Texture Tools Reference

PlayStation®Vita Programmer Tool Runtime Library Release 3.50

Table of Contents

GXT Conversion Library

Data Types

DoSharePalettes

An enumeration defining palette sharing options.

Definition

```
#include <gxt_conversion.h>
typedef enum DoSharePalettes {
   SCE_TEXTURE_SHARE_PALETTES,
   SCE_TEXTURE_NO_SHARE_PALETTES
} DoSharePalettes;
```

Enumeration Values

Macro	Description
`SCE_TEXTURE_SHARE_PALETTES`	Share identical palettes across different textures.
`SCE_TEXTURE_NO_SHARE_PALETTES`	Keep an individual copy of each palette for every texture.

Description

An enumeration defining palette sharing options.

SceTextureContainerType

An enumeration defining texture container types.

Definition

```
#include <gxt_conversion.h>
typedef enum SceTextureContainerType {
    SCE_TEXTURE_DDS_CONTAINER,
    SCE_TEXTURE_TGA_CONTAINER,
    SCE_TEXTURE_PVR_CONTAINER
} SceTextureContainerType;
```

Enumeration Values

Macro	Description
`SCE_TEXTURE_DDS_CONTAINER`	DDS texture container.
`SCE_TEXTURE_TGA_CONTAINER`	TGA texture container.
`SCE_TEXTURE_PVR_CONTAINER`	PVR texture container.

Description

An enumeration defining texture container types.

SceTextureGxtConversionOptions

Encapsulates GXT conversion options.

Definition

```
#include <gxt_conversion.h>
typedef struct SceTextureGxtConversionOptions {
    const void *inputBuffer;
    size_t inputBufferSizeInBytes;
    SceGxmTextureFormat outputFormat;
    SceTextureMemoryLayout memoryLayout;
    const float *borderColor;
    const char *name;
    uint32_t wndWidth;
    uint32_t wndHeight;
} SceTextureGxtConversionOptions;
```

Members

inputBuffer	A pointer to the input texture data (which contains a DDS, PVR, or TGA file). This must be a valid pointer.
inputBufferSizeInBytes	The size in bytes of the input texture data.
outputFormat	The required texture format for the output GXT texture. This must correspond to a `SceGxmTextureFormat` object with an equivalent bit-depth to the input texture. Alternatively, *outputFormat* can be set to `SCE_TEXTURE_DEFAULT_OUTPUT_FORMAT` in which case the format is derived from the input texture format.
memoryLayout	The ordering of the output texture data within memory.
borderColor	A pointer to 4 floats which specify the border color (r,g,b,a). Set to `NULL` if there is no border data.
name	An identifier for the purposes of error reporting or `NULL`.
wndWidth	The texture's mapped window width in pixels. A value of 0 indicates the entire texture's width.
wndHeight	The texture's mapped window height in pixels. A value of 0 indicates the entire texture's height.

Description

Encapsulates GXT conversion options. This structure contains the set of variables which the user is required to provide to the core GXT conversion routines `sceTextureGxtConversion()` and `sceTextureGxtConversionEx()`.

Notes

For palette inputs to `sceTextureGxtConversionEx()`, the *outputFormat*, *memoryLayout*, and *borderColor* members are unused.

SceTextureMemoryLayout

An enumeration containing the possible orders of texture data within memory.

Definition

```
#include <gxt_conversion.h>
typedef enum SceTextureMemoryLayout {
    SCE_TEXTURE_GXT_SWIZZLED_MEMORY_LAYOUT = SCE_GXM_TEXTURE_SWIZZLED,
    SCE_TEXTURE_GXT_LINEAR_MEMORY_LAYOUT = SCE_GXM_TEXTURE_LINEAR,
    SCE_TEXTURE_GXT_TILED_MEMORY_LAYOUT = SCE_GXM_TEXTURE_TILED,
    SCE_TEXTURE_GXT_DEFAULT_MEMORY_LAYOUT =
    SCE_TEXTURE_GXT_SWIZZLED_MEMORY_LAYOUT
} SceTextureMemoryLayout;
```

Enumeration Values

Macro	Value	Description
`SCE_TEXTURE_GXT_SWIZZLED_MEMORY_LAYOUT`	`SCE_GXM_TEXTURE_SWIZZLED`	The texture uses a swizzled memory layout.
`SCE_TEXTURE_GXT_LINEAR_MEMORY_LAYOUT`	`SCE_GXM_TEXTURE_LINEAR`	The texture uses a linear memory layout with implicit stride.
`SCE_TEXTURE_GXT_TILED_MEMORY_LAYOUT`	`SCE_GXM_TEXTURE_TILED`	The texture uses a tiled memory layout.
`SCE_TEXTURE_GXT_DEFAULT_MEMORY_LAYOUT`	`SCE_TEXTURE_GXT_SWIZZLED_MEMORY_LAYOUT`	Default texture memory layout (swizzled).

Description

An enumeration containing the possible orders of texture data within memory. These correspond exactly to the equivalent Gxm constants (in `SceGxmTextureType`).

Functions

sceTextureGxtConversion

Converts a single input texture to the GXT format using the given conversion options.

Definition

```
#include <gxt_conversion.h>
SCE_PSP2_INTERFACE int SCE_STDCALL sceTextureGxtConversion(
    void **outputDataPtr,
    size_t *outputDataSizeInBytes,
    const SceTextureGxtConversionOptions *conversionOptions,
    const SceLoggerPtr logger,
    const SceMemoryAllocator *allocator
);
```

Arguments

[out] *outputDataPtr*	Receives the GXT data.
[out] *outputDataSizeInBytes*	Receives the size in bytes of the output GXT data.
[in] *conversionOptions*	The input texture to be converted along with the conversion options to be used.
[in] *logger*	A pointer to a `SceLogger` function to be used for logging. If `NULL` is passed for this parameter, logging will be performed using a suitable default.
[in] *allocator*	A pointer to a `SceMemoryAllocator` structure to be used for memory allocation and deallocation. If `NULL` is passed for this parameter, memory management will be performed using suitable defaults.

Return Values

Value	Description
`SCE_OK`	The operation was completed successfully.
`SCE_GXM_ERROR_INVALID_POINTER`	The operation failed due to an invalid input pointer.
`SCE_TEXTURE_ERROR_UNSUPPORTED_INPUT_FORMAT`	The operation failed because the input texture format is not supported.
`SCE_TEXTURE_ERROR_INVALID_PALETTE_DATA`	The operation failed because the input palette data is invalid.
`SCE_TEXTURE_ERROR_UNSUPPORTED_OUTPUT_FORMAT`	The operation failed because the output format is invalid or does not match the bit depth of the input texture.
`SCE_TEXTURE_ERROR_UNSUPPORTED_OUTPUT_TYPE`	The operation failed because the output memory layout is not supported.
`SCE_TEXTURE_ERROR_INVALID_DDS_DATA`	The operation failed because the input texture data is invalid.
`SCE_TEXTURE_ERROR_INVALID_PVR_DATA`	The operation failed because the input texture data is invalid.
`SCE_TEXTURE_ERROR_INCOMPLETE_CUBEMAP`	The operation failed because the input cubemap texture does not have 6 faces.

Value	Description
`SCE_TEXTURE_ERROR_INVALID_INPUT_DIMENSIONS`	The operation failed because the input texture has invalid dimensions.
`SCE_TEXTURE_ERROR_MEMORY_ALLOCATION_FAILURE`	The operation failed because a memory allocation error occurred.

Description

Converts a single input texture to the GXT format using the given conversion options.

sceTextureGxtConversionEx

Converts multiple input textures and palettes to the GXT format using the given conversion options.

Definition

```
#include <gxt_conversion.h>
SCE_PSP2_INTERFACE int SCE_STDCALL sceTextureGxtConversionEx(
    void **outputDataPtr,
    size_t *outputDataSizeInBytes,
    const SceTextureGxtConversionOptions *conversionOptions,
    uint32_t numTextures,
    uint32_t numP4Palettes,
    uint32_t numP8Palettes,
    DoSharePalettes sharePalettes,
    const SceLoggerPtr logger,
    const SceMemoryAllocator *allocator
);
```

Arguments

[out] *outputDataPtr*	Receives the GXT data.
[out] *outputDataSizeInBytes*	Receives the size in bytes of the output GXT data.
[in] *conversionOptions*	A pointer to an array of conversion inputs (textures and palettes) with their individual conversion options. The entries of this array must follow a specific order: firstly those referring to textures, secondly those referring to P4 palettes, and finally those referring to P8 palettes.
[in] *numTextures*	The number of input textures.
[in] *numP4Palettes*	The number of input 16-entry palettes.
[in] *numP8Palettes*	The number of input 256-entry palettes.
[in] *sharePalettes*	Set to `SCE_TEXTURE_SHARE_PALETTES` to only create new palettes from input textures if they don't exist from other inputs. Set to `SCE_TEXTURE_NO_SHARE_PALETTES` to always create a separate palette for each palettized input texture.
[in] *logger*	A pointer to a `SceLogger` function to be used for logging. If `NULL` is passed for this parameter, logging will be performed using a suitable default.
[in] *allocator*	A pointer to a `SceMemoryAllocator` structure to be used for memory allocation and deallocation. If `NULL` is passed for this parameter, memory management will be performed using suitable defaults.

Return Values

Value	Description
`SCE_OK`	The operation was completed successfully.
`SCE_GXM_ERROR_INVALID_POINTER`	The operation failed due to an invalid input pointer.
`SCE_TEXTURE_ERROR_UNSUPPORTED_INPUT_FORMAT`	The operation failed because the input texture format is not supported.
`SCE_TEXTURE_ERROR_INVALID_PALETTE_DATA`	The operation failed because the input palette data is invalid.

Value	Description
SCE_TEXTURE_ERROR_UNSUPPORTED_OUTPUT_FORMAT	The operation failed because the output format is invalid or does not match the bit depth of the input texture.
SCE_TEXTURE_ERROR_UNSUPPORTED_OUTPUT_TYPE	The operation failed because the output memory layout is not supported.
SCE_TEXTURE_ERROR_INVALID_DDS_DATA	The operation failed because the input texture data is invalid.
SCE_TEXTURE_ERROR_INVALID_PVR_DATA	The operation failed because the input texture data is invalid.
SCE_TEXTURE_ERROR_INCOMPLETE_CUBEMAP	The operation failed because the input cubemap texture does not have 6 faces.
SCE_TEXTURE_ERROR_INVALID_INPUT_DIMENSIONS	The operation failed because the input texture has invalid dimensions.
SCE_TEXTURE_ERROR_MEMORY_ALLOCATION_FAILURE	The operation failed because a memory allocation error occurred.

Description

Converts multiple input textures and palettes to the GXT format using the given conversion options.

sceTextureGxtConversionFinalize

Finalizes any resources allocated by a previous call to sceTextureGxtConversion().

Definition

```
#include <gxt_conversion.h>
SCE_PSP2_INTERFACE int SCE_STDCALL sceTextureGxtConversionFinalize(
    const SceLoggerPtr logger,
    const SceMemoryAllocator *allocator,
    void *textureData,
    size_t textureDataSizeInBytes
);
```

Arguments

[in] *logger*	A pointer to a SceLogger function to be used for logging. If NULL is passed for this parameter, logging will be performed using a suitable default.
[in] *allocator*	A pointer to a SceMemoryAllocator structure to be used for memory allocation and deallocation. If NULL is passed for this parameter, memory management will be performed using suitable defaults.
[in] *textureData*	A pointer to some GXT texture data previously allocated by sceTextureGxtConversion() and returned in the *outputDataPtr* argument.
[in] *textureDataSizeInBytes*	The size of the GXT texture data previously allocated by sceTextureGxtConversion() and returned in the *outputDataSizeInBytes* argument.

Return Values

Value	Description
SCE_OK	The operation was completed successfully.

Description

Finalizes any resources allocated by a previous call to sceTextureGxtConversion().

Notes

Be sure to use the same allocator and logger references that were used in the previous call to sceTextureGxtConversion().

Upon completion of this function no assumptions can be made about the contents of the buffer referred to by *textureData*.

sceTextureGxtRevert

Converts a GXT texture to either DDS, PVR, or TGA format.

Definition

```
#include <gxt_conversion.h>
SCE_PSP2_INTERFACE int SCE_STDCALL sceTextureGxtRevert(
    void **outputDataPtr,
    size_t *outputDataSizeInBytes,
    const void *inputGxt,
    size_t inputGxtSizeInBytes,
    uint32_t textureIndex,
    const SceLoggerPtr logger,
    const SceMemoryAllocator *allocator
);
```

Arguments

[out] *outputDataPtr*	Receives the DDS/PVR/TGA data.
[out] *outputDataSizeInBytes*	Receives the size in bytes of the output data.
[in] *inputGxt*	A pointer to the input GXT data.
[in] *inputGxtSizeInBytes*	The size in bytes of the input GXT data.
[in] *textureIndex*	The index of the texture to convert.
[in] *logger*	A pointer to a `SceLogger` function to be used for logging. If `NULL` is passed for this parameter, logging will be performed using a suitable default.
[in] *allocator*	A pointer to a `SceMemoryAllocator` structure to be used for memory allocation and deallocation. If `NULL` is passed for this parameter, memory management will be performed using suitable defaults.

Return Values

Value	Description
`SCE_OK`	The operation was completed successfully.
`SCE_GXM_ERROR_INVALID_POINTER`	The operation failed due to an invalid input pointer.
`SCE_TEXTURE_ERROR_INVALID_VALUE`	The operation failed due to an invalid input parameter.
`SCE_TEXTURE_ERROR_INVALID_GXT_DATA`	The operation failed because the input GXT data is not valid.
`SCE_TEXTURE_ERROR_MEMORY_ALLOCATION_FAILURE`	The operation failed because a memory allocation error occurred.

Description

Converts a GXT texture to either DDS, PVR, or TGA format. The format of the input GXT texture determines the container type of the output data:

- For PVRT compressed textures the output buffer will be in PVR format.
- For palettized textures the output buffer will be in TGA format.
- For all other texture formats the output buffer will be in DDS format.

The function `sceTextureQueryBufferType()` can be called to determine the container type of the output data.

sceTextureGxtRevertFinalize

Finalizes any resources allocated by a previous call to sceTextureGxtRevert().

Definition

```
#include <gxt_conversion.h>
SCE_PSP2_INTERFACE int SCE_STDCALL sceTextureGxtRevertFinalize(
    const SceLoggerPtr logger,
    const SceMemoryAllocator *allocator,
    void *textureData,
    size_t textureDataSizeInBytes
);
```

Arguments

[in] *logger*	A pointer to a SceLogger function to be used for logging. If NULL is passed for this parameter, logging will be performed using a suitable default.
[in] *allocator*	A pointer to a SceMemoryAllocator structure to be used for memory allocation and deallocation. If NULL is passed for this parameter, memory management will be performed using suitable defaults.
[in] *textureData*	A pointer to some GXT texture data previously allocated through sceTextureGxtRevert() and returned in the *outputDataPtr* argument.
[in] *textureDataSizeInBytes*	The size of the GXT texture data previously allocated through sceTextureGxtRevert() and returned in the *outputDataSizeInBytes* argument.

Return Values

Value	Description
SCE_OK	The operation was completed successfully.

Description

Finalizes any resources allocated by a previous call to sceTextureGxtRevert(). The user should be sure to use the same allocator and logger references that were used in the previous call to sceTextureGxtRevert(). Upon completion of this function no assumptions can be made about the contents of the buffer referred to by *textureData*.

sceTextureInitializeDefaultGxtConversionOptions

Initializes the conversion options to suitable default values.

Definition

```
#include <gxt_conversion.h>
SCE_PSP2_INTERFACE int SCE_STDCALL
sceTextureInitializeDefaultGxtConversionOptions(
    SceTextureGxtConversionOptions *conversionOptions,
    const void *inputBuffer,
    size_t inputBufferSizeInBytes
);
```

Arguments

[in] *conversionOptions*	The options for the subsequent conversion. These will be filled in with default values by this function.
[in] *inputBuffer*	A pointer to the input texture data.
[in] *inputBufferSizeInBytes*	The size in bytes of the input texture data.

Return Values

Value	Description
`SCE_OK`	The operation was completed successfully.
`SCE_GXM_ERROR_INVALID_POINTER`	The operation failed due to an invalid input pointer.

Description

Initializes the conversion options to suitable default values.

sceTextureQueryBufferType

Queries the image container type (DDS, PVR, or TGA) of a raw data buffer.

Definition

```
#include <gxt_conversion.h>
SCE_PSP2_INTERFACE int SCE_STDCALL sceTextureQueryBufferType(
    SceTextureContainerType *containerType,
    const void *inputBuffer,
    size_t inputBufferSizeInBytes
);
```

Arguments

[out] *containerType*	Receives the container type.
[in] *inputBuffer*	A pointer to the input buffer (which contains a DDS, PVR, or TGA file). This must be a valid pointer.
[in] *inputBufferSizeInBytes*	The size in bytes of the input buffer.

Return Values

Value	Description
`SCE_OK`	The operation was completed successfully.
`SCE_GXM_ERROR_INVALID_POINTER`	The operation failed due to an invalid input pointer.

Description

Queries the image container type (DDS, PVR, or TGA) of a raw data buffer.

Constants

Define Summary

Define	Value	Description
SCE_TEXTURE_GXT_DEFAULT_OUTPUT_FORMAT	((SceGxmTextureFormat)~0L)	Indicates to a GXT conversion routine that the output texture format should be derived from the input format. For a list of the default output formats for each input format see Table 2 in the *Texture Pipeline User's Guide*.

High Level PVRT Compression Library

Data Types

SceTextureMipGenerationOption

An enumeration that specifies whether sceTexturePvrtCompression() should generate mip levels in the absence of a complete mip-chain.

Definition

```
#include <pvrt_compression.h>
typedef enum SceTexturePvrtMipGenerationOption {
    SCE_TEXTURE_PVRT_GENERATE_MIPS_DISABLE = 0,
    SCE_TEXTURE_PVRT_GENERATE_MIPS_ENABLE = 1,
    SCE_TEXTURE_PVRT_DEFAULT_GENERATE_MIPS =
    SCE_TEXTURE_PVRT_GENERATE_MIPS_DISABLE
} SceTextureMipGenerationOption;
```

Enumeration Values

Macro	Value	Description
`SCE_TEXTURE_PVRT_GENERATE_MIPS_DISABLE`	`0`	Do not generate mip levels in the absence of a mip-chain.
`SCE_TEXTURE_PVRT_GENERATE_MIPS_ENABLE`	`1`	Generate mip levels in the absence of a mip-chain.
`SCE_TEXTURE_PVRT_DEFAULT_GENERATE_MIPS`	`SCE_TEXTURE_PVRT_ GENERATE_MIPS_DISABLE`	Default setting for generating mip levels in the absence of a mip-chain.

Description

An enumeration that specifies whether sceTexturePvrtCompression() should generate mip levels in the absence of a complete mip-chain.

SceTexturePvrtCompressionAlgorithm

Specifies the PVRT compression technique to use in sceTexturePvrtCompression().

Definition

```
#include <pvrt_compression.h>
typedef enum SceTexturePvrtCompressionTechnique {
    SCE_TEXTURE_PVRT_TECHNIQUE_RANGE_FIT,
    SCE_TEXTURE_PVRT_TECHNIQUE_CLUSTER_FIT,
    SCE_TEXTURE_PVRT_TECHNIQUE_DEFAULT = SCE_TEXTURE_PVRT_TECHNIQUE_RANGE_FIT
} SceTexturePvrtCompressionAlgorithm;
```

Enumeration Values

Macro	Value	Description
`SCE_TEXTURE_PVRT_TECHNIQUE_RANGE_FIT`	N/A	Use the "Range Fit" compression technique.
`SCE_TEXTURE_PVRT_TECHNIQUE_CLUSTER_FIT`	N/A	Use the "Cluster Fit" compression technique.
`SCE_TEXTURE_PVRT_TECHNIQUE_DEFAULT`	`SCE_TEXTURE_PVRT_TECHNIQUE_RANGE_FIT`	Default compression technique. Set to "Range Fit".

Description

Specifies the PVRT compression technique to use in sceTexturePvrtCompression().

Notes

Please see section 3 of the *Texture Pipeline User's Guide* document for more information on these options.

SceTexturePvrtCompressionOptions

Encapsulates PVRT compression options.

Definition

```
#include <pvrt_compression.h>
typedef struct SceTexturePvrtCompressionOptions {
    SceGxmTextureBaseFormat outputBaseFormat;
    SceTexturePvrtCompressionTechnique compressionTechnique;
    SceTexturePvrtTwoBppModulationMode twoBppModulationMode;
    SceTexturePvrtMipGenerationOption doMipGeneration;
    int numIterations;
    int numThreads;
    float flatTh;
    bool wgtAlpha;
} SceTexturePvrtCompressionOptions;
```

Members

outputBaseFormat	The required base texture format for the output PVRT texture. This should correspond to a valid PVRT format.
compressionTechnique	The compression technique to be used by `sceTexturePvrtCompression()`.
twoBppModulationMode	The block modulation mode to use for 2BPP formats.
doMipGeneration	Specifies whether mip levels should be generated in the absence of a complete mip chain.
numIterations	The number of iterations to be used by the global optimization algorithm. Specifying higher values results in greater quality but increased execution time.
numThreads	The number of threads to be used by `sceTexturePvrtCompression()`. A value of `-1` means one thread should be used per processor core.
flatTh	The variance threshold beyond which a block is considered to be flat. This variance is computed with respect to color components in the [0..255] range.
wgtAlpha	A flag that specifies whether to weight the compression error with the contents of the alpha channel. This is useful for textures where alpha denotes transparency.

