

Draws all the text items that are registered to the menu.

debugMenuUtilSelectDownItem

Selects the next changeable menu item.

Definition

```
#include <sample_utilities/menu_utility.h>
int32_t debugMenuUtilSelectDownItem(
   int32_t menuHandle
);
```

Arguments

[in] `menuHandle`	The handle for the menu in question.

Return Values

The index of the selected menu item in the main array or -1 if no changeable menu item is in the menu. `SCE_ERROR_ERRNO_EFAULT` is returned if no menu with a value of *`menuHandle`* is registered.

Description

Selects the next changeable menu item.

debugMenuUtilSelectUpItem

Selects the previous changeable menu item.

Definition

```
#include <sample_utilities/menu_utility.h>
int32_t debugMenuUtilSelectUpItem(
   int32_t menuHandle
);
```

Arguments

[in] *menuHandle*	The handle for the menu in question.

Return Values

The index of the selected menu item or -1 if no changeable menu item is in the menu.
`SCE_ERROR_ERRNO_EFAULT` is returned if no menu with a value of *menuHandle* is registered.

Description

Selects the previous changeable menu item.

debugMenuUtilShutdown

Clears the menu resources and exits the menu utility.

Definition

```
#include <sample_utilities/menu_utility.h>
int32_t debugMenuUtilShutdown();
```

Arguments

None

Return Values

Value	Description
`SCE_OK`	The operation was successful.

Description

Clears the menu resources and exits the menu utility.

Callback Functions

DebugMenuUtilItemCallback

A callback function for setting up user-defined menu items.

Definition

```
#include <sample_utilities/menu_utility.h>
typedef void (*DebugMenuUtilItemCallback)();
```

Arguments

None

Return Values

None

Description

A callback function for setting up user-defined menu items.