

# **PlayStation®Edge Animation Library Reference**



# Table of Contents

<b>Preface.....</b>	<b>4</b>
About This Document.....	5
<b>Shared PPU/SPU Runtime Data Types.....</b>	<b>6</b>
EdgeAnimPpuContext.....	7
EdgeAnimJointTransform.....	8
EdgeAnimBlendBranch.....	9
EdgeAnimBlendLeaf.....	11
EdgeAnimSkeleton.....	12
EdgeAnimAnimation.....	14
EdgeAnimMirrorPair.....	16
EdgeAnimCustomDataTable.....	17
<b>SPU Runtime Data Types.....</b>	<b>18</b>
EdgeAnimSpuContext.....	19
EdgeAnimFrameSetInfo.....	20
EdgeAnimPoseInfo.....	21
EdgeAnimCommand.....	22
<b>Shared PPU/SPU Functions.....</b>	<b>23</b>
edgeAnimSkeletonGetJointIndexByName.....	24
edgeAnimSkeletonGetJointIndexByHash.....	25
edgeAnimSkeletonGetUserChannelIndexByName.....	26
edgeAnimSkeletonGetUserChannelIndexByHash.....	27
edgeAnimGenerateNameHash.....	28
edgeAnimGetAnimTag.....	29
edgeAnimGetSkelTag.....	30
<b>PPU Functions.....</b>	<b>31</b>
edgeAnimPpuInitialize.....	32
edgeAnimPpuFinalize.....	33
edgeAnimComputeExternalStorageSize.....	34
edgeAnimEvaluateJoint.....	35
edgeAnimEvaluateUserChannel.....	36
<b>SPU Functions.....</b>	<b>37</b>
edgeAnimSpuInitialize.....	38
edgeAnimSpuFinalize.....	39
edgeAnimProcessBlendTree.....	40
edgeAnimProcessCommandList.....	42
edgeAnimLocalJointsToWorldMatrices3x4.....	43
edgeAnimLocalJointsToWorldMatrices4x4.....	44
edgeAnimLocalJointsToWorldJoints.....	45
edgeAnimWorldJointsToLocalJoints.....	46
edgeAnimJointsToMatrices4x4.....	47
edgeAnimMatrices4x4ToJoints.....	48
edgeAnimJointsToMatrices3x4.....	49
edgeAnimMatrices3x4ToJoints.....	50



---

edgeAnimBlendJointsLinear .....	51
edgeAnimBlendJointsRelative .....	53
edgeAnimBlendPose.....	55
edgeAnimBlendUserLinear .....	56
edgeAnimBlendUserRelative .....	58
edgeAnimPoseStackPush.....	60
edgeAnimPoseStackPop.....	61
edgeAnimPoseStackGetPose .....	62
edgeAnimCustomDataChunk.....	63
<b>Callback Functions .....</b>	<b>64</b>
EdgeAnimLeafCallback .....	65
EdgeAnimBranchCallback .....	66
EdgeAnimUserCallback .....	67
<b>Constants .....</b>	<b>68</b>
EdgeAnimBlendOp.....	69
EdgeAnimRelativeBlendMode .....	70



# Preface



---

# About This Document

---

## Purpose

This document provides an API reference for the animation component of the Edge library. Use this component to efficiently perform animation joint tree blending and transformation through the SPUs.

## Audience and Prerequisites

This document was written for PlayStation®3 developers who want to write high-performance applications for the PlayStation®3. It is assumed that such developers have familiarity with the following:

- C and C++
- PlayStation®3 hardware
- SCE standard library functions

## Related Documentation

In combination with this reference, the following documents provide complete usage and reference information about the Edge library:

- *PlayStation®Edge Library Overview*
- *PlayStation®Edge Geometry Library: Quick Start*
- *PlayStation®Edge Geometry Library Reference*
- *PlayStation®Edge Geometry Library for Offline Tool: Reference*
- *PlayStation®Edge Animation Library for Offline Tool: Reference*
- *PlayStation®Edge Zlib Library Reference*
- *PlayStation®Edge LZMA Library Reference*
- *PlayStation®Edge LZO Library Reference*
- *PlayStation®Edge DXT Library Reference*
- *PlayStation®Edge Post Library Reference*

## Typographic Conventions

This document uses the following typographic conventions:

Convention	Meaning
<code>fixed-width font</code>	Indicates programming code and literals, such as processing instructions, register names, data types, events, and file names. Also indicates function, structure, and macro names.
<b><code>fixed-width font + bold</code></b>	In structure/function definitions only, indicates names of structures and functions.
<i>fixed-width font + italics</i>	Indicates arguments, parameters, and variables.
<a href="#">blue + underlined text</a>	Indicates a hyperlink (blue displays in color printers or online only).



# Shared PPU/SPU Runtime Data Types



# EdgeAnimPpuContext

---

Contains information about the main-memory per-SPU temporary storage.

## Definition

---

```
#include <edge/anim/edgeanim_structs.h>
typedef struct
{
    /* undocumented */
} EdgeAnimPpuContext;
```

## Description

---

This structure contains information about the main-memory per-SPU temporary storage. The application must allocate space for this structure and pass it to [edgeAnimPpuInitialize\(\)](#), where it is initialized.

## Notes

---

This structure is accessed by SPU's and must therefore be allocated on the heap or otherwise exist in a global area.