Description

Encapsulates PVRT compression options. This structure contains the set of variables which the user is required to provide to the core PVRT compression routine `sceTexturePvrtCompression()`.

SceTexturePvrtThreadConfiguration

An enumeration that contains the default value for the *numThreads* member of SceTexturePvrtCompressionOptions.

Definition

```
#include <pvrt_compression.h>
typedef enum SceTexturePvrtThreadConfiguration {
    SCE_TEXTURE_PVRT_ONE_THREAD_PER_CORE = -1
} SceTexturePvrtThreadConfiguration;
```

Enumeration Values

Macro	Value	Description
SCE_TEXTURE_PVRT_ONE_THREAD_PER_CORE	-1	Spawn one worker thread per processor core.

Description

An enumeration that contains the default value for the *numThreads* member of SceTexturePvrtCompressionOptions. This default will spawn one worker thread per processor core.

SceTexturePvrtTwoBppModulationMode

An enumeration that contains the block modulation modes that can be used for 2BPP formats in sceTexturePvrtCompression().

Definition

```
#include <pvrt_compression.h>
typedef enum SceTexturePvrtTwoBppModulationMode {
    SCE_TEXTURE_PVRT_1BPP_MODULATION,
    SCE_TEXTURE_PVRT_2BPP_MODULATION,
    SCE_TEXTURE_PVRT_ADAPTIVE_MODULATION,
    SCE_TEXTURE_PVRT_DEFAULT_MODULATION = SCE_TEXTURE_PVRT_2BPP_MODULATION
} SceTexturePvrtTwoBppModulationMode;
```

Enumeration Values

Macro	Value	Description
SCE_TEXTURE_PVRT_1BPP_MODULATION	N/A	Use 1BPP modulation for all blocks.
SCE_TEXTURE_PVRT_2BPP_MODULATION	N/A	Use 2BPP modulation for all blocks.
SCE_TEXTURE_PVRT_ADAPTIVE_MODULATION	N/A	Choose the modulation mode that minimizes RMS locally.
SCE_TEXTURE_PVRT_DEFAULT_MODULATION	SCE_TEXTURE_PVRT_2BPP_MODULATION	Default block modulation mode. Set to 2BPP modulation.

Description

An enumeration that contains the block modulation modes that can be used for 2BPP formats in sceTexturePvrtCompression().

Notes

Please see section 3 of the *Texture Pipeline User's Guide* document for more information on these options.

Functions

sceTextureInitializeDefaultPvrtCompressionOptions

Initializes the compression options to suitable default values for a given base texture format.

Definition

```
#include <pvrt_compression.h>
SCE_PSP2_INTERFACE SceTextureErrorCode SCE_STDCALL
sceTextureInitializeDefaultPvrtCompressionOptions(
    SceTexturePvrtCompressionOptions *options,
    SceGxmTextureBaseFormat requiredFormat
);
```

Arguments

[out] *options*	Receives the default values.
[in] *requiredFormat*	The required base texture format for the output PVR texture.

Return Values

Value	Description
SCE_OK	The operation was completed successfully.
SCE_TEXTURE_ERROR_INVALID_POINTER	The operation failed because an invalid pointer was provided as one of the function parameters.
SCE_TEXTURE_ERROR_UNSUPPORTED_PVRT_TECH	The operation failed because the compression technique specified is not supported by this build of the compression library.

Description

Initializes the compression options to suitable default values for a given base texture format.

sceTexturePvrtCompression

Performs PVRT compression using the given compression options.

Definition

```
#include <pvrt_compression.h>
SCE_PSP2_INTERFACE SceTextureErrorCode SCE_STDCALL sceTexturePvrtCompression(
    void **outputDataBufferPtr,
    size_t *outputDataBufferSizeInBytes,
    void const *inputBuffer,
    size_t inputBufferSizeInBytes,
    SceTexturePvrtCompressionOptions const *compressionOptions,
    SceLoggerPtr const logger,
    SceMemoryAllocator const *allocator
);
```

Arguments

[out] *outputDataBufferPtr*	Receives the PVR data.
[out] *outputDataBufferSizeInBytes*	Receives the size in bytes of the output PVR data.
[in] *inputBuffer*	A pointer to a buffer containing the input DDS file. This must contain texture data in linear RGB or ARGB formats, using 8-bits per pixel.
[in] *inputBufferSizeInBytes*	The size in bytes of the input DDS texture data.
[in] *compressionOptions*	A pointer to the compression options to be used. These can either be constructed manually by the user or constructed using `sceTextureInitializeDefaultPvrtCompressionOptions()`.
[in] *logger*	A pointer to a `SceLogger` function to be used for logging. If `NULL` is passed for this parameter, logging will be performed using a suitable default.
[in] *allocator*	A pointer to a `SceMemoryAllocator` structure to be used for memory allocation and deallocation. If `NULL` is passed for this parameter, memory management will be performed using suitable defaults.

Return Values

Value	Description
`SCE_OK`	The operation was completed successfully.
`SCE_TEXTURE_ERROR_BUFFER_WRITE_FAILURE`	The operation failed because the size of the storage buffer is not sufficient to store this texture.
`SCE_TEXTURE_ERROR_INCOMPLETE_CUBEMAP`	The operation failed because the number of texture faces contained in the input file are not enough to form a cubemap.
`SCE_TEXTURE_ERROR_INVALID_DDS_DATA`	The operation failed because the input file is either corrupted or is not in the DDS format.
`SCE_TEXTURE_ERROR_INVALID_POINTER`	The operation failed because an invalid pointer was provided as one of the function parameters.
`SCE_TEXTURE_ERROR_MEMORY_ALLOCATION_FAILURE`	The operation failed because it could not allocate the internal data needed to perform the compression.

Value	Description
`SCE_TEXTURE_ERROR_UNSUPPORTED_INPUT_FORMAT`	The operation failed because the input file is encoded in an unsupported format.
`SCE_TEXTURE_ERROR_UNSUPPORTED_INPUT_TYPE`	The operation failed because the input file data is arranged in an unsupported layout.
`SCE_TEXTURE_ERROR_UNSUPPORTED_PVRT_BPP`	The operation failed because the compression BPP count specified is not supported by this build of the compression library.
`SCE_TEXTURE_ERROR_UNSUPPORTED_PVRT_FORMAT`	The operation failed because the compression format specified is not supported by this build of the compression library.

Description

Performs PVRT compression using the given compression options.

sceTexturePvrtCompressionFinalize

Finalizes any resources allocated by a previous call to sceTexturePvrtCompression().

Definition

```
#include <pvrt_compression.h>
SCE_PSP2_INTERFACE SceTextureErrorCode SCE_STDCALL
sceTexturePvrtCompressionFinalize(
    SceLoggerPtr const logger,
    SceMemoryAllocator const *allocator,
    void *textureData,
    size_t textureDataSizeInBytes
);
```

Arguments

[in] *logger*	A pointer to a `SceLogger` function to be used for logging. If `NULL` is passed for this parameter, logging will be performed using a suitable default.
[in] *allocator*	A pointer to a `SceMemoryAllocator` structure to be used for memory allocation and deallocation. If `NULL` is passed for this parameter, memory management will be performed using suitable defaults.
[in] *textureData*	A pointer to some PVR texture data previously allocated using `sceTexturePvrtCompression()`. The pointer is returned in the *outputDataPtr* argument passed to `sceTexturePvrtCompression()`.
[in] *textureDataSizeInBytes*	The size of the PVR texture data previously allocated using `sceTexturePvrtCompression()`. The data size is returned in the *outputDataSizeInBytes* argument passed to `sceTexturePvrtCompression()`.

Return Values

Value	Description
`SCE_OK`	The operation was completed successfully.

Description

Finalizes any resources allocated by a previous call to `sceTexturePvrtCompression()`.

Notes

Be sure to use the same allocator and logger references that were used in the previous call to `sceTexturePvrtCompression()`.

Upon completion of this function no assumptions can be made about the contents of the buffer referred to by the *textureData* argument.

sceTexturePvrtDecompression

Performs PVRT decompression.

Definition

```
#include <pvrt_compression.h>
SCE_PSP2_INTERFACE SceTextureErrorCode SCE_STDCALL
sceTexturePvrtDecompression(
   void **outputDataBufferPtr,
   size_t *outputDataBufferSizeInBytes,
   void const *inputBuffer,
   size_t inputBufferSizeInBytes,
   SceLoggerPtr const logger,
   SceMemoryAllocator const *allocator
);
```

Arguments

[out] *outputDataBufferPtr*	Receives the DDS data.
[out] *outputDataBufferSizeInBytes*	Receives the size in bytes of the output DDS data.
[in] *inputBuffer*	A pointer to a buffer containing the input PVR file.
[in] *inputBufferSizeInBytes*	The size in bytes of the input PVR texture data.
[in] *logger*	A pointer to a `SceLogger` function to be used for logging. If `NULL` is passed for this parameter, logging will be performed using a suitable default.
[in] *allocator*	A pointer to a `SceMemoryAllocator` structure to be used for memory allocation and deallocation. If `NULL` is passed for this parameter, memory management will be performed using suitable defaults.

Return Values

Value	Description
`SCE_OK`	The operation was completed successfully.
`SCE_TEXTURE_ERROR_BUFFER_WRITE_FAILURE`	The operation failed because the size of the storage buffer is not sufficient to store this texture.
`SCE_TEXTURE_ERROR_INVALID_PVR_DATA`	The operation failed because the input file is either corrupted or is not in the PVR format.
`SCE_TEXTURE_ERROR_INVALID_POINTER`	The operation failed because an invalid pointer was provided as one of the function parameters.
`SCE_TEXTURE_ERROR_MEMORY_ALLOCATION_FAILURE`	The operation failed because it could not allocate the internal data needed to perform the decompression.

Description

Performs PVRT decompression.

sceTexturePvrtDecompressionFinalize

Finalizes any resources allocated by a previous call to `sceTexturePvrtDecompression()`.

Definition

```
#include <pvrt_compression.h>
SCE_PSP2_INTERFACE SceTextureErrorCode SCE_STDCALL
sceTexturePvrtDecompressionFinalize(
    SceLoggerPtr const logger,
    SceMemoryAllocator const *allocator,
    void *textureData,
    size_t textureDataSizeInBytes
);
```

Arguments

[in] *`logger`*	A pointer to a `SceLogger` function to be used for logging. If `NULL` is passed for this parameter, logging will be performed using a suitable default.
[in] *`allocator`*	A pointer to a `SceMemoryAllocator` structure to be used for memory allocation and deallocation. If `NULL` is passed for this parameter, memory management will be performed using suitable defaults.
[in] *`textureData`*	A pointer to some DDS texture data previously allocated through `sceTexturePvrtDecompression()` and returned in the *`outputDataPtr`* argument.
[in] *`textureDataSizeInBytes`*	The size of the DDS texture data previously allocated through `sceTexturePvrtDecompression()` and returned in the *`outputDataSizeInBytes`* argument.

Return Values

Value	Description
`SCE_OK`	The operation was completed successfully.

Description

Finalizes any resources allocated by a previous call to `sceTexturePvrtDecompression()`.

Notes

Be sure to use the same allocator and logger references that were used in the previous call to `sceTexturePvrtCompression()`.

Upon completion of this function no assumptions can be made about the contents of the buffer referred to by the *`textureData`* argument.

sceTexturePvrtDifferenceRMS

Computes the RMS of the difference between two images in DDS format.

Definition

```
#include <pvrt_compression.h>
SCE_PSP2_INTERFACE SceTextureErrorCode SCE_STDCALL
sceTexturePvrtDifferenceRMS(
    float *rms,
    void const *texture0Data,
    size_t texture0DataSizeInBytes,
    void const *texture1Data,
    size_t texture1DataSizeInBytes,
    SceLoggerPtr const logger,
    SceMemoryAllocator const *allocator
);
```

Arguments

[out] *rms*	Receives the computed result.
[in] *texture0Data*	A pointer to the texture data for the first image in DDS format.
[in] *texture0DataSizeInBytes*	The size of the texture data referenced by *texture0Data*.
[in] *texture1Data*	A pointer to the texture data for the second image in DDS format.
[in] *texture1DataSizeInBytes*	The size of the texture data referenced by *texture1Data*.
[in] *logger*	A pointer to a SceLogger function to be used for logging. If NULL is passed for this parameter, logging will be performed using a suitable default.
[in] *allocator*	A pointer to a SceMemoryAllocator structure to be used for memory allocation and deallocation. If NULL is passed for this parameter, memory management will be performed using suitable defaults.

Return Values

Value	Description
SCE_OK	The was operation completed successfully.
SCE_TEXTURE_ERROR_INCOMPLETE_CUBEMAP	The operation failed because the number of texture faces contained in at least one of the input files are not enough to form a cubemap.
SCE_TEXTURE_ERROR_INVALID_DDS_DATA	The operation failed because at least one of the input files is corrupted.
SCE_TEXTURE_ERROR_INVALID_POINTER	The operation failed because an invalid pointer was provided as one of the function parameters.
SCE_TEXTURE_ERROR_MEMORY_ALLOCATION_FAILURE	The operation failed because it could not allocate the internal data needed to perform the comparison.
SCE_TEXTURE_ERROR_UNSUPPORTED_INPUT_FORMAT	The operation failed because at least one of the input files is not in the DDS format.

Description

Computes the RMS of the difference between two images in DDS format.

sceTexturePvrtDifferenceRMSExt

Computes the RMS of the difference between a specified number of levels and faces belonging to two DDS format textures.

Definition

```
#include <pvrt_compression.h>
SCE_PSP2_INTERFACE SceTextureErrorCode SCE_STDCALL
sceTexturePvrtDifferenceRMSExt(
   float *rms,
   void const *texture0Data,
   size_t texture0DataSizeInBytes,
   void const *texture1Data,
   size_t texture1DataSizeInBytes,
   uint32_t maxLevels,
   uint32_t maxFaces,
   SceLoggerPtr const logger,
   SceMemoryAllocator const *allocator
);
```

Arguments

[out] *rms*	Receives the computed result.
[in] *texture0Data*	A pointer to the texture data for the first image in DDS format.
[in] *texture0DataSizeInBytes*	The size of the texture data referenced by *texture0Data*.
[in] *texture1Data*	A pointer to the texture data for the second image in DDS format.
[in] *texture1DataSizeInBytes*	The size of the texture data referenced by *texture1Data*.
maxLevels	The maximum number of mipmap levels to compare between the two textures.
maxFaces	The maximum number of cubemap faces to compare between the two textures.
logger	A pointer to a `SceLogger` function to be used for logging. If `NULL` is passed for this parameter, then logging will be performed using a suitable default.
allocator	A pointer to a `SceMemoryAllocator` structure to be used for memory allocation and deallocation. If `NULL` is passed for this parameter, then memory management will be performed using suitable defaults.

Return Values

Value	Description
`SCE_OK`	The operation was completed successfully.
`SCE_TEXTURE_ERROR_INCOMPLETE_CUBEMAP`	The operation failed because the number of texture faces contained in at least one of the input files are not enough to form a cubemap.
`SCE_TEXTURE_ERROR_INVALID_DDS_DATA`	The operation failed because at least one of the input files is corrupted.
`SCE_TEXTURE_ERROR_INVALID_POINTER`	The operation failed because an invalid pointer was provided as one of the function parameters.

Value	Description
`SCE_TEXTURE_ERROR_MEMORY_ALLOCATION_FAILURE`	The operation failed because it could not allocate the internal data needed to perform the comparison.
`SCE_TEXTURE_ERROR_UNSUPPORTED_INPUT_FORMAT`	The operation failed because at least one of the input files is not in the DDS format.

Description

Computes the RMS of the difference between a specified number of levels and faces belonging to two DDS format textures. This function is a variant of `sceTexturePvrtDifferenceRMS()` which restricts the maximum number of faces and levels to compare.

sceTexturePvrtPadInput

Pads the dimensions of an input DDS texture to the enclosing power of two, mirroring its contents where applicable.

Definition

```
#include <pvrt_compression.h>
SCE_PSP2_INTERFACE SceTextureErrorCode SCE_STDCALL sceTexturePvrtPadInput(
   void **tgtData,
   uint32_t srcWidth,
   uint32_t srcHeight,
   uint32_t srcComps,
   int32_t srcMips,
   uint32_t srcFaces,
   void const *srcData,
   SceLoggerPtr const logger,
   SceMemoryAllocator const *allocator
);
```

Arguments

[out] *tgtData*	Receives the padded DDS data.
[in] *srcWidth*	The width of the input texture.
[in] *srcHeight*	The height of the input texture.
[in] *srcComps*	The number of components of the input texture.
[in] *srcMips*	The number of mip levels of the input texture.
[in] *srcFaces*	The number of cubemap faces of the input texture.
[in] *srcData*	A pointer to input texture data in DDS format.
[in] *logger*	A pointer to a `SceLogger` function to be used for logging. If `NULL` is passed for this parameter, logging will be performed using a suitable default.
[in] *allocator*	A pointer to a `SceMemoryAllocator` structure to be used for memory allocation and deallocation. If `NULL` is passed for this parameter, memory management will be performed using suitable defaults.

Return Values

Value	Description
`SCE_OK`	The operation was completed successfully.
`SCE_TEXTURE_ERROR_INCOMPLETE_CUBEMAP`	The operation failed because the number of texture faces contained in the input file are not enough to form a cubemap.
`SCE_TEXTURE_ERROR_INVALID_DDS_DATA`	The operation failed because the input file is either corrupted or is not in the DDS format.
`SCE_TEXTURE_ERROR_INVALID_POINTER`	The operation failed because an invalid pointer was provided as one of the function parameters.
`SCE_TEXTURE_ERROR_UNSUPPORTED_INPUT_BPP`	The operation failed because the color component count specified is not supported.
`SCE_TEXTURE_ERROR_MEMORY_ALLOCATION_FAILURE`	The operation failed because it could not allocate the data needed to store the padded texture.

Description

Pads the dimensions of an input DDS texture to the enclosing power of two, mirroring its contents where applicable.

Low Level PVRT Compression Library

Data Types

ScePvrtTask

The opaque data structure for the compression task.

Definition

```
#include <pvrt.h>
typedef struct ScePvrtTask;
```

Description

The opaque data structure for the compression task.

Functions

scePvrtCompressImage

Compresses a single texture plane.

Definition

```
#include <pvrt.h>
SCE_PVRT_EXPORT SceTextureErrorCode scePvrtCompressImage(
   uint8_t const *src,
   uint32_t width,
   uint32_t height,
   uint32_t comps,
   uint32_t flags,
   int32_t iters,
   int32_t threads,
   float flatTh,
   void *tgt
);
```

Arguments

[in] `src`	A pointer to the source image data, which should be in R8G8B8A8 or R8G8B8 format.
[in] `width`	The width of the source image.
[in] `height`	The height of the source image.
[in] `comps`	The number of color components in the source image.
[in] `flags`	A set of flags indicating the compression format, BPP, and the technique to use when compressing.
[in] `iters`	The number of iterations to perform. A value of -1 uses the default indicated by the library.
[in] `threads`	The number of threads to use (if the library build supports multi-threading). A value of -1 uses all available cores.
[in] `flatTh`	The variance threshold beyond which a block is considered to be flat. This variance is computed with respect to color components in the [0..255] range.
[out] `tgt`	A memory block that receives the compressed data.

Return Values

Value	Description
`SCE_OK`	The operation was successful.
`SCE_TEXTURE_ERROR_UNSUPPORTED_INPUT_BPP`	The operation failed because the color component count specified is not supported.
`SCE_TEXTURE_ERROR_UNSUPPORTED_PVRT_FORMAT`	The operation failed because the compression format specified is not supported by this build of the compression library.
`SCE_TEXTURE_ERROR_UNSUPPORTED_PVRT_TECH`	The operation failed because the compression technique specified is not supported by this build of the compression library.

Value	Description
`SCE_TEXTURE_ERROR_UNSUPPORTED_PVRT_BPP`	The operation failed because the compression BPP count specified is not supported by this build of the compression library.
`SCE_TEXTURE_ERROR_UNSUPPORTED_PVRT_BPP2_MOD`	The operation failed because the modulation for BPP2 mode is not supported by this build of the compression library.
`SCE_TEXTURE_ERROR_NPOT_HEIGHT`	The operation failed because the source image height is not a power of two.
`SCE_TEXTURE_ERROR_NPOT_WIDTH`	The operation failed because the source image width is not a power of two.
`SCE_TEXTURE_ERROR_PVRT_COMMAND_SEQ_ORDER`	The operation failed because the operation failed because of an internal error.
`SCE_TEXTURE_ERROR_TOO_MANY_OPEN_PVRT_TASKS`	The operation failed because there are too many active compression tasks.

Description

Compresses a single texture plane. No mips levels or multiple cubemap faces are taken into account.

scePvrtCompressImageMemReqs

Computes the runtime memory necessary to compress an image.