## See Also

---

[edgeAnimPpuInitialize](#)



---

# EdgeAnimJointTransform

---

Describes a single joint transform.

## Definition

---

```
#include <edge/anim/edgeanim_structs.h>
typedef struct
{
    Vectormath::Aos::Quat rotation;
    Vectormath::Aos::Point3 translation;
    Vectormath::Aos::Vector4 scale;
} EdgeAnimJointTransform;
```

## Members

---

<i>rotation</i>	Rotational part of transform
<i>translation</i>	Translational part of transform
<i>scale</i>	Scaling part of transform

## Description

---

This structure describes a single joint transform.



# EdgeAnimBlendBranch

Specifies a branch in the blend tree.

## Definition

```
#include <edge/anim/edgeanim_structs.h>
typedef struct
{
    uint16_t operation;
    uint16_t left;
    uint16_t right;
    uint16_t flags;
    float alpha;
    uint32_t userVal;
} EdgeAnimBlendBranch;
```

## Members

<i>operation</i>	Blending operation (see below)
<i>left</i>	Left leaf/branch index (see below)
<i>right</i>	Right leaf/branch index (see below)
<i>flags</i>	Branch flags (see below)
<i>alpha</i>	Blend factor (0-1)
<i>userVal</i>	User-defined value

The *operation* member can take the following values:

Macro	Value	Description
EDGE_ANIM_BLENDOP_BLEND_LINEAR	0	Linear blend
EDGE_ANIM_BLENDOP_BLEND_ADD_DELTA_RIGHT	1	Linear blend between left ( $\alpha = 0$ ) and left + right ( $\alpha = 1$ )
EDGE_ANIM_BLENDOP_BLEND_ADD_DELTA_LEFT	2	Linear blend between right ( $\alpha = 0$ ) and right + left ( $\alpha = 1$ )
EDGE_ANIM_BLENDOP_COMPOSE_ADD_RIGHT	3	Compose left + right
EDGE_ANIM_BLENDOP_COMPOSE_ADD_LEFT	4	Compose right + left
EDGE_ANIM_BLENDOP_COMPOSE_SUB_RIGHT_FROM_LEFT	5	Compose left - right
EDGE_ANIM_BLENDOP_COMPOSE_SUB_LEFT_FROM_RIGHT	6	Compose right - left

For *left* and *right*, specify the bitwise OR of the index with one of the following two flags.

Macro	Value	Description
EDGE_ANIM_BLEND_TREE_INDEX_LEAF	0x8000	Leaf specifier
EDGE_ANIM_BLEND_TREE_INDEX_BRANCH	0x4000	Branch specifier

For flags, the bitwise OR of the following values can be specified.

Macro	Value	Description
EDGE_ANIM_FLAG_MIRROR	1	Result of blend will be mirrored.

## Description

This structure is used to specify a branch in the blend tree. A branch is a blend operation between two tree elements.

The left and right indices each represent either a leaf or a branch, and correspond to elements in the array of leaves or branches in the blend tree. In both cases index 0 is the first element.



A user-defined value can optionally be passed for *userVal*. This value is passed to the leaf callback function [EdgeAnimLeafCallback\(\)](#) (if it exists) for user leaf processing.

**See Also**

---

[edgeAnimProcessBlendTree](#), [edgeAnimBlendPose](#), [EdgeAnimLeafCallback](#)



# EdgeAnimBlendLeaf

Specifies a leaf in a blend tree.

## Definition

```
#include <edge/anim/edgeanim_structs.h>
typedef struct
{
    Union
    {
        uint32_t animationHeaderEa;
        uint32_t poseAddr;
    }
    uint16_t animationHeaderSize;
    uint16_t flags;
    float evalTime;
    uint32_t userVal;
} EdgeAnimBlendLeaf;
```

## Members

<i>animationHeaderEa</i>	Effective address of animation header
<i>poseAddr</i>	Address of precomputed pose
<i>animationHeaderSize</i>	Size of animation header
<i>flags</i>	Leaf flags (see below)
<i>evalTime</i>	Evaluation time (seconds)
<i>userVal</i>	User-defined value

## Description

This structure specifies a leaf in a blend tree. A leaf represents an animation evaluated at a specific time.

*evalTime* is clamped at evaluation time to the range (0, *d*) where *d* is the duration of the clip.

A user-defined value can optionally be passed for *userVal*. This value is passed to the leaf callback function [EdgeAnimLeafCallback\(\)](#) (if it exists) for user leaf processing.

If the leaf represents a precomputed pose, *poseAddr* is used to specify the address of the pose in main memory or local store. In this case, either the `EDGE_ANIM_FLAG_POSE_FROM_MAIN` or `EDGE_ANIM_FLAG_POSE_FROM_LOCAL` flag must be set, respectively.

For *flags*, the bitwise OR of the following values can be specified.

Macro	Value	Description
<code>EDGE_ANIM_FLAG_MIRROR</code>	1	Animation will be mirrored.
<code>EDGE_ANIM_FLAG_POSE_FROM_MAIN</code>	2	Leaf specifies a precomputed pose in main memory.
<code>EDGE_ANIM_FLAG_POSE_FROM_LOCAL</code>	4	Leaf specifies a precomputed pose in local memory.

## See Also

[edgeAnimProcessBlendTree](#), [EdgeAnimLeafCallback](#)



# EdgeAnimSkeleton

Describes the skeleton binary header.

## Definition

```
#include <edge/anim/edgeanim_structs.h>
typedef struct
{
    uint32_t tag;
    uint32_t sizeTotal;
    uint32_t sizeCustomData;
    uint32_t sizeNameHashes;
    uint16_t numJoints;
    uint16_t numUserChannels;
    uint16_t numSimdHierarchyQuads;
    uint16_t locomotionJointIndex;
    uint32_t offsetBasePose;
    uint32_t offsetParentIndicesArray;
    uint32_t offsetJointNameHashArray;
    uint32_t offsetUserChannelNameHashArray;
    uint32_t offsetUserChannelNodeNameHashArray;
    uint32_t offsetUserChannelFlagsArray;
    uint32_t offsetCustomData;
    uint32_t pad1[3];
    uint16_t simdHierarchy[0];
} EdgeAnimSkeleton;
```

## Members

<i>tag</i>	Version tag
<i>sizeTotal</i>	Total size
<i>sizeCustomData</i>	Size of custom user data or custom data table
<i>sizeNameHashes</i>	Size of name hashes
<i>numJoints</i>	Number of joints
<i>numUserChannels</i>	Number of user channels
<i>numSimdHierarchyQuads</i>	Number of quads in the single instruction, multiple data (SIMD) hierarchy
<i>locomotionJointIndex</i>	Index of the joint used for locomotion deltas
<i>offsetBasePose</i>	Offset to base pose ( <a href="#">EdgeAnimJointTransform*</a> )
<i>offsetParentIndicesArray</i>	Offset to parent indices array (int16_t*)
<i>offsetJointNameHashArray</i>	Offset to joint name hash array (uint32_t*)
<i>offsetUserChannelNameHashArray</i>	Offset to user channel name hash array (uint32_t*)
<i>offsetUserChannelNodeNameHashArray</i>	Offset to user channel node name hash array (uint32_t*)
<i>offsetUserChannelFlagsArray</i>	Offset to user channel flags array (uint8_t*)
<i>offsetCustomData</i>	Offset to custom user data
<i>pad1[3]</i>	Unused
<i>simdHierarchy[0]</i>	SIMD hierarchy

## Description

This structure describes the skeleton binary header.

All offsets are relative to their address – use the macro `EDGE_OFFSET_GET_POINTER()` to get the pointer from an offset.



*tag* contains the current version number of the binary and should match the value returned by [edgeAnimGetSkelTag\(\)](#).

*sizeTotal* gives the total size of the binary. Because the custom data and the hash names exist at the end of the skeleton header, the application can skip their upload if they are not required by subtracting their sizes from the total. (None of the core SPU processing functions require custom data or hash names.)

*offsetParentIndicesArray* contains the offset to the array of joint parent indices (-1 indicates that there is no parent).

*simdHierarchy* contains the hierarchy information in a SIMD friendly format required by the SPU runtime. Each `uint32_t` contains a 16-bit parent/child index pair. Entries can be duplicated within this array.

*numSimdHierarchyQuads* gives the quadword size of *simdHierarchy*.

### **See Also**

---

[edgeAnimGetSkelTag](#)



# EdgeAnimAnimation

Describes the animation binary header.

## Definition

```
#include <edge/anim/edgeanim_structs.h>
typedef struct
{
    uint32_t tag;
    float duration;
    float sampleFrequency;
    uint16_t sizeHeader;
    uint16_t numJoints;
    uint16_t numFrames;
    uint16_t numFrameSets;
    uint16_t evalBufferSizeRequired;
    uint16_t numConstRChannels;
    uint16_t numConstTChannels;
    uint16_t numConstSChannels;
    uint16_t numConstUserChannels;
    uint16_t numAnimRChannels;
    uint16_t numAnimTChannels;
    uint16_t numAnimSChannels;
    uint16_t numAnimUserChannels;
    uint16_t flags;
    uint32_t sizeJointsWeightArray;
    uint32_t eaUserJointWeightArray;
    uint32_t offsetJointsWeightArray;
    uint32_t offsetFrameSetDmaArray;
    uint32_t offsetFrameSetInfoArray;
    uint32_t offsetConstRData;
    uint32_t offsetConstTData;
    uint32_t offsetConstSData;
    uint32_t offsetConstUserData;
    uint32_t offsetPackingSpecs;
    uint32_t offsetCustomData;
    uint32_t sizeCustomData;
    uint32_t offsetLocomotionDelta;
    uint32_t pad1;
    uint16_t channelTables[0];
} EdgeAnimAnimation;
```

## Members

<i>tag</i>	Version tag
<i>duration</i>	Duration of animation (seconds)
<i>sampleFrequency</i>	Sampling frequency (Hz)
<i>sizeHeader</i>	Size of header, aligned to 16 bytes
<i>numJoints</i>	Number of joints
<i>numFrames</i>	Number of frames
<i>numFrameSets</i>	Number of frame sets
<i>evalBufferSizeRequired</i>	Size of evaluation buffer required
<i>numConstRChannels</i>	Number of constant rotation channels
<i>numConstTChannels</i>	Number of constant translation channels
<i>numConstSChannels</i>	Number of constant scale channels
<i>numConstUserChannels</i>	Number of constant user channels
<i>numAnimRChannels</i>	Number of animated rotation channels