Definition

```
#include <pvrt.h>
SCE_PVRT_EXPORT SceTextureErrorCode scePvrtCompressImageMemReqs(
   uint32_t width,
   uint32_t height,
   uint32_t comps,
   uint32_t flags,
   int32_t iters,
   int32_t threads,
   float flatTh,
   size_t *size
);
```

Arguments

[in] *width*	The width of the source image.
[in] *height*	The height of the source image.
[in] *comps*	The number of color components in the source image.
[in] *flags*	A set of compression flags indicating the compression format, BPP, and the technique to use when compressing.
[in] *iters*	The number of iterations to perform. A value of -1 uses the default indicated by the library.
[in] *threads*	The number of threads to use (if the library build supports multi-threading). A value of -1 uses all available cores.
[in] *flatTh*	The variance threshold beyond which a block is considered to be flat. This variance is computed with respect to color components in the [0..255] range.
[out] *size*	Receives the size of runtime memory required to compress the source image.

Return Values

Value	Description
`SCE_OK`	The operation was successful.
`SCE_TEXTURE_ERROR_UNSUPPORTED_INPUT_BPP`	The operation failed because the color component count specified is not supported.
`SCE_TEXTURE_ERROR_UNSUPPORTED_PVRT_FORMAT`	The operation failed because the compression format specified is not supported by this build of the compression library.
`SCE_TEXTURE_ERROR_UNSUPPORTED_PVRT_TECH`	The operation failed because the compression technique specified is not supported by this build of the compression library.
`SCE_TEXTURE_ERROR_UNSUPPORTED_PVRT_BPP`	The operation failed because the compression BPP count specified is not supported by this build of the compression library.
`SCE_TEXTURE_ERROR_UNSUPPORTED_PVRT_BPP2_MOD`	The operation failed because the modulation for BPP2 mode is not supported by this build of the compression library.

Value	Description
`SCE_TEXTURE_ERROR_NPOT_HEIGHT`	The operation failed because the source image height is not a power of two.
`SCE_TEXTURE_ERROR_NPOT_WIDTH`	The operation failed because the source image width is not a power of two.
`SCE_TEXTURE_ERROR_PVRT_COMMAND_SEQ_ORDER`	The operation failed because of an internal error.

Description

Computes the runtime memory necessary to compress an image.

scePvrtCompressImagePreAlloc

Compresses an image using a preallocated buffer as dynamic memory.

Definition

```
#include <pvrt.h>
SCE_PVRT_EXPORT SceTextureErrorCode scePvrtCompressImagePreAlloc(
   uint8_t const *src,
   uint32_t width,
   uint32_t height,
   uint32_t comps,
   uint32_t flags,
   int32_t iters,
   int32_t threads,
   float flatTh,
   void *buf,
   size_t bufSize,
   void *tgt
);
```

Arguments

[in] `src`	A pointer to the source image data, in R8G8B8A8 or R8G8B8 format.
[in] `width`	The width of the source image.
[in] `height`	The height of the source image.
[in] `comps`	The number of color components in the source image.
[in] `flags`	A set of compression flags indicating the compression format, BPP, and the technique to use when compressing.
[in] `iters`	The number of iterations to perform. A value of -1 uses the default indicated by the library.
[in] `threads`	The number of threads to use if the library build supports multi-threading. A value of -1 uses all available cores.
[in] `flatTh`	The variance threshold beyond which a block is considered to be flat. This variance is computed with respect to color components in the [0..255] range.
[in] `buf`	A pointer to a preallocated buffer to use as dynamic memory.
[in] `bufSize`	The size of the preallocated buffer to use as dynamic memory.
[out] `tgt`	A memory block that receives the compressed data.

Return Values

Value	Description
`SCE_OK`	The operation was successful.
`SCE_TEXTURE_ERROR_UNSUPPORTED_INPUT_BPP`	The operation failed because the color component count specified is not supported.
`SCE_TEXTURE_ERROR_UNSUPPORTED_PVRT_FORMAT`	The operation failed because the compression format specified is not supported by this build of the compression library.
`SCE_TEXTURE_ERROR_UNSUPPORTED_PVRT_TECH`	The operation failed because the compression technique specified is not supported by this build of the compression library.

Value	Description
SCE_TEXTURE_ERROR_UNSUPPORTED_PVRT_BPP	The operation failed because the compression BPP count specified is not supported by this build of the compression library.
SCE_TEXTURE_ERROR_UNSUPPORTED_PVRT_BPP2_MOD	The operation failed because the modulation for the BPP2 mode is not supported by this build of the compression library.
SCE_TEXTURE_ERROR_NPOT_HEIGHT	The operation failed because the source image height is not a power of two.
SCE_TEXTURE_ERROR_NPOT_WIDTH	The operation failed because the source image width is not a power of two.
SCE_TEXTURE_ERROR_PVRT_COMMAND_SEQ_ORDER	The operation failed because of an internal error.
SCE_TEXTURE_ERROR_TOO_MANY_OPEN_PVRT_TASKS	The operation failed because there are too many active compression tasks.

Description

Compresses an image using a preallocated buffer as dynamic memory. This function is a version of `scePvrtCompressImage()` that uses a preallocated buffer as dynamic memory. The size required for a given image and its compression settings can be obtained from `scePvrtCompressImageMemReqs()`.

scePvrtCompressTexture

Computes the compressed version of a texture consisting of multiple mip levels.

Definition

```
#include <pvrt.h>
SCE_PVRT_EXPORT SceTextureErrorCode scePvrtCompressTexture(
    uint8_t const *src,
    uint32_t width,
    uint32_t height,
    uint32_t comps,
    uint32_t *mips,
    uint32_t flags,
    int32_t iters,
    int32_t threads,
    float flatTh,
    void *tgt
);
```

Arguments

[in] `src`	A pointer to the source image data (in R8G8B8A8 or R8G8B8 format) with successive mip levels stored sequentially.
[in] `width`	The width of the finest mip level of the source texture.
[in] `height`	The height of the finest mip level of the source texture.
[in] `comps`	The number of color components in the source texture.
[in,out] `mips`	A pointer to the number of mip levels in the source texture. Receives the number of mip levels the compressed texture can store.
[in] `flags`	A set of compression flags indicating the compression format, BPP, and the technique to use when compressing.
[in] `iters`	The number of iterations to perform. A value of -1 uses the default indicated by the library.
[in] `threads`	The number of threads to use (if the library build supports multi-threading). A value of -1 uses all available cores.
[in] `flatTh`	The variance threshold beyond which a block is considered to be flat. This variance is computed with respect to color components in the [0..255] range.
[out] `tgt`	Receives the compressed data.

Return Values

Value	Description
`SCE_OK`	The operation was successful.
`SCE_TEXTURE_ERROR_UNSUPPORTED_INPUT_BPP`	The operation failed because the color component count specified is not supported.
`SCE_TEXTURE_ERROR_UNSUPPORTED_PVRT_FORMAT`	The operation failed because the compression format specified is not supported by this build of the compression library.
`SCE_TEXTURE_ERROR_UNSUPPORTED_PVRT_TECH`	The operation failed because the compression technique specified is not supported by this build of the compression library.

Value	Description
`SCE_TEXTURE_ERROR_UNSUPPORTED_PVRT_BPP`	The operation failed because the compression BPP count specified is not supported by this build of the compression library.
`SCE_TEXTURE_ERROR_UNSUPPORTED_PVRT_BPP2_MOD`	The operation failed because the modulation for BPP2 mode is not supported by this build of the compression library.
`SCE_TEXTURE_ERROR_NPOT_HEIGHT`	The operation failed because the source image height is not a power of two.
`SCE_TEXTURE_ERROR_NPOT_WIDTH`	The operation failed because the source image width is not a power of two.
`SCE_TEXTURE_ERROR_PVRT_COMMAND_SEQ_ORDER`	The operation failed because of an internal error.
`SCE_TEXTURE_ERROR_TOO_MANY_OPEN_PVRT_TASKS`	The operation failed because there are too many active compression tasks.

Description

Computes the compressed version of a texture consisting of multiple mip levels.

scePvrtCompressTextureMemReqs

Computes the runtime memory necessary to compress a texture face.

Definition

```
#include <pvrt.h>
SCE_PVRT_EXPORT SceTextureErrorCode scePvrtCompressTextureMemReqs(
   uint32_t width,
   uint32_t height,
   uint32_t comps,
   uint32_t *mips,
   uint32_t flags,
   int32_t iters,
   int32_t threads,
   float flatTh,
   size_t *size
);
```

Arguments

[in] *width*	The width of the finest mip level of the source texture.
[in] *height*	The height of the finest mip level of the source texture.
[in] *comps*	The number of color components in the source texture.
[in,out] *mips*	A pointer to the number of mip levels in the source texture. Receives how many mip levels the compressed texture can store.
[in] *flags*	A set of compression flags indicating the compression format, BPP, and the technique to use when compressing.
[in] *iters*	The number of iterations to perform. A value of -1 uses the default indicated by the library.
[in] *threads*	The number of threads to use (if the library build supports multi-threading). A value of -1 uses all available cores.
[in] *flatTh*	The variance threshold beyond which a block is considered to be flat. This variance is computed with respect to color components in the [0..255] range.
[out] *size*	Receives the size of runtime memory required to compress the source texture face.

Return Values

Value	Description
`SCE_OK`	The operation was successful.
`SCE_TEXTURE_ERROR_UNSUPPORTED_INPUT_BPP`	The operation failed because the color component count specified is not supported.
`SCE_TEXTURE_ERROR_UNSUPPORTED_PVRT_FORMAT`	The operation failed because the compression format specified is not supported by this build of the compression library.
`SCE_TEXTURE_ERROR_UNSUPPORTED_PVRT_TECH`	The operation failed because the compression technique specified is not supported by this build of the compression library.
`SCE_TEXTURE_ERROR_UNSUPPORTED_PVRT_BPP`	The operation failed because the compression BPP count specified is not supported by this build of the compression library.

Value	Description
`SCE_TEXTURE_ERROR_UNSUPPORTED_PVRT_BPP2_MOD`	The operation failed because the modulation for BPP2 mode is not supported by this build of the compression library.
`SCE_TEXTURE_ERROR_NPOT_HEIGHT`	The operation failed because the source image height is not a power of two.
`SCE_TEXTURE_ERROR_NPOT_WIDTH`	The operation failed because the source image width is not a power of two.
`SCE_TEXTURE_ERROR_PVRT_COMMAND_SEQ_ORDER`	The operation failed because of an internal error.
`SCE_TEXTURE_ERROR_TOO_MANY_OPEN_PVRT_TASKS`	The operation failed because there are too many active compression tasks.

Description

Computes the runtime memory necessary to compress a texture face.

scePvrtCompressTexturePreAlloc

Compresses a texture face using a preallocated buffer as dynamic memory.

Definition

```
#include <pvrt.h>
SCE_PVRT_EXPORT SceTextureErrorCode scePvrtCompressTexturePreAlloc(
    uint8_t const *src,
    uint32_t width,
    uint32_t height,
    uint32_t comps,
    uint32_t *mips,
    uint32_t flags,
    int32_t iters,
    int32_t threads,
    float flatTh,
    void *buf,
    size_t bufSize,
    void *tgt
);
```

Arguments

[in] *src*	A pointer to the source image data (in R8G8B8A8 or R8G8B8 format) with successive mip levels stored sequentially.
[in] *width*	The width of the finest mip level of the source texture.
[in] *height*	The height of the finest mip level of the source texture.
[in] *comps*	The number of color components in the source texture.
[in,out] *mips*	A pointer to the number of mip levels in the source texture. Returns how many mip levels the compressed texture can store.
[in] *flags*	A set of compression flags indicating the compression format, BPP, and the technique to use when compressing.
[in] *iters*	The number of iterations to perform. A value of -1 uses the default indicated by the library.
[in] *threads*	The number of threads to use (if the library build supports multi-threading). A value of -1 uses all available cores.
[in] *flatTh*	The variance threshold beyond which a block is considered to be flat. This variance is computed with respect to color components in the [0..255] range.
[in] *buf*	A pointer to the preallocated buffer to use as dynamic memory.
[in] *bufSize*	The size of the preallocated buffer to use as dynamic memory.
[out] *tgt*	A memory block that receives the compressed data.

Return Values

Value	Description
`SCE_OK`	The operation was successful.
`SCE_TEXTURE_ERROR_UNSUPPORTED_INPUT_BPP`	The operation failed because the color component count specified is not supported.
`SCE_TEXTURE_ERROR_UNSUPPORTED_PVRT_FORMAT`	The operation failed because the compression format specified is not supported by this build of the compression library.

Value	Description
`SCE_TEXTURE_ERROR_UNSUPPORTED_PVRT_TECH`	The operation failed because the compression technique specified is not supported by this build of the compression library.
`SCE_TEXTURE_ERROR_UNSUPPORTED_PVRT_BPP`	The operation failed because the compression BPP count specified is not supported by this build of the compression library.
`SCE_TEXTURE_ERROR_UNSUPPORTED_PVRT_BPP2_MOD`	The operation failed because the modulation for BPP2 mode is not supported by this build of the compression library.
`SCE_TEXTURE_ERROR_NPOT_HEIGHT`	The operation failed because the source image height is not a power of two.
`SCE_TEXTURE_ERROR_NPOT_WIDTH`	The operation failed because the source image width is not a power of two.
`SCE_TEXTURE_ERROR_PVRT_COMMAND_SEQ_ORDER`	The operation failed because of an internal error.
`SCE_TEXTURE_ERROR_TOO_MANY_OPEN_PVRT_TASKS`	The operation failed because there are too many active compression tasks.

Description

Compresses a texture face using a preallocated buffer as dynamic memory. This function is a version of `scePvrtCompressTexture()` that uses a preallocated buffer as dynamic memory. The size required for a given image and the compression settings can be obtained from `scePvrtCompressTextureMemReqs()`.

scePvrtComputeMipMaps

Computes mip levels for an image up to the maximum allowed by the given compression format and BPP.

Definition

```
#include <pvrt.h>
SCE_PVRT_EXPORT SceTextureErrorCode scePvrtComputeMipMaps(
    uint8_t const *src,
    uint32_t width,
    uint32_t height,
    uint32_t comps,
    int32_t *mips,
    uint8_t *tgt
);
```

Arguments

[in] *src*	A pointer to the source image data (in R8G8B8A8 or R8G8B8 format).
[in] *width*	The width of the source image.
[in] *height*	The height of the source image.
[in] *comps*	The number of color components in the source image.
[in,out] *mips*	A pointer to the number of mip levels in the source texture. Receives the number of mip levels that can be stored.
[out] *tgt*	A memory block that receives the mipmap data sequentially.

Return Values

Value	Description
SCE_OK	The operation was successful.
SCE_TEXTURE_ERROR_UNSUPPORTED_INPUT_BPP	The operation failed because the color component count specified is not supported.
SCE_TEXTURE_ERROR_UNSUPPORTED_PVRT_FORMAT	The operation failed because the compression format specified is not supported by this build of the compression library.
SCE_TEXTURE_ERROR_UNSUPPORTED_PVRT_BPP	The operation failed because the compression BPP count specified is not supported by this build of the compression library.
SCE_TEXTURE_ERROR_NPOT_HEIGHT	The operation failed because the source image height is not a power of two.
SCE_TEXTURE_ERROR_NPOT_WIDTH	The operation failed because the source image width is not a power of two.

Description

Computes mip levels for an image up to the maximum allowed by the given compression format and BPP.

scePvrtCreateCompressTask

Creates a compression task.

Definition

```
#include <pvrt.h>
SCE_PVRT_EXPORT SceTextureErrorCode scePvrtCreateCompressTask(
   uint8_t const *src,
   uint32_t width,
   uint32_t height,
   uint32_t comps,
   uint32_t flags,
   uint32_t threads,
   float flatTh,
   void *tgt,
   ScePvrtTask **task
);
```

Arguments

[in] *src*	A pointer to the source image data (in R8G8B8A8 or R8G8B8 format).
[in] *width*	The width of the source image.
[in] *height*	The height of the source image.
[in] *comps*	The number of color components in the source image.
[in] *flags*	A set of compression flags indicating the compression format, BPP, and the technique to use when compressing.
[in] *threads*	The number of threads the compression will be parallelized into.
[in] *flatTh*	The variance threshold beyond which a block is considered to be flat. This variance is computed with respect to color components in the [0..255] range.
[out] *tgt*	A memory block that receives the compressed data.
[out] *task*	Receives the task handle.

Return Values

Value	Description
SCE_OK	The operation was successful.
SCE_TEXTURE_ERROR_UNSUPPORTED_INPUT_BPP	The operation failed because the color component count specified is not supported.
SCE_TEXTURE_ERROR_UNSUPPORTED_PVRT_FORMAT	The operation failed because the compression format specified is not supported by this build of the compression library.
SCE_TEXTURE_ERROR_UNSUPPORTED_PVRT_BPP	The operation failed because the compression BPP count specified is not supported by this build of the compression library.
SCE_TEXTURE_ERROR_NPOT_HEIGHT	The operation failed because the source image height is not a power of two.
SCE_TEXTURE_ERROR_NPOT_WIDTH	The operation failed because the source image width is not a power of two.
SCE_TEXTURE_ERROR_TOO_MANY_OPEN_PVRT_TASKS	The operation failed because there are too many active compression tasks.

Description

Creates a compression task. This can later be used through a low-level interface, which is provided by this library, to perform parallelized compression on the client side. This can be done using whichever implementation of threading primitives is available.

The compression algorithm works by minimizing the compression error iteratively. The iteration is preceded by a small prologue, `scePvrtTaskPrologue()`, which computes a low cost coarse approximation. The iteration must also be followed by an epilogue, `scePvrtTaskEpilogue()`, which performs the actual encoding in the PVRTC format of choice.

The iteration itself is the most costly step of the algorithm and, when performing 4 iteration steps, uses about 90% of the computation time. It can be parallelized but it needs to perform spread and gather steps. This is because it is solving a non-separable problem and needs to combine the results from each thread prior to the next iteration step. The steps are carried out in `scePvrtTaskSpread()` and `scePvrtTaskGather()`. The body of each thread computation is contained in `scePvrtTaskRun()`, and all the threaded calls for a given iteration must have finished prior to the corresponding gather call. The library checks this condition and will signal a `SCE_TEXTURE_ERROR_PVRT_COMMAND_SEQ_ORDER` error if it is not met.

scePvrtCreateCompressTaskPreAlloc

Creates a compression task which uses a preallocated buffer as dynamic memory.

Definition

```
#include <pvrt.h>
SCE_PVRT_EXPORT SceTextureErrorCode scePvrtCreateCompressTaskPreAlloc(
    uint8_t const *src,
    uint32_t width,
    uint32_t height,
    uint32_t comps,
    uint32_t flags,
    uint32_t threads,
    float flatTh,
    void *tgt,
    void *buf,
    size_t bufSize,
    ScePvrtTask **task
);
```

Arguments

[in] *src*	A pointer to the source image data (in R8G8B8A8 or R8G8B8 format).
[in] *width*	The width of the source image.
[in] *height*	The height of the source image.
[in] *comps*	The number of color components in the source image.
[in] *flags*	A set of compression flags indicating the compression format and BPP as well as the technique to use.
[in] *threads*	The number of threads the compression will be parallelized into.
[in] *flatTh*	The variance threshold beyond which a block is considered to be flat. This variance is computed with respect to color components in the [0..255] range.
[out] *tgt*	A memory block that receives the compressed data.
[in] *buf*	A pointer to the preallocated buffer to use as dynamic memory.
[in] *bufSize*	The size of the preallocated buffer to use as dynamic memory.
[out] *task*	Receives the task handle.

Return Values

Value	Description
`SCE_OK`	The operation was successful.
`SCE_TEXTURE_ERROR_UNSUPPORTED_INPUT_BPP`	The operation failed because the color component count specified is not supported.
`SCE_TEXTURE_ERROR_UNSUPPORTED_PVRT_FORMAT`	The operation failed because the compression format specified is not supported by this build of the compression library.
`SCE_TEXTURE_ERROR_UNSUPPORTED_PVRT_BPP`	The operation failed because the compression BPP count specified is not supported by this build of the compression library.
`SCE_TEXTURE_ERROR_NPOT_HEIGHT`	The operation failed because the source image height is not a power of two.
`SCE_TEXTURE_ERROR_NPOT_WIDTH`	The operation failed because the source image width is not a power of two.

Value	Description
`SCE_TEXTURE_ERROR_TOO_MANY_OPEN_PVRT_TASKS`	The operation failed because there are too many active compression tasks.

Description

Creates a compression task which uses a preallocated buffer as dynamic memory. This function is a version of `scePvrtCreateCompressTask()` where the created compression task uses a preallocated buffer as dynamic memory. The size required for a given image and compression settings can be obtained from `scePvrtCompressImageMemReqs()`.

scePvrtDecompressImage

Decompresses a single texture plane.

Definition

```
#include <pvrt.h>
SCE_PVRT_EXPORT SceTextureErrorCode scePvrtDecompressImage(
   void const *src,
   uint32_t width,
   uint32_t height,
   uint32_t comps,
   uint32_t flags,
   uint8_t *tgt
);
```

Arguments

[in] *src*	A pointer to the compressed data.
[in] *width*	The width of the compressed image.
[in] *height*	The height of the compressed image.
[in] *comps*	The number of color components in the compressed image.
[in] *flags*	A set of flags indicating the compression format and BPP.
[out] *tgt*	A memory block that receives the uncompressed image.