---

<i>numAnimTChannels</i>	Number of animated translation channels
<i>numAnimSChannels</i>	Number of animated scale channels
<i>numAnimUserChannels</i>	Number of animated user channels
<i>flags</i>	Internal flags
<i>sizeJointsWeightArray</i>	Size of weights array (0 if no array)
<i>eaUserJointWeightArray</i>	User override for joint weights array. It should be set to NULL by tools
<i>offsetJointsWeightArray</i>	Offset to weights array
<i>offsetFrameSetDmaArray</i>	Offset to frame set DMA array ( <code>CellDmaListElement*</code> )
<i>offsetFrameSetInfoArray</i>	Offset to frame set info array ( <a href="#">EdgeAnimFrameSetInfo*</a> )
<i>offsetConstRData</i>	Offset to constant rotation data
<i>offsetConstTData</i>	Offset to constant translation data
<i>offsetConstSData</i>	Offset to constant scale data
<i>offsetConstUserData</i>	Offset to constant user data
<i>offsetPackingSpecs</i>	Offset to packing specifications (for bit-packed channels)
<i>offsetCustomData</i>	Offset to custom user data or custom data table
<i>sizeCustomData</i>	Size of custom user data or custom data table
<i>offsetLocomotionDelta</i>	Offset to optional locomotion deltas (0 if no deltas)
<i>pad1</i>	Unused
<i>channelTables</i>	Channel tables

## Description

---

This structure describes the animation binary header.

All offsets are relative to their address – use the macro `EDGE_OFFSET_GET_POINTER()` to get the pointer from an offset.

*tag* contains the current version number of the binary and should match the value returned by [edgeAnimGetAnimTag\(\)](#).

*evalBufferSizeRequired* gives the size of the evaluation buffer required for this clip. A value of at least this size must be passed as the *maxSizeEvalBuffer* argument to [edgeAnimSpuInitialize\(\)](#).

*sizeJointsWeightArray* gives the size of the per-joint weights array. If it is 0, it is assumed to be a full body animation with all weights set to 1.

## See Also

---

[edgeAnimGetAnimTag](#)



# EdgeAnimMirrorPair

Specifies a mirroring operation for a joint or joint pair.

## Definition

```
#include <edge/anim/edgeanim_structs.h>
typedef struct
{
    uint16_t idx0;
    uint16_t idx1;
    uint32_t spec;
} EdgeAnimMirrorPair;
```

## Members

<i>idx0</i>	Index of first joint
<i>idx1</i>	Index of second joint
<i>spec</i>	Mirroring operation

## Description

This structure is used to specify a mirroring operation for a joint or joint pair.

*spec* defines the mirroring operation that is to be applied to the two joints *idx0* and *idx1*, and it consists of a series of sign inversions and component shuffles of the rotation and translation components. The two joints will also get their entries swapped within the pose.

This structure is also used to specify mirroring for single (non-paired) joints, in which case *idx0* and *idx1* should be the same.

For *spec*, the following values can be specified.

Macro	Value	Description
EDGE_ANIM_MIRROR_SPEC_NON_PAISED	0x881122b3U	Mirroring operation for non-paired joints
EDGE_ANIM_MIRROR_SPEC_LINK	0xb8219203U	Mirroring operation for paired joints that have non-paired parents
EDGE_ANIM_MIRROR_SPEC_PAISED	0x08192233U	Mirroring operation for paired joints that have paired parents

## See Also

[edgeAnimProcessCommandList](#), [edgeAnimProcessBlendTree](#)



---

# EdgeAnimCustomDataTable

---

Describes an optional structure to store multiple, independent custom data entries for use with [EdgeAnimSkeleton](#) and [EdgeAnimAnimation](#) and accessible by hash name identifier.

## Definition

---

```
#include <edge/anim/edgeanim_structs.h>
typedef struct
{
    uint32_t numEntries;
    uint32_t offsetHashArray;
    uint32_t offsetEntrySizes;
    uint32_t offsetEntries;
} EdgeAnimCustomDataTable ;
```

## Members

---

<i>numEntries</i>	Number of data entries in the table
<i>offsetHashArray</i>	Offset to the hash names list ( uint32_t [] )
<i>offsetEntrySizes</i>	Offset to the entry sizes list ( uint32_t [] )
<i>offsetEntries</i>	Offset to the entry data list

## Description

---

This structure provides a method to store multiple independent pieces of custom data with the [EdgeAnimSkeleton](#) and [EdgeAnimAnimation](#) structures. The use of this table is optional, and has been implemented as a base for supporting custom data within Edge Animation.

Each entry is identified by a hash value that is generated by [edgeAnimGenerateNameHash\(\)](#). A helper function, [edgeAnimCustomDataChunk\(\)](#), is available to locate entries within the table.

## See Also

---

[edgeAnimGenerateNameHash](#), [edgeAnimCustomDataChunk](#)



# SPU Runtime Data Types



---

# EdgeAnimSpuContext

---

Contains SPU context-specific data.

## Definition

---

```
#include <edge/anim/edgeanim_structs.h>
typedef struct
{
    /* not documented */
} EdgeAnimSpuContext;
```

## Description

---

This structure contains SPU context-specific data. The application must instantiate this structure and pass it to [edgeAnimSpuInitialize\(\)](#), where it is initialized.

## See Also

---

[edgeAnimSpuInitialize](#)



---

# EdgeAnimFrameSetInfo

---

Describes a single frameset (subset of frames) within an animation.

## Definition

---

```
#include <edge/anim/edgeanim_structs.h>
typedef struct
{
    uint16_t baseFrame;
    uint16_t numIntraFrames;
} EdgeAnimFrameSetInfo;
```

## Members

---

<i>baseFrame</i>	Index of first frame in frameset
<i>numIntraFrames</i>	Number of intra-frames in frameset

## Description

---

This structure describes a single frameset (subset of frames) within an animation.



---

# EdgeAnimPoseInfo

---

Contains information about a single pose.

## Definition

---

```
#include <edge/anim/edgeanim_structs.h>
typedef struct
{
    EdgeAnimJointTransform* jointArray;
    uint8_t* weightArray;
    float* userChannelArray;
    uint8_t* userChannelWeightArray;
    uint32_t* flags;
    uint32_t pad;
} EdgeAnimPoseInfo;
```

## Members

---

<i>jointArray</i>	Pointer to joint array
<i>weightArray</i>	Pointer to weights array
<i>userChannelArray</i>	Pointer to user channel array
<i>userChannelWeightArray</i>	Pointer to user channel weights array
<i>flags</i>	Pointer to flags
<i>pad</i>	Unused

## Description

---

This structure contains information about a single pose.

## See Also

---

[edgeAnimPoseStackGetPose](#)



# EdgeAnimCommand

Describes a single animation command.

## Definition

```
#include <edge/anim/edgeanim_structs.h>
typedef struct
{
    uint16_t command;
    uint16_t arg0;
    union{
        const EdgeAnimBlendLeaf* leaf;
        const EdgeAnimBlendBranch* branch;
        uint32_t user;
    };
} EdgeAnimCommand;
```

## Members

<i>command</i>	Command ID (see below)
<i>arg0</i>	Additional argument for user commands
<i>leaf</i>	Pointer to leaf if command is <code>EDGE_ANIM_CMD_PUSH_AND_EVAL</code>
<i>branch</i>	Pointer to branch if command is <code>EDGE_ANIM_CMD_BLEND_AND_POP</code>
<i>user</i>	User-specified value for custom commands

The *command* member can be set to the following values:

Macro	Value	Description
<code>EDGE_ANIM_CMD_END_LIST</code>	0	End of command list
<code>EDGE_ANIM_CMD_EVAL</code>	1	Evaluate to top of the pose stack
<code>EDGE_ANIM_CMD_PUSH_AND_EVAL</code>	2	Allocate a temporary pose and evaluate
<code>EDGE_ANIM_CMD_BLEND_AND_POP</code>	3	Blend and pop
<code>EDGE_ANIM_CMD_MIRROR</code>	4	Mirror pose at top of pose stack

## Description

This structure describes an animation command. Commands are generated internally by Edge Animation when processing a blend tree.

For user-generated command lists, any command not listed above is interpreted as a user command, and processing is passed from [edgeAnimProcessCommandList\(\)](#) to a user-specified callback.

## See Also

[edgeAnimProcessCommandList](#), [edgeAnimProcessBlendTree](#)



# Shared PPU/SPU Functions



---

# edgeAnimSkeletonGetJointIndexByName

---

Returns the index of the joint with the specified name.

## Definition

---

```
#include <edge/anim/edgeanim_spu.h>
or
#include <edge/anim/edgeanim_ppu.h>

int32_t edgeAnimSkeletonGetJointIndexByName (
    const EdgeAnimSkeleton* skeleton,
    const char* jointName
)
```

## Calling Conditions

---

Multithread safe

## Arguments

---

<i>skeleton</i>	Pointer to skeleton containing the joint
<i>jointName</i>	Name of joint

## Return Values

---

Returns the index of the joint.

Returns -1 if the joint is not found.

## Description

---

This function returns the index of the joint with the specified name.

## Notes

---

This function internally computes the hashed value of the string. Therefore for performance reasons it is preferable to call [edgeAnimSkeletonGetJointIndexByHash\(\)](#) instead, with a pre-computed hash value.

## See Also

---

[edgeAnimSkeletonGetJointIndexByHash](#)



---

# edgeAnimSkeletonGetJointIndexByHash

---

Returns the index of the joint with the specified hashed name.

## Definition

---

```
#include <edge/anim/edgeanim_spu.h>
or
#include <edge/anim/edgeanim_ppu.h>

int32_t edgeAnimSkeletonGetJointIndexByHash(
    const EdgeAnimSkeleton* skeleton,
    uint32_t jointHash
)
```

## Calling Conditions

---

Multithread safe

## Arguments

---

<i>skeleton</i>	Pointer to skeleton containing the joint
<i>jointHash</i>	Hashed name of joint

## Return Values

---

Returns the index of the joint.

Returns -1 if the joint is not found.

## Description

---

This function returns the index of the joint with the specified hashed name.

## Notes

---

Hashed names can be generated from strings by calling [edgeAnimGenerateNameHash\(\)](#).