Return Values

Value	Description
SCE_OK	The operation was successful.
SCE_TEXTURE_ERROR_UNSUPPORTED_INPUT_BPP	The operation failed because the color component count specified is not supported.
SCE_TEXTURE_ERROR_UNSUPPORTED_PVRT_FORMAT	The operation failed because the compression format specified is not supported by this build of the compression library.
SCE_TEXTURE_ERROR_UNSUPPORTED_PVRT_BPP	The operation failed because the compression BPP count specified is not supported by this build of the compression library.
SCE_TEXTURE_ERROR_NPOT_HEIGHT	The operation failed because the compressed image height is not a power of two.
SCE_TEXTURE_ERROR_NPOT_WIDTH	The operation failed because the compressed image width is not a power of two.

Description

Decompresses a single texture plane. This function can be used repeatedly on a data stream to proccess multiple cubemap faces and/or mip levels.

scePvrtDecompressImageMemReqs

Computes the runtime memory necessary to decompress an image.

Definition

```
#include <pvrt.h>
SCE_PVRT_EXPORT SceTextureErrorCode scePvrtDecompressImageMemReqs(
    uint32_t width,
    uint32_t height,
    uint32_t comps,
    uint32_t flags,
    size_t *size
);
```

Arguments

[in] *width*	The width of the compressed image.
[in] *height*	The height of the compressed image.
[in] *comps*	The number of color components in the compressed image.
[in] *flags*	A set of compression flags indicating the compression format and BPP.
[out] *size*	Receives the memory size required to decompress the compressed image.

Return Values

Value	Description
SCE_OK	The operation was successful.
SCE_TEXTURE_ERROR_UNSUPPORTED_INPUT_BPP	The operation failed because the color component count specified is not supported.
SCE_TEXTURE_ERROR_UNSUPPORTED_PVRT_FORMAT	The operation failed because the compression format specified is not supported by this build of the compression library.
SCE_TEXTURE_ERROR_UNSUPPORTED_PVRT_BPP	The operation failed because the compression BPP count specified is not supported by this build of the compression library.
SCE_TEXTURE_ERROR_NPOT_HEIGHT	The operation failed because the compressed image height is not a power of two.
SCE_TEXTURE_ERROR_NPOT_WIDTH	The operation failed because the compressed image width is not a power of two.

Description

Computes the runtime memory necessary to decompress an image.

scePvrtDecompressImagePreAlloc

Decompresses an image using a preallocated buffer as dynamic memory.

Definition

```
#include <pvrt.h>
SCE_PVRT_EXPORT SceTextureErrorCode scePvrtDecompressImagePreAlloc(
    void const *src,
    uint32_t width,
    uint32_t height,
    uint32_t comps,
    uint32_t flags,
    void *buf,
    size_t bufSize,
    uint8_t *tgt
);
```

Arguments

[in] *src*	A pointer to the compressed data.
[in] *width*	The width of the compressed image.
[in] *height*	The height of the compressed image.
[in] *comps*	The number of color components in the compressed image.
[in] *flags*	A set of flags indicating the compression format and BPP.
[in] *buf*	A pointer to the preallocated buffer to use as dynamic memory.
[in] *bufSize*	The size of the preallocated buffer to use as dynamic memory.
[out] *tgt*	A memory block that receives the uncompressed image.

Return Values

Value	Description
SCE_OK	The operation was successful.
SCE_TEXTURE_ERROR_UNSUPPORTED_INPUT_BPP	The operation failed because the color component count specified is not supported.
SCE_TEXTURE_ERROR_UNSUPPORTED_PVRT_FORMAT	The operation failed because the compression format specified is not supported by this build of the compression library.
SCE_TEXTURE_ERROR_UNSUPPORTED_PVRT_BPP	The operation failed because the compression BPP count specified is not supported by this build of the compression library.
SCE_TEXTURE_ERROR_NPOT_HEIGHT	The operation failed because the compressed image height is not a power of two.
SCE_TEXTURE_ERROR_NPOT_WIDTH	The operation failed because the compressed image width is not a power of two.

Description

Decompresses an image using a preallocated buffer as dynamic memory. This function is a version of scePvrtDecompressImage() that uses a preallocated buffer as dynamic memory. The size required for a given image and compression settings can be obtained from scePvrtDecompressImageMemReqs().

scePvrtDecompressTexture

Decompresses a texture consisting of multiple mip levels.

Definition

```
#include <pvrt.h>
SCE_PVRT_EXPORT SceTextureErrorCode scePvrtDecompressTexture(
    void const *src,
    uint32_t width,
    uint32_t height,
    uint32_t comps,
    uint32_t mips,
    uint32_t flags,
    uint8_t *tgt
);
```

Arguments

[in] *src*	A pointer to the compressed data.
[in] *width*	The width of the compressed image.
[in] *height*	The height of the compressed image.
[in] *comps*	The number of color components in the compressed texture.
[in] *mips*	The number of mip levels in the compressed texture.
[in] *flags*	A set of flags indicating the compression format and BPP.
[out] *tgt*	A pointer to a memory block that receives the uncompressed texture.

Return Values

Value	Description
SCE_OK	The operation was successful and the compressed image was stored in the memory block supplied.
SCE_TEXTURE_ERROR_UNSUPPORTED_INPUT_BPP	The operation failed because the color component count specified is not supported.
SCE_TEXTURE_ERROR_UNSUPPORTED_PVRT_FORMAT	The compression format specified is not supported by this build of the compression library.
SCE_TEXTURE_ERROR_UNSUPPORTED_PVRT_BPP	The compression BPP count specified is not supported by this build of the compression library.
SCE_TEXTURE_ERROR_NPOT_HEIGHT	The compressed image height is not a power of two.
SCE_TEXTURE_ERROR_NPOT_WIDTH	The compressed image width is not a power of two.

Description

Decompresses a texture consisting of multiple mip levels. This function can be used repeatedly on a data stream to process multiple cubemap faces.

scePvrtDecompressTextureMemReqs

Computes the memory size necessary to decompress a texture.

Definition

```
#include <pvrt.h>
SCE_PVRT_EXPORT SceTextureErrorCode scePvrtDecompressTextureMemReqs(
    uint32_t width,
    uint32_t height,
    uint32_t comps,
    uint32_t mips,
    uint32_t flags,
    size_t *size
);
```

Arguments

[in] *width*	The width of the compressed texture.
[in] *height*	The height of the compressed texture.
[in] *comps*	The number of color components in the compressed texture.
[in] *mips*	The number of mip levels in the compressed texture.
[in] *flags*	A set of flags indicating the compression format and BPP.
[out] *size*	A pointer to the `size_t` type variables in which to store the required size.

Return Values

Value	Description
`SCE_OK`	The necessary memory size was correctly computed.
`SCE_TEXTURE_ERROR_UNSUPPORTED_INPUT_BPP`	The operation failed because the color component count specified is not supported.
`SCE_TEXTURE_ERROR_UNSUPPORTED_PVRT_FORMAT`	The compression format specified is not supported by this build of the compression library.
`SCE_TEXTURE_ERROR_UNSUPPORTED_PVRT_BPP`	The compression BPP count specified is not supported by this build of the compression library.
`SCE_TEXTURE_ERROR_NPOT_HEIGHT`	The compressed image height is not a power of two.
`SCE_TEXTURE_ERROR_NPOT_WIDTH`	The compressed image width is not a power of two.

Description

Computes the memory size necessary to decompress a texture.

scePvrtDecompressTexturePreAlloc

Decompresses a texture using a preallocated buffer as dynamic memory.

Definition

```
#include <pvrt.h>
SCE_PVRT_EXPORT SceTextureErrorCode scePvrtDecompressTexturePreAlloc(
    void const *src,
    uint32_t width,
    uint32_t height,
    uint32_t comps,
    uint32_t mips,
    uint32_t flags,
    void *buf,
    size_t bufSize,
    uint8_t *tgt
);
```

Arguments

[in] *src*	A pointer to the compressed data.
[in] *width*	The width of the compressed texture.
[in] *height*	The height of the compressed texture.
[in] *comps*	The number of color components in the compressed texture.
[in] *mips*	The number of mip levels in the compressed texture.
[in] *flags*	A set of flags indicating the compression format and BPP.
[out] *buf*	A pointer to the preallocated buffer to use as dynamic memory.
[out] *bufSize*	The size of the preallocated buffer to use as dynamic memory.
[out] *tgt*	A pointer to a memory block that receives the uncompressed texture.

Return Values

Value	Description
SCE_OK	The operation was successful and the compressed image was stored in the memory block supplied.
SCE_TEXTURE_ERROR_UNSUPPORTED_INPUT_BPP	The operation failed because the color component count specified is not supported.
SCE_TEXTURE_ERROR_UNSUPPORTED_PVRT_FORMAT	The compression format specified is not supported by this build of the compression library.
SCE_TEXTURE_ERROR_UNSUPPORTED_PVRT_BPP	The compression BPP count specified is not supported by this build of the compression library.
SCE_TEXTURE_ERROR_NPOT_HEIGHT	The compressed image height is not a power of two.
SCE_TEXTURE_ERROR_NPOT_WIDTH	The compressed image width is not a power of two.

Description

Decompresses a texture using a preallocated buffer as dynamic memory. This function is a version of `scePvrtDecompressTexture()` that uses a preallocated buffer as dynamic memory. The size required for a given image and compression settings can be obtained from `scePvrtDecompressTextureMemReqs()`.

scePvrtDeleteCompressTask

Deletes a compression task, previously created with scePvrtCreateCompressTask(), at any point of its execution.

Definition

```
#include <pvrt.h>
SCE_PVRT_EXPORT SceTextureErrorCode scePvrtDeleteCompressTask(
    ScePvrtTask *task
);
```

Arguments

[in] *task*	The task handle.

Return Values

Value	Description
SCE_OK	The operation was successful.

Description

Deletes a compression task, previously created with scePvrtCreateCompressTask(), at any point of its execution. It deallocates all associated memory.

scePvrtImageComponents

Computes the number of color components used in a compressed texture face.

Definition

```
#include <pvrt.h>
SCE_PVRT_EXPORT SceTextureErrorCode scePvrtImageComponents(
   void const *src,
   uint32_t width,
   uint32_t height,
   uint32_t flags,
   uint32_t *comps
);
```

Arguments

[in] *src*	A pointer to the compressed data.
[in] *width*	The width of the compressed image.
[in] *height*	The height of the compressed image.
[in] *flags*	A set of flags indicating the compression format and BPP.
[out] *comps*	Receives the computed number of components.

Return Values

Value	Description
`SCE_OK`	The operation was successful.
`SCE_TEXTURE_ERROR_UNSUPPORTED_INPUT_BPP`	The operation failed because the color component count specified is not supported.
`SCE_TEXTURE_ERROR_UNSUPPORTED_PVRT_FORMAT`	The operation failed because the compression format specified is not supported by this build of the compression library.
`SCE_TEXTURE_ERROR_UNSUPPORTED_PVRT_BPP`	The operation failed because the compression BPP count specified is not supported by this build of the compression library.
`SCE_TEXTURE_ERROR_INVALID_INPUT_DIMENSIONS`	The operation failed because the source image dimensions are not a power of two.

Description

Computes the number of color components used in a compressed texture face.

scePvrtImageSize

Computes the memory needed to store a compressed version of a single texture plane.

Definition

```
#include <pvrt.h>
SCE_PVRT_EXPORT SceTextureErrorCode scePvrtImageSize(
    uint32_t width,
    uint32_t height,
    uint32_t flags,
    size_t *size
);
```

Arguments

[in] *width*	The width of the source image.
[in] *height*	The height of the source image.
[in] *flags*	A set of flags that indicate the compression format and BPP count.
[out] *size*	Receives the computed memory size.

Return Values

Value	Description
SCE_OK	The operation was successful.
SCE_TEXTURE_ERROR_UNSUPPORTED_PVRT_FORMAT	The operation failed because the compression format specified is not supported by this build of the compression library.
SCE_TEXTURE_ERROR_UNSUPPORTED_PVRT_BPP	The operation failed because the compression BPP count specified is not supported by this build of the compression library.
SCE_TEXTURE_ERROR_INVALID_INPUT_DIMENSIONS	The operation failed because the source image dimensions are not a power of two.

Description

Computes the memory needed to store a compressed version of a single texture plane. No mip levels or multiple cubemap faces are taken into account during computation.

scePvrtMipMapsSize

Computes the memory size required to store mip levels for an image including the base level.

Definition

```
#include <pvrt.h>
SCE_PVRT_EXPORT SceTextureErrorCode scePvrtMipMapsSize(
   uint32_t width,
   uint32_t height,
   uint32_t comps,
   int32_t *mips,
   size_t *size
);
```

Arguments

[in] *width*	The width of the source image.
[in] *height*	The height of the source image.
[in] *comps*	The number of color components in the source image.
[in,out] *mips*	A pointer to the number of mip levels in the source texture. Receives how many mip levels the image can store.
[out] *size*	Receives the computed memory size.

Return Values

Value	Description
`SCE_OK`	The operation was successful.
`SCE_TEXTURE_ERROR_UNSUPPORTED_INPUT_BPP`	The operation failed because the color component count specified is not supported.
`SCE_PVRTC_ERROR_NON_POW_2_HEIGHT`	The operation failed because the source image height is not a power of two.
`SCE_PVRTC_ERROR_NON_POW_2_WIDTH`	The operation failed because the source image width is not a power of two.

Description

Computes the memory size required to store mip levels for an image including the base level.

scePvrtPadInput

Pads the input size up to the next power of two.

Definition

```
#include <pvrt.h>
SCE_PVRT_EXPORT SceTextureErrorCode scePvrtPadInput(
    uint8_t const *src,
    uint32_t width,
    uint32_t height,
    uint32_t comps,
    int32_t mips,
    uint32_t faces,
    uint8_t *tgt
);
```

Arguments

[in] *src*	A pointer to the source image data (in R8G8B8A8 or R8G8B8 format).
[in] *width*	The width of the source image.
[in] *height*	The height of the source image.
[in] *comps*	The number of color components in the source image.
[in] *mips*	The number of mip levels in the source image.
[in] *faces*	The number of cubemap faces in the source image.
[out] *tgt*	A memory block that receives the padded data.

Return Values

Value	Description
SCE_OK	The operation was successful.
SCE_TEXTURE_ERROR_UNSUPPORTED_INPUT_BPP	The operation failed because the color component count specified is not supported.

Description

Pads the input size up to the next power of two.

scePvrtTaskCurrMemSize

Computes the amount of runtime memory currently used by a compression task.

Definition

```
#include <pvrt.h>
SCE_PVRT_EXPORT SceTextureErrorCode scePvrtTaskCurrMemSize(
    ScePvrtTask *task,
    size_t *size
);
```

Arguments

[in] *task*	The task handle.
[out] *size*	Receives the amount of runtime memory currently used by a compression task.

Return Values

Value	Description
SCE_OK	The operation was successful.

Description

Computes the amount of runtime memory currently used by a compression task.

scePvrtTaskEpilogue

Task epilogue.

Definition

```
#include <pvrt.h>
SCE_PVRT_EXPORT SceTextureErrorCode scePvrtTaskEpilogue(
    ScePvrtTask *task
);
```

Arguments

[in] *task*	The task handle.

Return Values

Value	Description
SCE_OK	The operation was successful.
SCE_TEXTURE_ERROR_PVRT_COMMAND_SEQ_ORDER	The operation failed because this command was run in an incorrect order.

Description

Task epilogue. This function must be run once after all iterations have concluded.

scePvrtTaskGather

Task gather.

Definition

```
#include <pvrt.h>
SCE_PVRT_EXPORT SceTextureErrorCode scePvrtTaskGather(
    ScePvrtTask *task
);
```

Arguments

[in] *task*	The task handle.

Return Values

Value	Description
`SCE_OK`	The operation was successful.
`SCE_TEXTURE_ERROR_PVRT_COMMAND_SEQ_ORDER`	The operation failed because this command was run in an incorrect order.

Description

Task gather. This function must be run once at the end of each iteration when the execution of all calls to `scePvrtTaskRun()` have finished.

scePvrtTaskPeakMemSize

Computes the maximum amount of runtime memory used by a compression task up to this point of execution.

Definition

```
#include <pvrt.h>
SCE_PVRT_EXPORT SceTextureErrorCode scePvrtTaskPeakMemSize(
    ScePvrtTask *task,
    size_t *size
);
```

Arguments

[in] *task*	The task handle.
[out] *size*	Receives the maximum amount of runtime memory currently used by a compression task.

Return Values

Value	Description
SCE_OK	The operation was successful.

Description

Computes the maximum amount of runtime memory used by a compression task up to this point of execution.

scePvrtTaskPrologue

Task prologue.

Definition

```
#include <pvrt.h>
SCE_PVRT_EXPORT SceTextureErrorCode scePvrtTaskPrologue(
    ScePvrtTask *task
);
```

Arguments

[in] *task*	The task handle.

Return Values

Value	Description
`SCE_OK`	The operation was successful.
`SCE_TEXTURE_ERROR_PVRT_COMMAND_SEQ_ORDER`	The operation failed because the command was run in an incorrect order.

Description

Task prologue. This function must be run once per task prior to any iteration.

scePvrtTaskRun

Task threaded run.

Definition

```
#include <pvrt.h>
SCE_PVRT_EXPORT SceTextureErrorCode scePvrtTaskRun(
    ScePvrtTask *task,
    uint32_t thread
);
```

Arguments

[in] *task*	The task handle.
[in] *thread*	The thread index.

Return Values

Value	Description
SCE_OK	The operation was successful.
SCE_TEXTURE_ERROR_PVRT_COMMAND_SEQ_ORDER	The operation failed because this command was run in an incorrect order.

Description

Task threaded run. This function must be run once in the body of each iteration for each thread specified in `scePvrtCreateCompressTask()`. Calls to the function can be made in parallel.

scePvrtTaskSpread

Task spread.

Definition

```
#include <pvrt.h>
SCE_PVRT_EXPORT SceTextureErrorCode scePvrtTaskSpread(
    ScePvrtTask *task
);
```

Arguments

[in] *task*	The task handle.

Return Values

Value	Description
`SCE_OK`	The operation was successful.
`SCE_TEXTURE_ERROR_PVRT_COMMAND_SEQ_ORDER`	The operation failed because the command was run in an incorrect order.

Description

Task spread. This function must be run once at the beginning of each iteration.

scePvrtTextureSize

Computes the memory size required to store the compressed version of a texture consisting of multiple mip levels.

Definition

```
#include <pvrt.h>
SCE_PVRT_EXPORT SceTextureErrorCode scePvrtTextureSize(
    uint32_t width,
    uint32_t height,
    uint32_t *mips,
    uint32_t flags,
    size_t *size
);
```

Arguments

[in] *width*	The width of the finest mip level of the source texture.
[in] *height*	The height of the finest mip level of the source texture.
[in,out] *mips*	A pointer to the number of mip levels in the source texture. Receives the number of mip levels the compressed texture can store.
[in] *flags*	A set of flags indicating the compression format and BPP.
[out] *size*	Receives the computed memory size.

Return Values

Value	Description
SCE_OK	The operation was successful.
SCE_TEXTURE_ERROR_UNSUPPORTED_PVRT_FORMAT	The operation failed because the compression format specified is not supported by this build of the compression library.
SCE_TEXTURE_ERROR_UNSUPPORTED_PVRT_BPP	The operation failed because the compression BPP count specified is not supported by this build of the compression library.
SCE_TEXTURE_ERROR_NPOT_HEIGHT	The operation failed because the source image height is not a power of two.
SCE_TEXTURE_ERROR_NPOT_WIDTH	The operation failed because the source image width is not a power of two.

Description

Computes the memory size required to store the compressed version of a texture consisting of multiple mip levels.

scePvrtVerticalFlip

Performs an in-place vertical flip of a PVRT texture.

Definition

```
#include <pvrt.h>
SCE_PVRT_EXPORT SceTextureErrorCode scePvrtVerticalFlip(
    uint32_t width,
    uint32_t height,
    uint32_t comps,
    uint32_t mips,
    uint32_t flags,
    void *data
);
```

Arguments

[in] *width*	The width of the source image.
[in] *height*	The height of the source image.
[in] *comps*	The number of color components in the source image.
[in] *mips*	The number of mip levels in the source image.
[in] *flags*	A set of flags indicating the compression format and BPP.
[out] *data*	A memory block that receives the flipped texture data.

Return Values

Value	Description
SCE_OK	The operation was successful.
SCE_TEXTURE_ERROR_UNSUPPORTED_INPUT_BPP	The operation failed because the color component count specified is not supported.
SCE_TEXTURE_ERROR_UNSUPPORTED_PVRT_FORMAT	The operation failed because the compression format specified is not supported by this build of the compression library.
SCE_TEXTURE_ERROR_UNSUPPORTED_PVRT_BPP	The operation failed because the compression BPP count specified is not supported by this build of the compression library.

Description

Performs an in-place vertical flip of a PVRT texture.