## See Also

---

[edgeAnimSkeletonGetJointIndexByName](#), [edgeAnimGenerateNameHash](#)



---

# edgeAnimSkeletonGetUserChannelIndexByName

---

Returns the index of the user channels with the specified node/name pair.

## Definition

---

```
#include <edge/anim/edgeanim_spu.h>
or
#include <edge/anim/edgeanim_ppu.h>

int32_t edgeAnimSkeletonGetUserChannelIndexByName (
    const EdgeAnimSkeleton* skeleton,
    const char* userChannelNodeName,
    const char* userChannelName
)
```

## Calling Conditions

---

Multithread safe

## Arguments

---

<i>skeleton</i>	Pointer to skeleton containing the user channel
<i>userChannelNodeName</i>	Name of user channel node
<i>userChannelName</i>	Name of user channel

## Return Values

---

Returns the index of the user channel.

Returns -1 if user channel is not found.

## Description

---

This function returns the index of the user channel with the specified node/name pair.

*userChannelNodeName* specifies the name of the parent node of the channel.

*userChannelName* specifies the name of the channel itself.

## Notes

---

This function internally computes the hashed value of the string. Therefore, for performance reasons it is preferable to call [edgeAnimSkeletonGetUserChannelIndexByHash\(\)](#) instead, with a pre-computed hash value.

## See Also

---

[edgeAnimSkeletonGetUserChannelIndexByHash](#)



---

# edgeAnimSkeletonGetUserChannelIndexByHash

---

Returns the index of the user channels with the specified node/name pair.

## Definition

---

```
#include <edge/anim/edgeanim_spu.h>
or
#include <edge/anim/edgeanim_ppu.h>

int32_t edgeAnimSkeletonGetUserChannelIndexByName (
    const EdgeAnimSkeleton* skeleton,
    const char* userChannelNodeName,
    const char* userChannelName
)
```

## Calling Conditions

---

Multithread safe

## Arguments

---

<i>skeleton</i>	Pointer to skeleton containing the user channel
<i>userChannelNodeNameHash</i>	Hashed name of user channel node
<i>userChannelNameHash</i>	Hashed name of user channel

## Return Values

---

Returns the index of the user channel.

Returns -1 if user channel is not found.

## Description

---

This function returns the index of the user channel with the specified node/name pair.

*userChannelNodeName* specifies the hashed name of the parent node of the channel.

*userChannelName* specifies the hashed name of the channel itself.

## Notes

---

Hashed names can be generated from strings by calling [edgeAnimGenerateNameHash\(\)](#).

## See Also

---

[edgeAnimSkeletonGetUserChannelIndexByName](#), [edgeAnimGenerateNameHash](#)



---

# edgeAnimGenerateNameHash

---

Computes the hash value of a string.

## Definition

---

```
#include <edge/anim/edgeanim_spu.h>
or
#include <edge/anim/edgeanim_ppu.h>

uint32_t edgeAnimGenerateNameHash(
    const char* name
)
```

## Calling Conditions

---

Multithread safe

## Arguments

---

<i>name</i>	String to be evaluated
-------------	------------------------

## Return Values

---

Returns the hash value.

## Description

---

This function computes the hash value of a string.



---

# edgeAnimGetAnimTag

---

Gets the animation binary tag.

## Definition

---

```
#include <edge/anim/edgeanim_common.h>
uint32_t edgeAnimGetAnimTag()
```

## Calling Conditions

---

Multithread safe

## Arguments

---

None.

## Return Values

---

Returns the animation binary tag.

## Description

---

This function gets the animation binary *tag* from [EdgeAnimAnimation](#). The tag has the form `EA $nn$`  where `EA` means “Edge Animation” and  $nn$  is the current version number. The version is incremented at every modification of the format.

## See Also

---

[EdgeAnimAnimation](#)



---

# edgeAnimGetSkelTag

---

Gets the skeleton binary tag.

## Definition

---

```
#include <edge/anim/edgeanim_common.h>
uint32_t edgeAnimGetSkelTag()
```

## Calling Conditions

---

Multithread safe

## Arguments

---

None.

## Return Values

---

Returns the skeleton binary tag.

## Description

---

This function gets the skeleton binary *tag* from [EdgeAnimSkeleton](#). The tag has the form *ESnn*, where *ES* means “Edge Skeleton” and *nn* is the current version number. The version is incremented at every modification of the format.

## See Also

---

[EdgeAnimSkeleton](#)



# PPU Functions



# edgeAnimPpuInitialize

Initializes the PPU Edge Animation library.

## Definition

```
#include <edge/anim/edgeanim_ppu.h>
void edgeAnimPpuInitialize(
    EdgeAnimPpuContext* ppuContext,
    uint32_t numSpus,
    int32_t spuExternalStorageMask,
    size_t sizeExternalStoragePerSpu,
    void* externalStorageBlock
)
```

## Calling Conditions

Multithread safe (as long as different threads work on different *ppuContexts*)

## Arguments

<i>ppuContext</i>	Address of PPU context structure, which is filled by this function
<i>numSpus</i>	Maximum number of SPUs to process animation (0–6)
<i>spuExternalStorageMask</i>	Bitmask defining which SPUs have external storage
<i>sizeExternalStoragePerSpu</i>	Size of per-SPU external storage
<i>externalStorageBlock</i>	Address of external storage block

## Return Values

None.

## Description

This function initializes the PPU Edge Animation library.

If no external pose stack storage is required, the last three arguments should be set to 0.

If external pose stack storage is required, the application must allocate and specify an area of main memory for this purpose. This buffer must be 16-byte aligned and its size determined by passing *numSpus*, *spuExternalStorageMask*, and *sizeExternalStoragePerSpu* as arguments to [edgeAnimComputeExternalStorageSize\(\)](#).

For *spuExternalStorageMask*, specify 1 for each bit corresponding to an SPU that will have external storage.

## Notes

The *ppuContext* structure is accessed by SPUs and must therefore be allocated on the heap or otherwise exist in a global area.

## See Also

[edgeAnimComputeExternalStorageSize](#)



---

# edgeAnimPpuFinalize

---

Terminates the PPU Edge Animation library.

## Definition

---

```
#include <edge/anim/edgeanim_ppu.h>
void edgeAnimPpuFinalize (
    EdgeAnimPpuContext* ppuContext
)
```

## Calling Conditions

---

Multithread safe (as long as different threads work on different *ppuContexts*)

## Arguments

---

<i>ppuContext</i>	Pointer to PPU context
-------------------	------------------------

## Return Values

---

None.

## Description

---

This function terminates the PPU Edge Animation library.



---

# edgeAnimComputeExternalStorageSize

---

Computes the total storage size of the external pose cache buffer.

## Definition

---

```
#include <edge/anim/edgeanim_ppu.h>
size_t edgeAnimComputeExternalStorageSize (
    uint32_t numSpus,
    int32_t spuExternalStorageMask,
    size_t sizeExternalStoragePerSpu
)
```

## Calling Conditions

---

Multithread safe.

## Arguments

---

<i>numSpus</i>	Maximum number of SPU's to process animation (0-6)
<i>spuExternalStorageMask</i>	Bitmask defining which SPU's have external storage
<i>sizeExternalStoragePerSpu</i>	Size of per-SPU external storage

## Return Values

---

The computed storage size is returned.

## Description

---

This function computes the total storage size of the external pose cache buffer.  
The arguments must be the same as those passed to [edgeAnimPpuInitialize\(\)](#).

## See Also

---

[edgeAnimPpuInitialize](#)



# edgeAnimEvaluateJoint

Evaluates a single joint for a given animation and time.

## Definition

```
#include <edge/anim/edgeanim_ppu.h>
void edgeAnimEvaluateJoint (
    EdgeAnimJointTransform* outputJoint,
    const EdgeAnimAnimation* anim,
    const EdgeAnimSkeleton* skel,
    uint32_t jointIndex,
    float evalTime
)
```

## Calling Conditions

Multithread safe.

## Arguments

<i>outputJoint</i>	Pointer to output joint transform
<i>anim</i>	Pointer to animation
<i>skel</i>	Pointer to skeleton
<i>jointIndex</i>	Index of joint to evaluate
<i>evalTime</i>	Evaluation time (seconds)

## Return Values

None.

## Description

This function evaluates a single joint for a given animation and time.

*evalTime* is clamped to the range (0, d) where *d* is the duration of the clip.

## Notes

This function is provided for convenience rather than efficiency and is intended to be used only in rare cases or at initialization. For general evaluation, use the SPU processing functions.

This function does not support animation data created with bit-packing enabled.



---

# edgeAnimEvaluateUserChannel

---

Evaluates a single user channel for a given animation and time.

## Definition

---

```
#include <edge/anim/edgeanim_ppu.h>
float edgeAnimEvaluateUserChannel (
    const EdgeAnimAnimation* anim,
    const EdgeAnimSkeleton* skel,
    uint32_t channelIndex,
    float evalTime
)
```

## Calling Conditions

---

Multithread safe.

## Arguments

---

<i>anim</i>	Pointer to animation
<i>skel</i>	Pointer to skeleton
<i>channelIndex</i>	Index of user channel to evaluate
<i>evalTime</i>	Evaluation time (seconds)

## Return Values

---

Returns the evaluated user channel.

## Description

---

This function evaluates a single user channel for a given animation and time.  
*evalTime* is clamped to the range (0, d) where *d* is the duration of the clip.

## Notes

---

This function is provided for convenience rather than efficiency and is intended to be used only in rare cases or at initialization. For general evaluation, use the SPU processing functions.  
This function does not support animation data created with bit-packing enabled.



# SPU Functions



# edgeAnimSpuInitialize

Initializes the SPU Edge Animation library.

## Definition

```
#include <edge/anim/edgeanim_spu.h>
void edgeAnimSpuInitialize (
    EdgeAnimSpuContext* spuContext,
    const EdgeAnimPpuContext* ppuContext,
    uint32_t spuId,
    void* lsStorage,
    uint32_t sizeLsStorage,
    uint32_t numJoints,
    uint32_t numUserChannels,
    uint32_t maxSizeEvalBuffer,
    uint32_t maxSizeUserBuffer
)
```

## Arguments

<i>spuContext</i>	Address of SPU context structure, which is filled by this function
<i>ppuContext</i>	Pointer to PPU context
<i>spuId</i>	Unique ID of this SPU (0-5)
<i>lsStorage</i>	Address of local storage to be used by Edge Animation
<i>sizeLsStorage</i>	Size of local storage available to Edge Animation
<i>numJoints</i>	Total number of joints
<i>numUserChannels</i>	Total number of user channels
<i>maxSizeEvalBuffer</i>	Maximum size of one evaluation buffer (EDGE_ANIM_EVAL_BUFFER_SIZE by default)
<i>maxSizeUserBuffer</i>	Maximum size of one user buffer (0 by default)

## Return Values

None.