Constants

Define Summary

Define	Value	Description
`SCE_PVRT_FLAGS_1BGR`	`(1 << 8)`	A flag that specifies 1BGR encoding.
`SCE_PVRT_FLAGS_BPP2`	`(1 << 1)`	A flag that specifies 2BPP compression.
`SCE_PVRT_FLAGS_BPP2_MASK`	`SCE_PVRT_FLAGS_BPP2_MIXED`	The mask for all modulation variants in the 2BPP mode.
`SCE_PVRT_FLAGS_BPP2_MIXED`	`(SCE_PVRT_FLAGS_BPP2_MOD0 | SCE_PVRT_FLAGS_BPP2_MOD1)`	A flag that specifies the use of mixed modulation in 2BPP mode.
`SCE_PVRT_FLAGS_BPP2_MOD0`	`(1 << 6)`	A flag that specifies the use of 1BPP modulation in 2BPP mode.
`SCE_PVRT_FLAGS_BPP2_MOD1`	`(1 << 7)`	A flag that specifies the use of 2BPP modulation in 2BPP mode.
`SCE_PVRT_FLAGS_BPP4`	`(1 << 0)`	A flag that specifies 4BPP compression.
`SCE_PVRT_FLAGS_BPP_MASK`	`(SCE_PVRT_FLAGS_BPP2 | SCE_PVRT_FLAGS_BPP4)`	The mask for all compression flags.
`SCE_PVRT_FLAGS_CLUSTER_FIT`	`(1 << 3)`	A flag that specifies a compression technique consisting of an optimal bounding fit achieved by testing all cluster arrangements.
`SCE_PVRT_FLAGS_COMP_MASK`	`(SCE_PVRT_FLAGS_ABGR | SCE_PVRT_FLAGS_1BGR)`	The mask for all component encoding variants.
`SCE_PVRT_FLAGS_FORMAT_MASK`	`(SCE_PVRT_FLAGS_TCI | SCE_PVRT_FLAGS_TCII)`	The mask for all format flags.
`SCE_PVRT_FLAGS_RANGE_FIT`	`(1 << 2)`	A flag that specifies a compression technique consisting of a range fit for the bounding amplitude.
`SCE_PVRT_FLAGS_TCI`	`(1 << 4)`	A flag that specifies the TC-I format.
`SCE_PVRT_FLAGS_TCII`	`(1 << 5)`	A flag that specifies the TC-II format.
`SCE_PVRT_FLAGS_TECH_MASK`	`(SCE_PVRT_FLAGS_RANGE_FIT | SCE_PVRT_FLAGS_CLUSTER_FIT)`	The mask for all compression technique flags.
`SCE_PVRT_FLAGS_WEIGHT_BY_ALPHA`	`(1 << 9)`	A flag that specifies the use of alpha as weighting in the encoding.

UBC Helper Functions

Functions

sceUbcImageSize

Computes the memory required to store a compressed version of a single texture plane.

Definition

```
#include <ubc.h>
SCE_UBC_EXPORT SceTextureErrorCode sceUbcImageSize(
    uint32_t width,
    uint32_t height,
    uint32_t flags,
    size_t *size
);
```

Arguments

[in] *width*	The width of the source image.
[in] *height*	The height of the source image.
[in] *flags*	A flag that specifies the compression format to use.
[out] *size*	Receives the computed memory size.

Return Values

Value	Description
SCE_OK	The operation was successful.
SCE_TEXTURE_ERROR_UNSUPPORTED_UBC_FORMAT	The operation failed because the compression format specified is not supported by this build of the compression library.

Description

Computes the memory required to store a compressed version of a single texture plane. No mips levels or multiple planes are taken into account during computation.

sceUbcTextureSize

Computes the memory required to store the compressed version of a texture that consists of multiple mipmap levels.

Definition

```
#include <ubc.h>
SCE_UBC_EXPORT SceTextureErrorCode sceUbcTextureSize(
   uint32_t width,
   uint32_t height,
   uint32_t *mips,
   uint32_t flags,
   size_t *size
);
```

Arguments

[in] *width*	The width of the finest mipmap level of the source texture.
[in] *height*	The height of the finest mipmap level of the source texture.
[in,out] *mips*	A pointer to the number of mipmap levels in the source texture. Receives the number of levels that the compressed texture can store.
[in] *flags*	A flag that specifies the compression format to use.
[out] *size*	Receives the computed memory size.

Return Values

Value	Description
SCE_OK	The operation was successful.
SCE_TEXTURE_ERROR_UNSUPPORTED_UBC_FORMAT	The operation failed because the compression format specified is not supported by this build of the compression library.

Description

Computes the memory required to store the compressed version of a texture that consists of multiple mipmap levels.

Constants

Define Summary

Define	Value	Description
SCE_UBC_FLAGS_FORMAT_MASK	(SCE_UBC_FLAGS_UBC1 | SCE_UBC_FLAGS_UBC2 | SCE_UBC_FLAGS_UBC3 | SCE_UBC_FLAGS_UBC4 | SCE_UBC_FLAGS_UBC5)	Mask for all format flags.
SCE_UBC_FLAGS_UBC1	(1 << 0)	Flag indicating UBC1 format.
SCE_UBC_FLAGS_UBC2	(1 << 1)	Flag indicating UBC2 format.
SCE_UBC_FLAGS_UBC3	(1 << 2)	Flag indicating UBC3 format.
SCE_UBC_FLAGS_UBC4	(1 << 3)	Flag indicating UBC4 format.
SCE_UBC_FLAGS_UBC5	(1 << 4)	Flag indicating UBC5 format.

Core Texture Library

sce::Texture Summary

sce::Texture

A namespace containing the texture framework.

Definition

```
namespace Texture {}
```

Description

A namespace containing the texture framework.

Variables

Public Variables

`uint32_t const kDefaultDataAlignmentInBytes`	The default texture data alignment.

Function Summary

Function	Description
`convertFace`	Converts a texture cubemap face in a particular color format to another using a specific mipmap level alignment.
`convertLevel`	Converts a texture mipmap level from one particular color format to another using a specific scanline alignment.
`convertPalette`	Converts palette data from one particular color format to another.
`convertPixel`	Converts color data from one particular color format to another.
`convertTexture`	Converts a full texture in a particular color format to another using a specific cubemap face alignment.
`deSwizzleLevel`	Converts swizzled data to a linear layout.
`faceSize`	Computes the size of a texture cubemap face using a specific mipmap level alignment.
`levelSize`	Computes the size of a texture mipmap level using a specific scanline alignment.
`operator==`	A comparison operator for palettes.
`paletteSize`	Computes the size of a palette.
`size`	Computes the size (in bytes) of a multi-face texture.
`swizzleLevel`	Converts linear data to a swizzled layout.
`textureSize`	Computes the size of a full texture using a specific cubemap face alignment.
`unpackFace`	Unpacks a palettized texture cubemap face using a given palette and a specific mipmap level alignment.
`unpackLevel`	Unpacks a palettized texture mipmap level using a given palette and a specific scanline alignment.
`unpackTexture`	Unpacks a full palettized texture using a given palette and a specific cubemap face alignment.

Inner Classes, Structures, and Namespaces

Item	Description
sce::Texture::Data	The container class for a single texture.
sce::Texture::DDS	A namespace containing `DDS` functionality.

Item	Description
sce::Texture::Face	A namespace containing functionality for a single texture face.
sce::Texture::File	A container class for a collection of textures and palettes.
sce::Texture::Gxt	A namespace containing GXT functionality.
sce::Texture::Palette	A container class for a texture palette.
sce::Texture::PVR	A namespace containing `PVR` functionality.
sce::Texture::TGA	A namespace containing `TGA` functionality.

sce::Texture Type Definitions

Encoding

An enumeration containing the supported modalities of texture data encoding.

Definition

```
#include <texture.h>
typedef enum sce::Texture::Encoding {
    ENCODING_DIRECT,
    ENCODING_INDEXED_4BPP,
    ENCODING_INDEXED_8BPP,
    ENCODING_PVRT_2BPP,
    ENCODING_PVRT_4BPP,
    ENCODING_PVRTII_2BPP,
    ENCODING_PVRTII_4BPP,
    ENCODING_UBC1,
    ENCODING_UBC2,
    ENCODING_UBC3,
    ENCODING_UBC4,
    ENCODING_UBC5,
    ENCODING_UBC5_ATI,
    ENCODING_SBC4,
    ENCODING_SBC5
} Encoding;
```

Enumeration Values

Macro	Description
ENCODING_DIRECT	Directly encoded data.
ENCODING_INDEXED_4BPP	Indexed data using 4 bits per pixel.
ENCODING_INDEXED_8BPP	Indexed data using 8 bits per pixel.
ENCODING_PVRT_2BPP	PVR TCI block-compressed data using 2 bits per pixel.
ENCODING_PVRT_4BPP	PVR TCI block-compressed data using 4 bits per pixel.
ENCODING_PVRTII_2BPP	PVR TCII block-compressed data using 2 bits per pixel.
ENCODING_PVRTII_4BPP	PVR TCII block-compressed data using 4 bits per pixel.
ENCODING_UBC1	UBC1 block-compressed data.
ENCODING_UBC2	UBC2 block-compressed data.
ENCODING_UBC3	UBC3 block-compressed data.
ENCODING_UBC4	UBC4 block-compressed data.
ENCODING_UBC5	UBC5 block-compressed data.
ENCODING_UBC5_ATI	UBC5 block-compressed data, swizzled components.
ENCODING_SBC4	SBC4 block-compressed data.
ENCODING_SBC5	SBC5 block-compressed data.

Description

An enumeration containing the supported modalities of texture data encoding.

Layout

An enumeration containing the different ways in which texture data can be arranged in memory.

Definition

```
#include <texture.h>
typedef enum sce::Texture::Layout {
    LAYOUT_LINEAR,
    LAYOUT_SWIZZLED
} Layout;
```

Enumeration Values

Macro	Description
`LAYOUT_LINEAR`	Linearly arranged data.
`LAYOUT_SWIZZLED`	Swizzled data.

Description

An enumeration containing the different ways in which texture data can be arranged in memory.

sce::Texture Functions

convertFace

Converts a texture cubemap face in a particular color format to another using a specific mipmap level alignment.

Definition

```
#include <pixel_format.h>
void convertFace(
   uint32_t width,
   uint32_t height,
   uint32_t numLevels,
   PixelFormat const &srcPf,
   uint8_t const *& src,
   PixelFormat const &tgtPf,
   uint8_t *& tgt,
   size_t srcLineAlign = 1,
   size_t srcLevelAlign = 1,
   size_t tgtLineAlign = 1,
   size_t tgtLevelAlign = 1
);
```

Arguments

[in] *width*	The base mipmap level width.
[in] *height*	The base mipmap level height.
[in] *numLevels*	The number of mipmap levels.
[in] *srcPf*	The original data format.
[in] *src*	A pointer to the original data.
[in] *tgtPf*	The target data format.
[out] *tgt*	Receives the converted texture cubemap level.
[in] *srcLineAlign*	Optional. The original scanline alignment in bytes. Defaults to 1.
[in] *srcLevelAlign*	Optional. The original mipmap level alignment in bytes. Defaults to 1.
[in] *tgtLineAlign*	Optional. The target scanline alignment in bytes. Defaults to 1.
[in] *tgtLevelAlign*	Optional. The target mipmap level alignment in bytes. Defaults to 1.

Return Values

None

Description

Converts a texture cubemap face in a particular color format to another using a specific mipmap level alignment.

convertLevel

Converts a texture mipmap level from one particular color format to another using a specific scanline alignment.

Definition

```
#include <pixel_format.h>
void convertLevel(
   uint32_t width,
   uint32_t height,
   PixelFormat const &srcPf,
   uint8_t const *& src,
   PixelFormat const &tgtPf,
   uint8_t *& tgt,
   size_t srcLineAlign = 1,
   size_t tgtLineAlign = 1
);
```

Arguments

[in] *width*	The mipmap level width.
[in] *height*	The mipmap level height.
[in] *srcPf*	The original data format.
[in] *src*	A pointer to the original data.
[in] *tgtPf*	The target data format.
[out] *tgt*	Receives the converted texture mipmap level.
[in] *srcLineAlign*	Optional. The original scanline alignment in bytes. Defaults to 1.
[in] *tgtLineAlign*	Optional. The target scanline alignment in bytes. Defaults to 1.

Return Values

None

Description

Converts a texture mipmap level from one particular color format to another using a specific scanline alignment.

convertPalette

Converts palette data from one particular color format to another.

Definition

```
#include <pixel_format.h>
void convertPalette(
   uint32_t numEntries,
   PixelFormat const &srcPf,
   uint8_t const *& src,
   PixelFormat const &tgtPf,
   uint8_t *& tgt,
   size_t srcEntryAlign = 1,
   size_t tgtEntryAlign = 1
);
```

Arguments

[in] *numEntries*	The number of palette entries.
[in] *srcPf*	The original data format.
[in] *src*	A pointer to the original data.
[in] *tgtPf*	The target data format.
[out] *tgt*	Receives the converted data.
[in] *srcEntryAlign*	Optional. The original palette entry alignment in bytes. Defaults to 1.
[in] *tgtEntryAlign*	Optional. The target palette entry alignment in bytes. Defaults to 1.

Return Values

None

Description

Converts palette data from one particular color format to another.

convertPixel

Converts color data from one particular color format to another.

Definition

```
#include <pixel_format.h>
uint64_t convertPixel(
    uint64_t src,
    PixelFormat const &srcPf,
    PixelFormat const &tgtPf
);
```

Arguments

[in] *src*	The color data.
[in] *srcPf*	The original data format.
[in] *tgtPf*	The target data format.

Return Values

The converted data.

Description

Converts color data from one particular color format to another.

convertTexture

Converts a full texture in a particular color format to another using a specific cubemap face alignment.

Definition

```
#include <pixel_format.h>
void convertTexture(
   uint32_t width,
   uint32_t height,
   uint32_t numLevels,
   uint32_t numFaces,
   PixelFormat const &srcPf,
   uint8_t const *& src,
   PixelFormat const &tgtPf,
   uint8_t *& tgt,
   size_t srcLineAlign = 1,
   size_t srcLevelAlign = 1,
   size_t srcFaceAlign = 1,
   size_t tgtLineAlign = 1,
   size_t tgtLevelAlign = 1,
   size_t tgtFaceAlign = 1
);
```

Arguments

[in] *width*	The base mipmap level width.
[in] *height*	The base mipmap level height.
[in] *numLevels*	The number of mipmap levels.
[in] *numFaces*	The number of cubemap faces.
[in] *srcPf*	The original data format.
[in] *src*	A pointer to the original data.
[in] *tgtPf*	The target data format.
[out] *tgt*	Receives the converted full texture.
[in] *srcLineAlign*	Optional. The original scanline alignment in bytes. Defaults to 1.
[in] *srcLevelAlign*	Optional. The original mipmap level alignment in bytes. Defaults to 1.
[in] *srcFaceAlign*	Optional. The original face alignment in bytes. Defaults to 1.
[in] *tgtLineAlign*	Optional. The target scanline alignment in bytes. Defaults to 1.
[in] *tgtLevelAlign*	Optional. The target mipmap level alignment in bytes. Defaults to 1.
[in] *tgtFaceAlign*	Optional. The target face alignment in bytes. Defaults to 1.

Return Values

None

Description

Converts a full texture in a particular color format to another using a specific cubemap face alignment.

deSwizzleLevel

Converts swizzled data to a linear layout.

Definition

```
#include <swizzle.h>
void deSwizzleLevel(
   uint8_t *tgt,
   const uint8_t *src,
   uint32_t width,
   uint32_t height,
   uint32_t bpp
);
```

Arguments

[out] *tgt*	A pre-allocated buffer which receives the linear texture data.
[in] *src*	A pointer to the input swizzled texture data.
[in] *width*	The width of the texture.
[in] *height*	The height of the texture.
[in] *bpp*	The number of bits per pixel. For block compressed textures this should be the number of bits per block.

Return Values

None

Description

Converts swizzled data to a linear layout.

faceSize

Computes the size of a texture cubemap face using a specific mipmap level alignment.

Definition

```
#include <pixel_format.h>
size_t faceSize(
    uint32_t width,
    uint32_t height,
    uint32_t numLevels,
    size_t bitsPerPixel,
    size_t lineAlign = 1,
    size_t levelAlign = 1
);
```

Arguments

[in] *width*	The base mipmap level width.
[in] *height*	The base mipmap level height.
[in] *numLevels*	The number of mipmap levels.
[in] *bitsPerPixel*	The pixel size in bits.
[in] *lineAlign*	Optional. The scanline alignment in bytes. Defaults to 1.
[in] *levelAlign*	Optional. The mipmap level alignment in bytes. Defaults to 1.

Return Values

The cubemap face size in bytes.

Description

Computes the size of a texture cubemap face using a specific mipmap level alignment.

levelSize

Computes the size of a texture mipmap level using a specific scanline alignment.

Definition

```
#include <pixel_format.h>
inline size_t levelSize(
   uint32_t width,
   uint32_t height,
   size_t bitsPerPixel,
   size_t lineAlign = 1
);
```

Arguments

[in] *width*	The mipmap level width.
[in] *height*	The mipmap level height.
[in] *bitsPerPixel*	The pixel size in bits.
[in] *lineAlign*	Optional. The scanline alignment in bytes. Defaults to 1.

Return Values

The mipmap level size in bytes.

Description

Computes the size of a texture mipmap level using a specific scanline alignment.

operator==

A comparison operator for palettes.

Definition

```
#include <texture.h>
inline bool operator==(
   Palette const &pal0,
   Palette const &pal1
);
```

Arguments

[in] *pal0*	The first comparison term.
[in] *pal1*	The second comparison term.

Return Values

If both palettes are identical, true is returned; otherwise false is returned.

Description

A comparison operator for palettes.

paletteSize

Computes the size of a palette.

Definition

```
#include <pixel_format.h>
inline size_t paletteSize(
    PixelFormat const &pf,
    uint32_t numEntries
);
```

Arguments

[in] *pf*	The palette color format.
[in] *numEntries*	The number of palette entries.

Return Values

The palette size in bytes.

Description

Computes the size of a palette.

size

Computes the size (in bytes) of a multi-face texture.

Definition

```
#include <texture.h>
SceTextureErrorCode size(
    size_t &size,
    Encoding encoding,
    Layout layout,
    PixelFormat const &pixelFormat,
    uint32_t width,
    uint32_t height,
    int32_t numLevels = -1,
    uint32_t numFaces = 1
);
```

Arguments

[out] *size*	Receives the computed size.
[in] *encoding*	The encoding of the texture face's data.
[in] *layout*	The layout of the texture face's data.
[in] *pixelFormat*	The pixel format of the texture face's data.
[in] *width*	The width of the texture face's base level.
[in] *height*	The height of the texture face's base level.
[in] *numLevels*	Optional. The number of mip levels of the texture face. Defaults to -1, which indicates a full mip chain.
[in] *numFaces*	Optional. The number of texture faces. Defaults to 1.

Return Values

Value	Description
SCE_OK	The operation was successful.

Description

Computes the size (in bytes) of a multi-face texture.

swizzleLevel

Converts linear data to a swizzled layout.

Definition

```
#include <swizzle.h>
void swizzleLevel(
   uint8_t *tgt,
   const uint8_t *src,
   uint32_t width,
   uint32_t height,
   uint32_t bpp
);
```

Arguments

[out] *tgt*	A pre-allocated buffer which receives the swizzled texture data.
[in] *src*	A pointer to the input linear texture data.
[in] *width*	The width of the texture.
[in] *height*	The height of the texture.
[in] *bpp*	The number of bits per pixel. For block compressed textures this should be the number of bits per block.

Return Values

None

Description

Converts linear data to a swizzled layout.

textureSize

Computes the size of a full texture using a specific cubemap face alignment.

Definition

```
#include <pixel_format.h>
size_t textureSize(
   uint32_t width,
   uint32_t height,
   uint32_t numLevels,
   uint32_t numFaces,
   size_t bitsPerPixel,
   size_t lineAlign = 1,
   size_t levelAlign = 1,
   size_t faceAlign = 1
);
```

Arguments

[in] *width*	The base mipmap level width.
[in] *height*	The base mipmap level height.
[in] *numLevels*	The number of mipmap levels.
[in] *numFaces*	The number of cubemap faces.
[in] *bitsPerPixel*	The pixel size in bits.
[in] *lineAlign*	Optional. The scanline alignment in bytes. Defaults to 1.
[in] *levelAlign*	Optional. The mipmap level alignment in bytes. Defaults to 1.
[in] *faceAlign*	Optional. The cubemap face alignment in bytes. Defaults to 1.

Return Values

The texture size in bytes.

Description

Computes the size of a full texture using a specific cubemap face alignment.

unpackFace

Unpacks a palettized texture cubemap face using a given palette and a specific mipmap level alignment.

Definition

```
#include <pixel_format.h>
void unpackFace(
   uint32_t width,
   uint32_t height,
   uint32_t numLevels,
   size_t idxBits,
   uint8_t const *& inds,
   PixelFormat const &palPf,
   uint8_t const *pal,
   uint8_t *& tgt,
   size_t indsLineAlign = 1,
   size_t indsLevelAlign = 1,
   size_t tgtLineAlign = 1,
   size_t tgtLevelAlign = 1
);
```

Arguments

[in] *width*	The base mipmap level width.
[in] *height*	The base mipmap level height.
[in] *numLevels*	The number of mipmap levels.
[in] *idxBits*	The index size in bits.
[in] *inds*	The indexed data.
[in] *palPf*	The palette color format.
[in] *pal*	The palette data.
[out] *tgt*	Receives the unpacked palettized texture cubemap face.
[in] *indsLineAlign*	Optional. The indexed data scanline alignment in bytes. Defaults to 1.
[in] *indsLevelAlign*	Optional. The indexed data level alignment in bytes. Defaults to 1.
[in] *tgtLineAlign*	Optional. The target data scanline alignment in bytes. Defaults to 1.
[in] *tgtLevelAlign*	Optional. The target data level alignment in bytes. Defaults to 1.

Return Values

None

Description

Unpacks a palettized texture cubemap face using a given palette and a specific mipmap level alignment.

unpackLevel

Unpacks a palettized texture mipmap level using a given palette and a specific scanline alignment.

Definition

```
#include <pixel_format.h>
void unpackLevel(
   uint32_t width,
   uint32_t height,
   size_t idxBits,
   uint8_t const *& inds,
   PixelFormat const &palPf,
   uint8_t const *pal,
   uint8_t *& tgt,
   size_t indsLineAlign = 1,
   size_t tgtLineAlign = 1
);
```

Arguments

[in] *width*	The mipmap level width.
[in] *height*	The mipmap level height.
[in] *idxBits*	The index size in bits.
[in] *inds*	The indexed data.
[in] *palPf*	The palette color format.
[in] *pal*	The palette data.
[out] *tgt*	Receives the palettized texture mipmap level.
[in] *indsLineAlign*	Optional. The indexed data scanline alignment in bytes. Defaults to 1.
[in] *tgtLineAlign*	Optional. The target data scanline alignment in bytes. Defaults to 1.

Return Values

None

Description

Unpacks a palettized texture mipmap level using a given palette and a specific scanline alignment.

unpackTexture

Unpacks a full palettized texture using a given palette and a specific cubemap face alignment.

Definition

```
#include <pixel_format.h>
void unpackTexture(
   uint32_t width,
   uint32_t height,
   uint32_t numLevels,
   uint32_t numFaces,
   size_t idxBits,
   uint8_t const *& inds,
   PixelFormat const &palPf,
   uint8_t const *pal,
   uint8_t *& tgt,
   size_t indsLineAlign = 1,
   size_t indsLevelAlign = 1,
   size_t indsFaceAlign = 1,
   size_t tgtLineAlign = 1,
   size_t tgtLevelAlign = 1,
   size_t tgtFaceAlign = 1
);
```

Arguments

[in] *width*	The base mipmap level width.
[in] *height*	The base mipmap level height.
[in] *numLevels*	The number of mipmap levels.
[in] *numFaces*	The number of cubemap faces.
[in] *idxBits*	The index size in bits.
[in] *inds*	The indexed data.
[in] *palPf*	The palette color format.
[in] *pal*	The palette data.
[out] *tgt*	Receives the unpacked full palettized texture.
[in] *indsLineAlign*	Optional. The indexed data scanline alignment in bytes. Defaults to 1.
[in] *indsLevelAlign*	Optional. The indexed data level alignment in bytes. Defaults to 1.
[in] *indsFaceAlign*	Optional. The indexed data face alignment in bytes. Defaults to 1.
[in] *tgtLineAlign*	Optional. The target data scanline alignment in bytes. Defaults to 1.
[in] *tgtLevelAlign*	Optional. The target data level alignment in bytes. Defaults to 1.
[in] *tgtFaceAlign*	Optional. The target data face alignment in bytes. Defaults to 1.

Return Values

None

Description

Unpacks a full palettized texture using a given palette and a specific cubemap face alignment.

sce::Texture::Data Summary

sce::Texture::Data

The container class for a single texture.

Definition

```
#include <texture.h>
class Data {};
```

Description

The container class for a single texture.

Methods Summary

Methods	Description
build	A factory method used to create a texture instance which owns texture data.
build	A factory method used to create a texture instance which does not own texture data.
clone	Clones a texture duplicating any of the data it references.
Data	The constructor for the Data class.
~Data	The destructor for the Data class.
encoding	Gets the texture's data encoding.
faceSize	Gets the size of a single texture face.
getBorderColor	Gets the texture's border color.
getPalette	Gets the texture's palette.
getWndHeight	Gets the height of the texture's mapped window.
getWndWidth	Gets the width of the texture's mapped window.
hasBorder	Checks whether the texture uses a border color.
height	Gets the texture's height.
layout	Gets the texture's data layout.
numFaces	Gets the number of cubemap faces the texture has.
numLevels	Gets the number of mip levels in the texture.
pitch	Gets the texture's scanline size.
pixelFormat	Gets the texture's data pixel format.
raw	Gets a pointer to the raw texture's data.
release	Relinquishes ownership of the data that the texture references.
setBorderColor	Sets the texture's border color.
setPalette	Sets the texture's palette.
setWndHeight	Sets the height of the texture's mapped window.
setWndWidth	Sets the width of the texture's mapped window.
size	Gets the size of the entire texture.
swap	Exchanges properties and data ownership between two textures.
width	Gets the texture's width.

sce::Texture::Data Constructors and Destructors

Data

The constructor for the Data class.

Definition

```
#include <texture.h>
inline Data();
```

Arguments

None

Return Values

None

Description

The constructor for the Data class.

Notes

This is a dummy constructor. In order to initialize an instance of the Data class, use the factory method build().

~Data

The destructor for the Data class.

Definition

```
#include <texture.h>
~Data();
```

Arguments

None

Return Values

None

Description

The destructor for the Data class.

sce::Texture::Data Public Static Methods

build

A factory method used to create a texture instance which owns texture data.

Definition

```
#include <texture.h>
static SceTextureErrorCode build(
    Data &tgt,
    Encoding encoding,
    Layout layout,
    PixelFormat const &pixelFormat,
    uint32_t width,
    uint32_t height,
    uint32_t numLevels,
    uint32_t numFaces,
    float const *borderColor = NULL,
    uint32_t wndWidth = 0,
    uint32_t wndHeight = 0
);
```

Arguments

[out] `tgt`	Receives the texture instance built by this method.
[in] `encoding`	The texture's encoding.
[in] `layout`	The texture's layout.
[in] `pixelFormat`	The texture's pixel format descriptor. Use a blank instance when the format descriptor is not relevant to the encoding.
[in] `width`	The texture's width in pixels.
[in] `height`	The texture's height in pixels.
[in] `numLevels`	The number of mip levels in the texture.
[in] `numFaces`	The number of cubemap faces in the texture.
[in] `borderColor`	Optional. The texture's border color as a float array. Defaults to `NULL`, which indicates no border color is used.
[in] `wndWidth`	Optional. The texture's mapped window width in pixels. Defaults to 0, which indicates the entire texture's width.
[in] `wndHeight`	Optional. The texture's mapped window height in pixels. Defaults to 0, which indicates the entire texture's height.

Return Values

Value	Description
`SCE_OK`	The operation was successful.
`SCE_TEXTURE_ERROR_MEMORY_ALLOCATION_FAILURE`	The operation failed because memory could not be allocated for the texture.

Description

A factory method used to create a texture instance which owns texture data.

build

A factory method used to create a texture instance which does not own texture data.

Definition

```
#include <texture.h>
static SceTextureErrorCode build(
    Data &tgt,
    Encoding encoding,
    Layout layout,
    PixelFormat const &pixelFormat,
    uint32_t width,
    uint32_t height,
    uint32_t numLevels,
    uint32_t numFaces,
    uint8_t *data,
    float const *borderColor = NULL,
    Palette const *palette = NULL,
    uint32_t wndWidth = 0,
    uint32_t wndHeight = 0
);
```

Arguments

[out] *tgt*	Receives the texture instance built by this method.
[in] *encoding*	The texture's encoding.
[in] *layout*	The texture's layout.
[in] *pixelFormat*	The texture's pixel format descriptor. Use a blank instance when the format descriptor is not relevant to the encoding.
[in] *width*	The texture's width in pixels.
[in] *height*	The texture's height in pixels.
[in] *numLevels*	The number of mip levels in the texture.
[in] *numFaces*	The number of cubemap faces in the texture.
[in] *data*	A pointer to the texture's raw data. This is not deeply-copied and the caller is responsible for lifetime management.
[in] *borderColor*	Optional. The texture's border color as a float array. Defaults to `NULL`, which indicates no border color is used.
[in] *palette*	Optional. A pointer to the desired texture's palette. This is not deeply-copied and the caller is responsible for lifetime management. Defaults to `NULL`.
[in] *wndWidth*	Optional. The texture's mapped window width in pixels. Defaults to 0, which indicates the entire texture's width.
[in] *wndHeight*	Optional. The texture's mapped window height in pixels. Defaults to 0, which indicates the entire texture's height.

Return Values

Value	Description
`SCE_OK`	The operation was successful.

Description

A factory method used to create a texture instance which does not own texture data.

clone

Clones a texture duplicating any of the data it references.

Definition

```
#include <texture.h>
static SceTextureErrorCode clone(
    Data &tgt,
    Data const &src
);
```

Arguments

[out] *tgt*	Receives the copy of the data.
[in] *src*	A pointer to the source texture.

Return Values

Value	Description
`SCE_OK`	The operation was successful.
`SCE_TEXTURE_ERROR_MEMORY_ALLOCATION_FAILURE`	The operation failed because memory could not be allocated for the target texture.

Description

Clones a texture duplicating any of the data it references.

sce::Texture::Data Public Instance Methods

encoding

Gets the texture's data encoding.

Definition

```
#include <texture.h>
inline Encoding encoding() const;
```

Arguments

None

Return Values

The texture's encoding.

Description

Gets the texture's data encoding.

faceSize

Gets the size of a single texture face.

Definition

```
#include <texture.h>
inline size_t faceSize() const;
```

Arguments

None

Return Values

The texture's face size in bytes.

Description

Gets the size of a single texture face.

getBorderColor

Gets the texture’s border color.

Definition

```
#include <texture.h>
float const *getBorderColor() const;
```

Arguments

None

Return Values

The texture’s border color as a float array.

Description

Gets the texture’s border color.

getPalette

Gets the texture’s palette.

Definition

```
#include <texture.h>
Palette const *getPalette() const;
```

Arguments

None

Return Values

A pointer to the texture’s palette. `NULL` if returned if the texture has no palette.

Description

Gets the texture’s palette.

getWndHeight

Gets the height of the texture’s mapped window.

Definition

```
#include <texture.h>
inline uint32_t getWndHeight() const;
```

Arguments

None

Return Values

The height of the texture’s mapped window in pixels.

Description

Gets the height of the texture’s mapped window.

getWndWidth

Gets the width of the texture’s mapped window.

Definition

```
#include <texture.h>
inline uint32_t getWndWidth() const;
```

Arguments

None

Return Values

The width of the texture’s mapped window in pixels.

Description

Gets the width of the texture’s mapped window.

hasBorder

Checks whether the texture uses a border color.

Definition

```
#include <texture.h>
inline bool hasBorder() const;
```

Arguments

None

Return Values

If the texture is using a border color, true is returned; otherwise false is returned.

Description

Checks whether the texture uses a border color.

height

Gets the texture’s height.

Definition

```
#include <texture.h>
inline uint32_t height() const;
```

Arguments

None

Return Values

The texture’s height in pixels.

Description

Gets the texture’s height.

layout

Gets the texture's data layout.

Definition

```
#include <texture.h>
inline Layout layout() const;
```

Arguments

None

Return Values

The texture's layout.

Description

Gets the texture's data layout.

numFaces

Gets the number of cubemap faces the texture has.

Definition

```
#include <texture.h>
inline uint32_t numFaces() const;
```

Arguments

None

Return Values

The number of cubemap faces the texture has.

Description

Gets the number of cubemap faces the texture has.

numLevels

Gets the number of mip levels in the texture.

Definition

```
#include <texture.h>
inline uint32_t numLevels() const;
```

Arguments

None

Return Values

The number of mip levels in the texture.

Description

Gets the number of mip levels in the texture.

pitch

Gets the texture’s scanline size.

Definition

```
#include <texture.h>
inline size_t pitch() const;
```

Arguments

None

Return Values

The texture’s scanline size in bytes.

Description

Gets the texture’s scanline size.

pixelFormat

Gets the texture’s data pixel format.

Definition

```
#include <texture.h>
inline PixelFormat const &pixelFormat() const;
```

Arguments

None

Return Values

The texture’s pixel format.

Description

Gets the texture’s data pixel format.

raw

Gets a pointer to the raw texture's data.

Definition

```
#include <texture.h>
inline uint8_t *raw() const;
```

Arguments

None

Return Values

A pointer to the raw texture's data.

Description

Gets a pointer to the raw texture's data.

release

Relinquishes ownership of the data that the texture references.

Definition

```
#include <texture.h>
inline uint8_t *release();
```

Arguments

None

Return Values

The raw data for the texture.

Description

Relinquishes ownership of the data that the texture references.

setBorderColor

Sets the texture’s border color.

Definition

```
#include <texture.h>
void setBorderColor(
    float const *borderColor
);
```

Arguments

[in] *borderColor*	The border color as a float array. The number of components depends on the texture format.

Return Values

None

Description

Sets the texture’s border color.

setPalette

Sets the texture’s palette.

Definition

```
#include <texture.h>
void setPalette(
    Palette const *pal
);
```

Arguments

[in] `pal`	A pointer to the palette to set. Note that this is not deeply-copied and the caller is responsible for lifetime management.

Return Values

None

Description

Sets the texture’s palette.

setWndHeight

Sets the height of the texture's mapped window.

Definition

```
#include <texture.h>
SceTextureErrorCode setWndHeight(
   uint32_t wndHeight
);
```

Arguments

[in] *wndHeight*	The height of the window in pixels. A value of 0 indicates the entire texture's width.

Return Values

Value	Description
SCE_OK	The operation was successful.
SCE_TEXTURE_ERROR_INVALID_INPUT_DIMENSIONS	The input height was invalid.

Description

Sets the height of the texture's mapped window.

setWndWidth

Sets the width of the texture’s mapped window.

Definition

```
#include <texture.h>
SceTextureErrorCode setWndWidth(
    uint32_t wndWidth
);
```

Arguments

[in] `wndWidth`	The width of the window in pixels. A value of 0 indicates the entire texture’s width.

Return Values

Value	Description
`SCE_OK`	The operation was successful.
`SCE_TEXTURE_ERROR_INVALID_INPUT_DIMENSIONS`	The operation failed because the input width was invalid.

Description

Sets the width of the texture’s mapped window.

size

Gets the size of the entire texture.

Definition

```
#include <texture.h>
inline size_t size() const;
```

Arguments

None

Return Values

The texture's size in bytes.

Description

Gets the size of the entire texture.

swap

Exchanges properties and data ownership between two textures.

Definition

```
#include <texture.h>
Data &swap(
    Data &alt
);
```

Arguments

[in,out] *alt*	The texture instance with which to swap data.

Return Values

A reference to self once the data is swapped.

Description

Exchanges properties and data ownership between two textures.

width

Gets the texture’s width.

Definition

```
#include <texture.h>
inline uint32_t width() const;
```

Arguments

None

Return Values

The texture’s width in pixels.

Description

Gets the texture’s width.

sce::Texture::DDS Summary

sce::Texture::DDS

A namespace containing DDS functionality.

Definition

```
namespace DDS {}
```

Description

A namespace containing DDS functionality.

Function Summary

Function	Description
deserialize	Deserializes a DDS format texture.
isValidFileInput	Performs a simple check to determine whether a file is in the DDS format.
isValidFileOutput	Queries whether an output path points to a DDS file.
isValidInput	Performs a simple check to determine whether a stream of linear data is in the DDS format.
linearSize	Computes the linear size of a texture (in bytes) when stored in the DDS format.
load	Loads a texture from a file in the DDS format.
save	Saves a texture as a DDS file.
serialize	Serializes a DDS format texture.

sce::Texture::DDS Functions

deserialize

Deserializes a DDS format texture.

Definition

```
#include <dds_io.h>
SceTextureErrorCode deserialize(
    Texture::File &tgt,
    uint8_t const *srcBuffer,
    size_t srcSize
);
```

Arguments

[out] *tgt*	A texture file object in which the result of the operation will be stored (with an index of 0).
[in] *srcBuffer*	A pointer to the linear source data, which should be in the DDS format.
[in] *srcSize*	The size of the linear source data.

Return Values

Value	Description
`SCE_OK`	The operation was successful.
`SCE_TEXTURE_ERROR_INVALID_DDS_DATA`	The operation failed because the linear source data is either corrupted or is not in the DDS format.
`SCE_TEXTURE_ERROR_MEMORY_ALLOCATION_FAILURE`	The operation failed because an allocation error arose when creating the resulting texture.
`SCE_TEXTURE_ERROR_UNSUPPORTED_INPUT_FORMAT`	The operation failed because the data is encoded in an unsupported format.

Description

Deserializes a DDS format texture.

isValidFileInput

Performs a simple check to determine whether a file is in the DDS format.

Definition

```
#include <dds_io.h>
SceTextureErrorCode isValidFileInput(
    bool &valid,
    char const *fileName
);
```

Arguments

[out] *valid*	Receives whether the file is in the DDS format or not.
[in] *fileName*	The path to the input file.

Return Values

Value	Description
`SCE_OK`	The operation was successful.
`SCE_TEXTURE_ERROR_CANNOT_OPEN_FILE`	The operation failed because the file failed to open.
`SCE_TEXTURE_ERROR_MEMORY_ALLOCATION_FAILURE`	The operation failed because it could not allocate the internal data needed to perform the query.

Description

Performs a simple check to determine whether a file is in the DDS format. However, the file may be malformed, and this can only be detected by loading it.

isValidFileOutput

Queries whether an output path points to a DDS file.

Definition

```
#include <dds_io.h>
SceTextureErrorCode isValidFileOutput(
    bool &valid,
    char const *fileName
);
```

Arguments

[out] *valid*	Receives whether the output path points to a DDS file.
[in] *fileName*	The path to the output file.

Return Values

Value	Description
SCE_OK	The operation was successful.

Description

Queries whether an output path points to a DDS file. This function simply checks whether an output path includes the .DDS extension.

isValidInput

Performs a simple check to determine whether a stream of linear data is in the DDS format.

Definition

```
#include <dds_io.h>
bool isValidInput(
    uint8_t const *srcBuffer,
    size_t srcSize
);
```

Arguments

[in] *srcBuffer*	A pointer to the linear data to check.
[in] *srcSize*	The size of the linear data.

Return Values

Value	Description
`true`	The input is in the DDS format.
`false`	The input is not in the DDS format.

Description

Performs a simple check to determine whether a stream of linear data is in the DDS format. However, the data may be malformed, and this can only be detected by deserializing it.

linearSize

Computes the linear size of a texture (in bytes) when stored in the DDS format.

Definition

```
#include <dds_io.h>
SceTextureErrorCode linearSize(
    size_t &size,
    Texture::File const &src,
    uint32_t textureIndex = 0
);
```

Arguments

[out] *size*	Receives the computed size.
[in] *src*	A texture file containing the texture whose size will be computed.
[in] *textureIndex*	Optional. The index of the texture within the input texture file. Defaults to 0.

Return Values

Value	Description
SCE_OK	The operation was successful.
SCE_TEXTURE_ERROR_INVALID_VALUE	The operation failed because the supplied index was out of range.

Description

Computes the linear size of a texture (in bytes) when stored in the DDS format.

load

Loads a texture from a file in the DDS format.

Definition

```
#include <dds_io.h>
SceTextureErrorCode load(
    Texture::File &tgt,
    char const *fileName
);
```

Arguments

[out] *tgt*	A texture file object in which the result of the operation will be stored (with an index of 0).
[in] *fileName*	The path to the input file.

Return Values

Value	Description
SCE_OK	The operation was successful.
SCE_TEXTURE_ERROR_CANNOT_OPEN_FILE	The operation failed because the input file failed to open.
SCE_TEXTURE_ERROR_INVALID_DDS_DATA	The operation failed because the input file is either corrupted or is not in the DDS format.
SCE_TEXTURE_ERROR_MEMORY_ALLOCATION_FAILURE	The operation failed because an allocation error arose when creating the resulting texture.
SCE_TEXTURE_ERROR_UNSUPPORTED_INPUT_FORMAT	The operation failed because the input file is encoded in an unsupported format.

Description

Loads a texture from a file in the DDS format.

save

Saves a texture as a DDS file.

Definition

```
#include <dds_io.h>
SceTextureErrorCode save(
    char const *fileName,
    Texture::File const &src,
    uint32_t textureIndex = 0
);
```

Arguments

[in] *fileName*	The path to the DDS file in which the operation stores the texture.
[in] *src*	The texture file object containing the texture to be saved.
[in] *textureIndex*	Optional. The index of the specific texture within the file to save. Default is 0.

Return Values

Value	Description
SCE_OK	The operation was successful.
SCE_TEXTURE_ERROR_CANNOT_OPEN_FILE	The operation failed because the output file failed to open.
SCE_TEXTURE_ERROR_MEMORY_ALLOCATION_FAILURE	The operation failed because it could not allocate the internal data needed to perform the save.

Description

Saves a texture as a DDS file.

serialize

Serializes a DDS format texture.

Definition

```
#include <dds_io.h>
SceTextureErrorCode serialize(
    uint8_t *tgtBuffer,
    size_t tgtSize,
    Texture::File const &src,
    uint32_t textureIndex = 0
);
```

Arguments

[out] *tgtBuffer*	Receives the linear data.
[in] *tgtSize*	The size of the buffer that will receive the linear data.
[in] *src*	A texture file object containing the texture to be serialized.
[in] *textureIndex*	The index of the texture within the texture file to serialize. The default is 0.

Return Values

Value	Description
SCE_OK	The operation was successful.
SCE_TEXTURE_ERROR_BUFFER_WRITE_FAILURE	The operation failed because the size of the storage buffer is not sufficient to store this texture.
SCE_TEXTURE_ERROR_INVALID_VALUE	The operation failed because the supplied index was out of range.

Description

Serializes a DDS format texture.

sce::Texture::Face Summary

sce::Texture::Face

A namespace containing functionality for a single texture face.

Definition

```
namespace Face {}
```

Description

A namespace containing functionality for a single texture face.

Function Summary

Function	Description
`size`	Computes the size (in bytes) of a single texture face.

sce::Texture::Face Functions

size

Computes the size (in bytes) of a single texture face.

Definition

```
#include <texture.h>
SceTextureErrorCode size(
    size_t &size,
    Encoding encoding,
    Layout layout,
    PixelFormat const &pixelFormat,
    uint32_t width,
    uint32_t height,
    int32_t numLevels = -1
);
```

Arguments

[out] *size*	Receives the computed size.
[in] *encoding*	The encoding of the texture face's data.
[in] *layout*	The layout of the texture face's data.
[in] *pixelFormat*	The pixel format of the texture face's data.
[in] *width*	The width of the texture face's base level.
[in] *height*	The height of the texture face's base level.
[in] *numLevels*	Optional. The number of mip levels which the texture face has. Defaults to -1, which indicates a full mip chain.

Return Values

Value	Description
SCE_OK	The operation was successful.

Description

Computes the size (in bytes) of a single texture face.

sce::Texture::File Summary

sce::Texture::File

A container class for a collection of textures and palettes.

Definition

```
#include <texture.h>
class File {};
```

Description

A container class for a collection of textures and palettes.

Methods Summary

Methods	Description
build	A factory method used to create a texture file instance.
File	The constructor for the File class.
~File	The destructor for the File class.
getPalette	Gets a given palette in this file.
getTexture	Gets a given texture in this file.
index	Retrieves the index for a given palette in this file.
numPalettes	Gets the number of palettes in the file.
numTextures	Gets the number of textures in the file.
setPalette	Sets a given palette in this file.
setTexture	Sets a given texture in this file.
swap	Exchanges properties and data ownership between two files.

sce::Texture::File Constructors and Destructors

File

The constructor for the `File` class.

Definition

```
#include <texture.h>
inline File();
```

Arguments

None

Return Values

None

Description

The constructor for the `File` class.

Notes

This is a dummy constructor. In order to initialize an instance of the `File` class, use the factory method `build()`.

~File

The destructor for the File class.

Definition

```
#include <texture.h>
~File();
```

Arguments

None

Return Values

None

Description

The destructor for the File class.

sce::Texture::File Public Static Methods

build

A factory method used to create a texture file instance.

Definition

```
#include <texture.h>
static SceTextureErrorCode build(
   File &tgt,
   uint32_t numTextures,
   uint32_t numPalettes = 0
);
```

Arguments

[out] *tgt*	Receives the texture file instance built by this method.
[in] *numTextures*	The number of textures in the file.
[in] *numPalettes*	Optional. The number of palettes in the file. Defaults to 0.

Return Values

Value	Description
SCE_OK	The operation was successful.
SCE_TEXTURE_ERROR_MEMORY_ALLOCATION_FAILURE	The operation failed because the memory could not be allocated for the file.

Description

A factory method used to create a texture file instance.

sce::Texture::File Public Instance Methods

getPalette

Gets a given palette in this file.

Definition

```
#include <texture.h>
Palette const &getPalette(
    uint32_t idx
) const;
```

Arguments

[in] *idx*	The index of the desired palette.

Return Values

A reference to the desired palette.

Description

Gets a given palette in this file.

getTexture

Gets a given texture in this file.

Definition

```
#include <texture.h>
Data const &getTexture(
    uint32_t idx
) const;
```

Arguments

[in] *idx*	The index of the desired texture.

Return Values

A reference to the desired texture.

Description

Gets a given texture in this file.

index

Retrieves the index for a given palette in this file.

Definition

```
#include <texture.h>
int32_t index(
    Palette const *pal
) const;
```

Arguments

[in] *pal*	The palette to retrieve the index for.

Return Values

The index of the specified palette. -1 is returned if the specified palette is not present.

Description

Retrieves the index for a given palette in this file.

numPalettes

Gets the number of palettes in the file.

Definition

```
#include <texture.h>
inline uint32_t numPalettes() const;
```

Arguments

None

Return Values

The number of palettes in the file.

Description

Gets the number of palettes in the file.

numTextures

Gets the number of textures in the file.

Definition

```
#include <texture.h>
inline uint32_t numTextures() const;
```

Arguments

None

Return Values

The number of textures in the file.

Description

Gets the number of textures in the file.

setPalette

Sets a given palette in this file.

Definition

```
#include <texture.h>
SceTextureErrorCode setPalette(
    uint32_t idx,
    Palette const &pal
);
```

Arguments

[in] *idx*	The index under which the palette will be stored.
[in] *pal*	The source palette. Note this will be deep copied.

Return Values

Value	Description
`SCE_OK`	The operation was successful.
`SCE_TEXTURE_ERROR_MEMORY_ALLOCATION_FAILURE`	The operation failed because memory could not be allocated for the palette.

Description

Sets a given palette in this file.

setTexture

Sets a given texture in this file.

Definition

```
#include <texture.h>
SceTextureErrorCode setTexture(
    uint32_t idx,
    Data const &tex
);
```

Arguments

[in] *idx*	The index under which the texture will be stored.
[in] *tex*	The source texture. Note this will be deep copied.

Return Values

Value	Description
`SCE_OK`	The operation was successful.
`SCE_TEXTURE_ERROR_MEMORY_ALLOCATION_FAILURE`	The operation failed because memory could not be allocated for the texture.

Description

Sets a given texture in this file.

swap

Exchanges properties and data ownership between two files.

Definition

```
#include <texture.h>
File &swap(
    File &alt
);
```

Arguments

[in,out] `alt`	The file instance with which to swap data.

Return Values

A reference to self once the data is swapped.

Description

Exchanges properties and data ownership between two files.

sce::Texture::Gxt Summary

sce::Texture::Gxt

A namespace containing GXT functionality.

Definition

```
namespace Gxt {}
```

Description

A namespace containing GXT functionality.

Variables

Public Variables

`uint32_t const` *`Magic`*	A four byte code used to identify a file in GXT format.

Function Summary

Function	Description
`convertTextureData`	Converts compact (non-padded) texture data to GXT runtime ready layout.
`deserialize`	Deserializes a GXT format texture.
`findNamedFormat`	Finds the target texture format corresponding to a specific name.
`getBaseFormat`	Gets the texture base format.
`getBorderDataSize`	Gets the size of the border data.
`getBpp`	Gets the bits per pixel for this texture format.
`getNumSupportedFormats`	Gets the number of supported target texture formats.
`getRevertedTextureDataSize`	Gets the size of texture data that has a compact (non-padded) layout.
`getSupportedFormat`	Obtains information about a supported target texture format.
`getTextureDataSize`	Gets the size of texture data that has GXT runtime ready layout.
`isBlockCompressed`	Checks if the texture format is a compressed format (UBC or PVRT).
`isIndexed`	Checks if the texture format is an indexed (palettized) format.
`isPvr`	Checks if the texture format is a PVRT compressed format.
`isUbc`	Checks if the texture format is a UBC compressed format.
`isValidInput`	Performs a simple check to determine whether a stream of linear data is in the GXT format.
`linearSize`	Computes the linear size of a texture (in bytes) when stored in the GXT format.
`load`	Loads a texture from a file in the GXT format.
`nameBaseFormat`	Gets the name of the texture base format.
`nameFormat`	Names a target texture format.
`revertTextureData`	Reverts texture data from GXT runtime ready layout to a compact (non-padded) layout.
`save`	Saves a texture as a GXT file.
`serialize`	Serializes a GXT format texture.
`supportsBorderData`	Checks if the texture format supports border data.

sce::Texture::Gxt Type Definitions

Header

Represents a GXT file header.

Definition

```
#include <gxt_io.h>
typedef struct sce::Texture::Gxt::Header {
    uint32_t m_magic;
    uint32_t m_version;
    uint32_t m_numTextures;
    uint32_t m_dataOffset;
    uint32_t m_dataSize;
    uint32_t m_numPalettes16;
    uint32_t m_numPalettes256;
    uint32_t m_pad;
} Header;
```

Members

m_magic	The GXT identifier.
m_version	The GXT version number.
m_numTextures	The number of textures.
m_dataOffset	The offset to the texture data.
m_dataSize	The total size of texture data.
m_numPalettes16	The number of 16 entry palettes.
m_numPalettes256	The number of 256 entry palettes.
m_pad	Padding.

Description

Represents a GXT file header.

SceGxtTextureInfo

Represents a GXT texture header.

Definition

```
#include <gxt_io.h>
typedef struct sce::Texture::Gxt::TextureInfo {
   uint32_t m_dataOffset;
   uint32_t m_dataSize;
   uint32_t m_paletteIndex;
   uint32_t m_flags;
   uint32_t m_type;
   uint32_t m_format;
   uint16_t m_width;
   uint16_t m_height;
   uint8_t m_mipCount;
   uint8_t m_pad[3];
} SceGxtTextureInfo;
```

Members

m_dataOffset	The offset to the texture data.
m_dataSize	The size of the texture data.
m_paletteIndex	The index of the palette.
m_flags	Flags.
m_type	The texture type (`SceGxmTextureType`).
m_format	The texture format (`SceGxmTextureFormat`).
m_width	The texture width.
m_height	The texture height.
m_mipCount	The number of mipmaps.
m_pad	Padding.

Description

Represents a GXT texture header.

sce::Texture::Gxt Functions

convertTextureData

Converts compact (non-padded) texture data to GXT runtime ready layout.

Definition

```
#include <gxt_util.h>
SceTextureErrorCode convertTextureData(
   uint8_t *tgt,
   const uint8_t *src,
   uint32_t width,
   uint32_t height,
   uint32_t numLevels,
   uint32_t numFaces,
   SceGxmTextureFormat format,
   SceGxmTextureType type,
   bool swizzle
);
```

Arguments

[out] `tgt`	Receives the GXT runtime ready data.
[in] `src`	A pointer to the input texture data.
[in] `width`	The width of the texture.
[in] `height`	The height of the texture.
[in] `numLevels`	The number of mip levels in the texture (including the top level).
[in] `numFaces`	The number of faces in the texture.
[in] `format`	The texture format.
[in] `type`	The texture type.
[in] `swizzle`	A flag that indicates whether the texture data will get swizzled.

Return Values

Value	Description
`SCE_OK`	The operation was completed successfully.

Description

Converts compact (non-padded) texture data to GXT runtime ready layout.

deserialize

Deserializes a GXT format texture.

Definition

```
#include <gxt_io.h>
SceTextureErrorCode deserialize(
    Texture::File &tgt,
    uint8_t const *srcBuffer,
    size_t srcSize
);
```

Arguments

[out] *tgt*	A texture file object in which the result of the operation will be stored (with an index of 0).
[in] *srcBuffer*	A pointer to the linear source data, which should be in the GXT format.
[in] *srcSize*	The size of the linear source data.

Return Values

Value	Description
`SCE_OK`	The operation was successful.
`SCE_TEXTURE_ERROR_INVALID_GXT_DATA`	The operation failed because the linear source data is either corrupted or is not in the GXT format.
`SCE_TEXTURE_ERROR_MEMORY_ALLOCATION_FAILURE`	The operation failed because an allocation error arose when creating the resulting texture.

Description

Deserializes a GXT format texture.

findNamedFormat

Finds the target texture format corresponding to a specific name.

Definition

```
#include <gxt_io.h>
SceTextureErrorCode findNamedFormat(
    SceGxmTextureFormat &texFormat,
    char const *name
);
```

Arguments

[out] *texFormat*	Receives the format named in *name*.
[in] *name*	A pointer to an ASCII string containing the name of the format sought.

Return Values

Value	Description
`SCE_OK`	The operation was successful.
`SCE_TEXTURE_ERROR_UNSUPPORTED_OUTPUT_FORMAT`	The operation failed because the name was invalid for a target texture format.

Description

Finds the target texture format corresponding to a specific name.

getBaseFormat

Gets the texture base format.

Definition

```
#include <gxt_util.h>
SceGxmTextureBaseFormat getBaseFormat(
    SceGxmTextureFormat format
);
```

Arguments

[in] *format*	The texture format.

Return Values

The base format corresponding to specified texture format.

Description

Gets the texture base format.

getBorderDataSize

Gets the size of the border data.

Definition

```
#include <gxt_util.h>
uint32_t getBorderDataSize(
    uint32_t width,
    uint32_t height,
    SceGxmTextureFormat format
);
```

Arguments

[in] *width*	The width of the texture.
[in] *height*	The height of the texture.
[in] *format*	The texture format.

Return Values

The size of the border data in bytes.

Description

Gets the size of the border data.

getBpp

Gets the bits per pixel for this texture format.

Definition

```
#include <gxt_util.h>
uint32_t getBpp(
    SceGxmTextureFormat format
);
```

Arguments

[in] *format*	The texture format.

Return Values

The bits per pixel.

Description

Gets the bits per pixel for this texture format.

getNumSupportedFormats

Gets the number of supported target texture formats.

Definition

```
#include <gxt_io.h>
uint32_t getNumSupportedFormats();
```

Arguments

None

Return Values

The number of supported target formats.

Description

Gets the number of supported target texture formats.

getRevertedTextureDataSize

Gets the size of texture data that has a compact (non-padded) layout.

Definition

```
#include <gxt_util.h>
SceTextureErrorCode getRevertedTextureDataSize(
    size_t *size,
    uint32_t width,
    uint32_t height,
    uint32_t numLevels,
    uint32_t numFaces,
    SceGxmTextureFormat format,
    SceGxmTextureType type
);
```

Arguments

[out] *size*	Receives the size in bytes of the texture data.
[in] *width*	The width of the texture.
[in] *height*	The height of the texture.
[in] *numLevels*	The number of mip levels in the texture (including the top level).
[in] *numFaces*	The number of faces in the texture.
[in] *format*	The texture format.
[in] *type*	The texture type.

Return Values

Value	Description
SCE_OK	The operation was completed successfully.

Description

Gets the size of texture data that has a compact (non-padded) layout.

getSupportedFormat

Obtains information about a supported target texture format.

Definition

```
#include <gxt_io.h>
SceTextureErrorCode getSupportedFormat(
    SceGxmTextureFormat &texFormat,
    char const *& name,
    sce::Texture::Encoding &encoding,
    sce::Texture::PixelFormat &pixelFormat,
    uint32_t idx
);
```

Arguments

[out] *texFormat*	Receives the target texture format.
[out] *name*	Receives a pointer to an ASCII string naming the format.
[out] *encoding*	Receives the encoding corresponding to the format.
[out] *pixelFormat*	Receives the pixel format corresponding to the format.
[in] *idx*	The index of the supported target texture format sought.

Return Values

Value	Description
SCE_OK	The operation was successful.
SCE_TEXTURE_ERROR_UNSUPPORTED_OUTPUT_FORMAT	The operation failed because the index was invalid and does not refer to a supported target texture format.

Description

Obtains information about a supported target texture format.

getTextureDataSize

Gets the size of texture data that has GXT runtime ready layout.

Definition

```
#include <gxt_util.h>
SceTextureErrorCode getTextureDataSize(
    size_t *size,
    uint32_t width,
    uint32_t height,
    uint32_t numLevels,
    uint32_t numFaces,
    SceGxmTextureFormat format,
    SceGxmTextureType type
);
```

Arguments

[out] *size*	Receives the size in bytes of the texture data.
[in] *width*	The width of the texture.
[in] *height*	The height of the texture.
[in] *numLevels*	The number of mip levels in the texture (including the top level).
[in] *numFaces*	The number of faces in the texture.
[in] *format*	The texture format.
[in] *type*	The texture type.

Return Values

Value	Description
`SCE_OK`	The operation was completed successfully.

Description

Gets the size of texture data that has GXT runtime ready layout.

isBlockCompressed

Checks if the texture format is a compressed format (UBC or PVRT).

Definition

```
#include <gxt_util.h>
bool isBlockCompressed(
    SceGxmTextureFormat format
);
```

Arguments

[in] *format*	The texture format.

Return Values

If the format is a compressed format, true is returned; otherwise false is returned.

Description

Checks if the texture format is a compressed format (UBC or PVRT).

isIndexed

Checks if the texture format is an indexed (palettized) format.

Definition

```
#include <gxt_util.h>
bool isIndexed(
   SceGxmTextureFormat format
);
```

Arguments

[in] *format*	The texture format.

Return Values

If the format is an indexed (palettized) format, true is returned; otherwise false is returned.

Description

Checks if the texture format is an indexed (palettized) format.

isPvr

Checks if the texture format is a PVRT compressed format.

Definition

```
#include <gxt_util.h>
bool isPvr(
    SceGxmTextureFormat format
);
```

Arguments

[in] *format*	The texture format.

Return Values

If the format is a PVRT compressed format, true is returned; otherwise false is returned.

Description

Checks if the texture format is a PVRT compressed format.

isUbc

Checks if the texture format is a UBC compressed format.

Definition

```
#include <gxt_util.h>
bool isUbc(
    SceGxmTextureFormat format
);
```

Arguments

[in] *format*	The texture format.

Return Values

If the format is a UBC compressed format, true is returned; otherwise false is returned.

Description

Checks if the texture format is a UBC compressed format.

isValidInput

Performs a simple check to determine whether a stream of linear data is in the GXT format.

Definition

```
#include <gxt_io.h>
bool isValidInput(
   uint8_t const *srcBuffer,
   size_t srcSize
);
```

Arguments

[in] *srcBuffer*	A pointer to the linear data to check.
[in] *srcSize*	The size of the linear data.

Return Values

Value	Description
`true`	The input is in the GXT format.
`false`	The input is not in the GXT format.

Description

Performs a simple check to determine whether a stream of linear data is in the GXT format. However, the data may be malformed, and this can only be detected by deserializing it.

linearSize

Computes the linear size of a texture (in bytes) when stored in the GXT format.

Definition

```
#include <gxt_io.h>
SceTextureErrorCode linearSize(
    size_t &size,
    Texture::File const &src,
    bool const *swizzleFlags = NULL
);
```

Arguments

[out] *size*	Receives the computed size.
[in] *src*	A texture file containing the texture whose size will be computed.
[in] *swizzleFlags*	Optional. An array of flags which indicate whether each of the textures in *src* should be swizzled. Defaults to NULL.

Return Values

Value	Description
SCE_OK	The operation was successful.
SCE_TEXTURE_ERROR_INVALID_VALUE	The operation failed because the supplied index was out of range.

Description

Computes the linear size of a texture (in bytes) when stored in the GXT format.

load

Loads a texture from a file in the GXT format.

Definition

```
#include <gxt_io.h>
SceTextureErrorCode load(
    Texture::File &tgt,
    char const *fileName
);
```

Arguments

[out] *tgt*	A texture file object in which the result of the operation will be stored (with an index of 0).
[in] *fileName*	The path to the input file.

Return Values

Value	Description
`SCE_OK`	The operation was successful.
`SCE_TEXTURE_ERROR_CANNOT_OPEN_FILE`	The operation failed because the input file failed to open.
`SCE_TEXTURE_ERROR_INVALID_GXT_DATA`	The operation failed because the input file is either corrupted or is not in the GXT format.
`SCE_TEXTURE_ERROR_MEMORY_ALLOCATION_FAILURE`	The operation failed because an allocation error arose when creating the resulting texture.

Description

Loads a texture from a file in the GXT format.

nameBaseFormat

Gets the name of the texture base format.

Definition

```
#include <gxt_util.h>
char const *nameBaseFormat(
    SceGxmTextureBaseFormat format
);
```

Arguments

[in] *format*	The texture base format.

Return Values

The name of the base format.

Description

Gets the name of the texture base format.

nameFormat

Names a target texture format.

Definition

```
#include <gxt_io.h>
SceTextureErrorCode nameFormat(
    char const *& name,
    SceGxmTextureFormat texFormat
);
```

Arguments

[in] *name*	A pointer to an ASCII string naming the input format.
[in] *texFormat*	The target texture format to be named.

Return Values

Value	Description
SCE_OK	The operation was successful.
SCE_TEXTURE_ERROR_UNSUPPORTED_OUTPUT_FORMAT	The operation failed because of an invalid/unknown target texture format.

Description

Names a target texture format.

revertTextureData

Reverts texture data from GXT runtime ready layout to a compact (non-padded) layout.

Definition

```
#include <gxt_util.h>
SceTextureErrorCode revertTextureData(
    uint8_t *tgt,
    const uint8_t *src,
    uint32_t width,
    uint32_t height,
    uint32_t numLevels,
    uint32_t numFaces,
    SceGxmTextureFormat format,
    SceGxmTextureType type,
    bool deSwizzle
);
```

Arguments

[out] *tgt*	Receives the compact texture data.
[in] *src*	A pointer to the texture data in GXT runtime ready layout.
[in] *width*	The width of the texture.
[in] *height*	The height of the texture.
[in] *numLevels*	The number of mip levels in the texture (including the top level).
[in] *numFaces*	The number of faces in the texture.
[in] *format*	The texture format.
[in] *type*	The texture type.
[in] *deSwizzle*	A flag that indicates whether the texture data will get deswizzled.

Return Values

Value	Description
SCE_OK	The operation was completed successfully.

Description

Reverts texture data from GXT runtime ready layout to a compact (non-padded) layout.

save

Saves a texture as a GXT file.

Definition

```
#include <gxt_io.h>
SceTextureErrorCode save(
    char const *fileName,
    Texture::File const &src,
    bool *swizzleFlags = NULL
);
```

Arguments

[in] *fileName*	The path to the GXT file in which the operation stores the texture.
[in] *src*	The texture file object containing the texture to be saved.
[in] *swizzleFlags*	Optional. An array of flags which indicate whether each of the textures in *src* should be swizzled. Defaults to `NULL`.

Return Values

Value	Description
`SCE_OK`	The operation was successful.
`SCE_TEXTURE_ERROR_CANNOT_OPEN_FILE`	The operation failed because the output file failed to open.
`SCE_TEXTURE_ERROR_INVALID_INPUT_DIMENSIONS`	The dimensions of one of the textures in *src* is either too large for the hardware or incompatible with the target format chosen.
`SCE_TEXTURE_ERROR_MEMORY_ALLOCATION_FAILURE`	The operation failed because it could not allocate the internal data needed to perform the save.

Description

Saves a texture as a GXT file.

serialize

Serializes a GXT format texture.

Definition

```
#include <gxt_io.h>
SceTextureErrorCode serialize(
    uint8_t *tgtBuffer,
    size_t tgtSize,
    Texture::File const &src,
    bool const *swizzleFlags = NULL,
    SceGxmTextureFormat const *texFormats = NULL
);
```

Arguments

[out] `tgtBuffer`	Receives the linear data.
[in] `tgtSize`	The size of the buffer that will receive the linear data.
[in] `src`	A texture file object containing the texture to be serialized.
[in] `swizzleFlags`	Optional. An array of flags which indicate whether each of the textures in `src` should be swizzled. Defaults to `NULL`, which indicates that all textures should be swizzled where possible.
[in] `texFormats`	Optional. An array of texture formats for each of the textures in `src`. Defaults to `NULL`, which indicates that suitable defaults should be chosen.

Return Values

Value	Description
`SCE_OK`	The operation was successful.
`SCE_TEXTURE_ERROR_BUFFER_WRITE_FAILURE`	The operation failed because the size of the storage buffer is not sufficient to store this texture.
`SCE_TEXTURE_ERROR_INVALID_INPUT_DIMENSIONS`	The operation failed because the dimensions of one of the textures in `src` is either too large for the hardware or is incompatible with the target format chosen.

Description

Serializes a GXT format texture.

supportsBorderData

Checks if the texture format supports border data.

Definition

```
#include <gxt_util.h>
bool supportsBorderData(
   SceGxmTextureFormat format
);
```

Arguments

[in] *format*	The texture format.

Return Values

If the format supports border data, true is returned; otherwise false is returned.

Description

Checks if the texture format supports border data.

sce::Texture::Palette Summary

sce::Texture::Palette

A container class for a texture palette.

Definition

```
#include <texture.h>
class Palette {};
```

Description

A container class for a texture palette.

Methods Summary

Methods	Description
build	A factory method used to create a palette instance which owns palette data.
build	A factory method used to create a palette instance which does not own palette data.
clone	Clones a palette duplicating any of the data it references.
numEntries	Gets the number of entries in the palette.
Palette	The constructor for the Palette class.
~Palette	The destructor for the Palette class.
pixelFormat	Gets the palette's data pixel format.
raw	Gets a pointer to the raw palette data.
size	Gets the size of the palette.
swap	Exchanges properties and data ownership between two palettes.

sce::Texture::Palette Constructors and Destructors

Palette

The constructor for the Palette class.

Definition

```
#include <texture.h>
inline Palette();
```

Arguments

None

Return Values

None

Description

The constructor for the Palette class.

Notes

This is a dummy constructor. In order to initialize an instance of the Palette class, use the factory method build().

~Palette

The destructor for the Palette class.

Definition

```
#include <texture.h>
~Palette();
```

Arguments

None

Return Values

None

Description

The destructor for the Palette class.

sce::Texture::Palette Public Static Methods

build

A factory method used to create a palette instance which owns palette data.

Definition

```
#include <texture.h>
static SceTextureErrorCode build(
    Palette &tgt,
    PixelFormat const &pixelFormat,
    uint32_t numEntries
);
```

Arguments

[out] *tgt*	Receives the palette instance built by this method.
[in] *pixelFormat*	The palette's pixel format descriptor.
[in] *numEntries*	The number of entries in the palette.

Return Values

Value	Description
SCE_OK	The operation was successful.
SCE_TEXTURE_ERROR_MEMORY_ALLOCATION_FAILURE	The operation failed when allocating memory for the texture.

Description

A factory method used to create a palette instance which owns palette data.

build

A factory method used to create a palette instance which does not own palette data.

Definition

```
#include <texture.h>
static SceTextureErrorCode build(
    Palette &tgt,
    PixelFormat const &pixelFormat,
    uint32_t numEntries,
    uint8_t *data
);
```

Arguments

[out] *tgt*	Receives the palette instance built by this method.
[in] *pixelFormat*	The palette's pixel format descriptor.
[in] *numEntries*	The number of entries in the palette.
[in] *data*	The palette's raw data. This is not deeply-copied and the caller is responsible for lifetime management.

Return Values

Value	Description
`SCE_OK`	The operation was successful.

Description

A factory method used to create a palette instance which does not own palette data.

clone

Clones a palette duplicating any of the data it references.

Definition

```
#include <texture.h>
static SceTextureErrorCode clone(
    Palette &tgt,
    Palette const &src
);
```

Arguments

[out] *tgt*	Receives the copy of the palette.
[in] *src*	A pointer to the source palette.

Return Values

Value	Description
`SCE_OK`	The operation was successful.
`SCE_TEXTURE_ERROR_MEMORY_ALLOCATION_FAILURE`	The operation failed because memory could not be allocated for the target palette.

Description

Clones a palette duplicating any of the data it references.

sce::Texture::Palette Public Instance Methods

numEntries

Gets the number of entries in the palette.

Definition

```
#include <texture.h>
inline uint32_t numEntries() const;
```

Arguments

None

Return Values

The number of entries in the palette.

Description

Gets the number of entries in the palette.

pixelFormat

Gets the palette's data pixel format.

Definition

```
#include <texture.h>
inline PixelFormat const &pixelFormat() const;
```

Arguments

None

Return Values

The palette's pixel format.

Description

Gets the palette's data pixel format.

raw

Gets a pointer to the raw palette data.

Definition

```
#include <texture.h>
inline uint8_t *raw() const;
```

Arguments

None

Return Values

A pointer to the palette's raw data.

Description

Gets a pointer to the raw palette data.

size

Gets the size of the palette.

Definition

```
#include <texture.h>
inline size_t size() const;
```

Arguments

None

Return Values

The palette’s size in bytes.

Description

Gets the size of the palette.

swap

Exchanges properties and data ownership between two palettes.

Definition

```
#include <texture.h>
Palette &swap(
    Palette &alt
);
```

Arguments

[in,out] *alt*	The palette instance with which to swap data.

Return Values

A reference to self once the data is swapped.

Description

Exchanges properties and data ownership between two palettes.

sce::Texture::PVR Summary

sce::Texture::PVR

A namespace containing PVR functionality.

Definition

```
namespace PVR {}
```

Description

A namespace containing PVR functionality.

Function Summary

Function	Description
deserialize	Deserializes a PVR format texture.
isValidFileInput	Queries whether a file is in the PVR format.
isValidFileOutput	Queries whether an output path points to a PVR file.
isValidInput	Performs a simple check to determine whether a stream of linear data is in the PVR format.
linearSize	Computes the linear size of a texture (in bytes) when stored in PVR format.
load	Loads a texture from a file in the PVR format.
save	Saves a texture as a PVR file.
serialize	Serializes a PVR format texture.

sce::Texture::PVR Functions

deserialize

Deserializes a PVR format texture.

Definition

```
#include <pvr_io.h>
SceTextureErrorCode deserialize(
    Texture::File &tgt,
    uint8_t const *srcBuffer,
    size_t srcSize
);
```

Arguments

[out] *tgt*	A texture file object in which the result of the operation will be stored (with an index of 0).
[in] *srcBuffer*	A pointer to the linear source data, which should be in the PVR format.
[in] *srcSize*	The size of the linear source data.

Return Values

Value	Description
SCE_OK	The operation was successful.
SCE_TEXTURE_ERROR_INVALID_PVR_DATA	The operation failed because the linear source data is either corrupted or is not in the PVR format.
SCE_TEXTURE_ERROR_MEMORY_ALLOCATION_FAILURE	The operation failed because an allocation error arose when creating the resulting texture.

Description

Deserializes a PVR format texture.

isValidFileInput

Queries whether a file is in the PVR format.

Definition

```
#include <pvr_io.h>
SceTextureErrorCode isValidFileInput(
    bool &valid,
    char const *fileName
);
```

Arguments

[out] *valid*	Receives whether the file is in the PVR format or not.
[in] *fileName*	The path to the input file.

Return Values

Value	Description
`SCE_OK`	The operation was successful.
`SCE_TEXTURE_ERROR_CANNOT_OPEN_FILE`	The operation failed because the output file failed to open.
`SCE_TEXTURE_ERROR_MEMORY_ALLOCATION_FAILURE`	The operation failed because it could not allocate the internal data needed to perform the query.

Description

Queries whether a file is in the PVR format. However, the file may be malformed, and this can only be detected by loading it.

isValidFileOutput

Queries whether an output path points to a PVR file.

Definition

```
#include <pvr_io.h>
SceTextureErrorCode isValidFileOutput(
    bool &valid,
    char const *fileName
);
```

Arguments

[out] *valid*	Receives whether the output path points to a PVR file.
[in] *fileName*	The path to the output file.

Return Values

Value	Description
SCE_OK	The operation was successful.

Description

Queries whether an output path points to a PVR file. This function simply checks whether an output path includes the .PVR extension.

isValidInput

Performs a simple check to determine whether a stream of linear data is in the PVR format.

Definition

```
#include <pvr_io.h>
bool isValidInput(
    uint8_t const *srcBuffer,
    size_t srcSize
);
```

Arguments

[in] *srcBuffer*	A pointer to the linear data to check.
[in] *srcSize*	The size of the linear data.

Return Values

Value	Description
`true`	The input is in the PVR format.
`false`	The input is not in the PVR format.

Description

Performs a simple check to determine whether a stream of linear data is in the PVR format. However, the data may be malformed, and this can only be detected by deserializing it.

linearSize

Computes the linear size of a texture (in bytes) when stored in PVR format.

Definition

```
#include <pvr_io.h>
SceTextureErrorCode linearSize(
    size_t &size,
    Texture::File const &src,
    uint32_t textureIndex = 0
);
```

Arguments

[out] *size*	Receives the computed size.
[in] *src*	A texture file containing the texture whose size will be computed.
[in] *textureIndex*	Optional. The index of the texture within the input texture file. Defaults to 0.

Return Values

Value	Description
SCE_OK	The operation was successful.
SCE_TEXTURE_ERROR_INVALID_VALUE	The operation failed because the supplied index was out of range.

Description

Computes the linear size of a texture (in bytes) when stored in PVR format.

load

Loads a texture from a file in the PVR format.

Definition

```
#include <pvr_io.h>
SceTextureErrorCode load(
    Texture::File &tgt,
    char const *fileName
);
```

Arguments

[out] *tgt*	A texture file object in which the result of the operation will be stored (with an index of 0).
[in] *fileName*	The path to the input file.

Return Values

Value	Description
SCE_OK	The operation was successful.
SCE_TEXTURE_ERROR_CANNOT_OPEN_FILE	The operation failed because the input file failed to open.
SCE_TEXTURE_ERROR_INVALID_PVR_DATA	The operation failed because the input file is either corrupted or is not in the PVR format.
SCE_TEXTURE_ERROR_MEMORY_ALLOCATION_FAILURE	The operation failed because an allocation error arose when creating the resulting texture.

Description

Loads a texture from a file in the PVR format.

save

Saves a texture as a PVR file.

Definition

```
#include <pvr_io.h>
SceTextureErrorCode save(
    char const *fileName,
    Texture::File const &src,
    uint32_t textureIndex = 0
);
```

Arguments

[in] *fileName*	The path to the PVR file in which to store the texture.
[in] *src*	The texture file object containing the texture to be saved.
[in] *textureIndex*	The index of the specific texture within the file to save. Default is 0.

Return Values

Value	Description
SCE_OK	The operation was successful.
SCE_TEXTURE_ERROR_CANNOT_OPEN_FILE	The operation failed because the output file failed to open.
SCE_TEXTURE_ERROR_MEMORY_ALLOCATION_FAILURE	The operation failed because it could not allocate the internal data needed to perform the save.

Description

Saves a texture as a PVR file.

serialize

Serializes a PVR format texture.

Definition

```
#include <pvr_io.h>
SceTextureErrorCode serialize(
    uint8_t *tgtBuffer,
    size_t tgtSize,
    Texture::File const &src,
    uint32_t textureIndex = 0
);
```

Arguments

[out] *tgtBuffer*	Receives the linear data.
[in] *tgtSize*	The size of the buffer that will receive the linear data.
[in] *src*	A texture file object containing the texture to be serialized.
[in] *textureIndex*	The index of the texture within the texture file to serialize. The default is 0.

Return Values

Value	Description
SCE_OK	The operation was successful.
SCE_TEXTURE_ERROR_BUFFER_WRITE_FAILURE	The operation failed because the size of the storage buffer is not sufficient to store this texture.
SCE_TEXTURE_ERROR_INVALID_VALUE	The operation failed because the supplied index was out of range.

Description

Serializes a PVR format texture.

sce::Texture::TGA Summary

sce::Texture::TGA

A namespace containing TGA functionality.

Definition

```
namespace TGA {}
```

Description

A namespace containing TGA functionality.

Function Summary

Function	Description
deserialize	Deserializes a TGA format texture.
isValidFileInput	Queries whether a file is in the TGA format.
isValidFileOutput	Queries whether an output path points to a TGA file.
isValidInput	Performs a simple check to determine whether a stream of linear data is in the TGA format.
linearSize	Computes the linear size of a texture (in bytes) when stored in the TGA format.
load	Loads a texture from a file in the TGA format.
save	Saves a texture as a TGA file.
serialize	Serializes a TGA format texture.

sce::Texture::TGA Functions

deserialize

Deserializes a TGA format texture.

Definition

```
#include <tga_io.h>
SceTextureErrorCode deserialize(
    Texture::File &tgt,
    uint8_t const *srcBuffer,
    size_t srcSize
);
```

Arguments

[out] *tgt*	A texture file object in which the result of the operation will be stored (with an index of 0).
[in] *srcBuffer*	A pointer to the linear source data, which should be in the TGA format.
[in] *srcSize*	The size of the linear source data.

Return Values

Value	Description
SCE_OK	The operation was successful.
SCE_TEXTURE_ERROR_INVALID_TGA_DATA	The operation failed because the linear source data is either corrupted or is not in the TGA format.
SCE_TEXTURE_ERROR_MEMORY_ALLOCATION_FAILURE	The operation failed because an allocation error arose when creating the resulting texture.

Description

Deserializes a TGA format texture.

isValidFileInput

Queries whether a file is in the TGA format.

Definition

```
#include <tga_io.h>
SceTextureErrorCode isValidFileInput(
    bool &valid,
    char const *fileName
);
```

Arguments

[out] *valid*	Receives whether the file is in the TGA format or not.
[in] *fileName*	The path to the input file.

Return Values

Value	Description
`SCE_OK`	The operation was successful.
`SCE_TEXTURE_ERROR_CANNOT_OPEN_FILE`	The operation failed because the output file failed to open.
`SCE_TEXTURE_ERROR_MEMORY_ALLOCATION_FAILURE`	The operation failed because it could not allocate the internal data needed to perform the query.

Description

Queries whether a file is in the TGA format. However, the file may be malformed, and this can only be detected by loading it.

isValidFileOutput

Queries whether an output path points to a TGA file.

Definition

```
#include <tga_io.h>
SceTextureErrorCode isValidFileOutput(
    bool &valid,
    char const *fileName
);
```

Arguments

[out] *valid*	Receives whether the output path points to a TGA file.
[in] *fileName*	The path to the output file.

Return Values

Value	Description
SCE_OK	The operation was successful.

Description

Queries whether an output path points to a TGA file. This function simply checks whether an output path includes the .TGA extension.

isValidInput

Performs a simple check to determine whether a stream of linear data is in the TGA format.

Definition

```
#include <tga_io.h>
bool isValidInput(
    uint8_t const *srcBuffer,
    size_t srcSize
);
```

Arguments

[in] *srcBuffer*	A pointer to the linear data to check.
[in] *srcSize*	The size of the linear data.

Return Values

Value	Description
`true`	The input is in the TGA format.
`false`	The input is not in the TGA format.

Description

Performs a simple check to determine whether a stream of linear data is in the TGA format. However, the data may be malformed, and this can only be detected by deserializing it.

linearSize

Computes the linear size of a texture (in bytes) when stored in the TGA format.

Definition

```
#include <tga_io.h>
SceTextureErrorCode linearSize(
    size_t &size,
    Texture::File const &src,
    uint32_t textureIndex = 0
);
```

Arguments

[out] *size*	Receives the computed size.
[in] *src*	A texture file containing the texture whose size will be computed.
[in] *textureIndex*	Optional. The index of the texture within the input texture file. Defaults to 0.

Return Values

Value	Description
`SCE_OK`	The operation was successful.
`SCE_TEXTURE_ERROR_INVALID_VALUE`	The operation failed because the supplied index was out of range.

Description

Computes the linear size of a texture (in bytes) when stored in the TGA format.

load

Loads a texture from a file in the TGA format.

Definition

```
#include <tga_io.h>
SceTextureErrorCode load(
    Texture::File &tgt,
    char const *fileName
);
```

Arguments

[out] *tgt*	A texture file object in which the result of the operation will be stored (with an index of 0).
[in] *fileName*	The path to the input file.

Return Values

Value	Description
`SCE_OK`	The operation was successful.
`SCE_TEXTURE_ERROR_CANNOT_OPEN_FILE`	The operation failed because the input file failed to open.
`SCE_TEXTURE_ERROR_INVALID_TGA_DATA`	The operation failed because the input file is either corrupted or is not in the TGA format.
`SCE_TEXTURE_ERROR_MEMORY_ALLOCATION_FAILURE`	The operation failed because an allocation error arose when creating the resulting texture.

Description

Loads a texture from a file in the TGA format.

save

Saves a texture as a TGA file.

Definition

```
#include <tga_io.h>
SceTextureErrorCode save(
    char const *fileName,
    Texture::File const &src,
    uint32_t textureIndex = 0
);
```

Arguments

[in] *fileName*	The path to the TGA file in which to store the texture.
[in] *src*	The texture file object containing the texture to be saved.
[in] *textureIndex*	The index of the specific texture within the file to save. Default is 0.

Return Values

Value	Description
SCE_OK	The operation was successful.
SCE_TEXTURE_ERROR_CANNOT_OPEN_FILE	The operation failed because the output file failed to open.
SCE_TEXTURE_ERROR_MEMORY_ALLOCATION_FAILURE	The operation failed because it could not allocate the internal data needed to perform the save.

Description

Saves a texture as a TGA file.

serialize

Serializes a TGA format texture.

Definition

```
#include <tga_io.h>
SceTextureErrorCode serialize(
    uint8_t *tgtBuffer,
    size_t tgtSize,
    Texture::File const &src,
    uint32_t textureIndex = 0
);
```

Arguments

[out] *tgtBuffer*	Receives the linear data.
[in] *tgtSize*	The size of the buffer that will receive the linear data.
[in] *src*	A texture file object containing the texture to be serialized.
[in] *textureIndex*	The index of the texture within the texture file to serialize. The default is 0.

Return Values

Value	Description
SCE_OK	The operation was successful.
SCE_TEXTURE_ERROR_BUFFER_WRITE_FAILURE	The operation failed because the size of the storage buffer is not sufficient to store this texture.
SCE_TEXTURE_ERROR_INVALID_VALUE	The operation failed because the supplied index was out of range.

Description

Serializes a TGA format texture.

sce::Texture::TGA Constants

Define Summary

Define	Value	Description
`SCE_GXT_TEXTURE_FLAG_HAS_BORDER_DATA`	`1U`	A flag to indicate that the texture contains border data.
`SCE_GXT_VERSION`	`0x10000003UL`	The GXT version number.