## Description

This function initializes the SPU Edge Animation library.

*lsStorage* and *sizeLsStorage* specify the address and size of the local storage area made available to Edge Animation. This function asserts if the buffer size is too small. The minimum required size is calculated as follows:

$$3 * (maxSizeEvalBuffer + maxSizeUserBuffer + sizePose)$$

This allocates three slots in the local store pose cache. Any additional space supplied is used for extra slots.

*maxSizeEvalBuffer* specifies the maximum evaluation buffer size and should be greater than or equal to the evaluation buffer size specified in the tools for any given animation.

*maxSizeUserBuffer* specifies the size of the user buffer passed to leaf and/or branch callbacks.



---

# edgeAnimSpuFinalize

---

Terminates the SPU Edge Animation library.

## Definition

---

```
#include <edge/anim/edgeanim_spu.h>
void edgeAnimSpuFinalize (
    EdgeAnimSpuContext* spuContext
)
```

## Arguments

---

<i>spuContext</i>	Pointer to SPU context
-------------------	------------------------

## Return Values

---

None.

## Description

---

This function terminates the SPU Edge Animation library.



# edgeAnimProcessBlendTree

Processes the blend tree, leaving the final result on top of the pose stack.

## Definition

```
#include <edge/anim/edgeanim_spu.h>
void edgeAnimProcessBlendTree(
    EdgeAnimSpuContext* spuContext,
    uint16_t rootIndex,
    const EdgeAnimBlendBranch* blendBranches,
    uint32_t numBranches,
    const EdgeAnimBlendLeaf* blendLeaves,
    uint32_t numLeaves,
    const EdgeAnimSkeleton* skeleton,
    const EdgeAnimMirrorPair* mirrorPairs,
    uint32_t numMirrorPairs,
    const EdgeAnimBranchCallback branchCallback,
    const EdgeAnimLeafCallback leafCallback
)
```

## Arguments

<i>spuContext</i>	Pointer to SPU context
<i>rootIndex</i>	Index of blend tree root
<i>blendBranches</i>	Pointer to array of branches
<i>numBranches</i>	Number of branches
<i>blendLeaves</i>	Pointer to array of leaves
<i>numLeaves</i>	Number of leaves
<i>skeleton</i>	Pointer to skeleton
<i>mirrorPairs</i>	Pointer to mirroring specification (0 by default)
<i>numMirrorPairs</i>	Number of entries in mirror specification (0 by default)
<i>branchCallback</i>	Pointer to user branch callback function (0 by default)
<i>leafCallback</i>	Pointer to user leaf callback function (0 by default)

For *rootIndex*, specify the bitwise OR of the index with one of the following two flags:

Macro	Value	Description
EDGE_ANIM_BLEND_TREE_INDEX_LEAF	0x8000	Leaf specifier
EDGE_ANIM_BLEND_TREE_INDEX_BRANCH	0x4000	Branch specifier

## Return Values

None.

## Description

This function processes the blend tree, leaving the final result on top of the pose stack.

*rootIndex* specifies the root index of the tree (normally 0) and must explicitly refer to either a leaf or a branch.

If mirroring is to be performed, the mirroring specification must be passed to this function. Mirroring is enabled on a per-leaf/branch basis by setting the appropriate flags inside [EdgeAnimBlendLeaf](#)/[EdgeAnimBlendBranch](#).

Callbacks can optionally be specified. See [EdgeAnimBranchCallback\(\)](#) and [EdgeAnimLeafCallback\(\)](#) for details.



## **See Also**

---

[EdgeAnimBlendBranch](#), [EdgeAnimBlendLeaf](#), [EdgeAnimBranchCallback](#), [EdgeAnimLeafCallback](#),  
[edgeAnimProcessBlendTree](#), [edgeAnimProcessCommandList](#)



# edgeAnimProcessCommandList

Processes a blend tree command list.

## Definition

```
#include <edge/anim/edgeanim_spu.h>
void edgeAnimProcessCommandList(
    EdgeAnimSpuContext* spuContext,
    const EdgeAnimCommand* commandList,
    const EdgeAnimSkeleton* skeleton,
    const EdgeAnimMirrorPair* mirrorPairs,
    uint32_t numMirrorPairs,
    const EdgeAnimBranchCallback branchCallback,
    const EdgeAnimLeafCallback leafCallback,
    const EdgeAnimUserCallback userCallback
)
```

## Arguments

<i>spuContext</i>	Pointer to SPU context
<i>commandList</i>	Pointer to command list
<i>skeleton</i>	Pointer to skeleton
<i>mirrorPairs</i>	Pointer to mirroring specification (0 by default)
<i>numMirrorPairs</i>	Number of entries in mirror specification (0 by default)
<i>branchCallback</i>	Pointer to user branch callback function (0 by default)
<i>leafCallback</i>	Pointer to user leaf callback function (0 by default)
<i>userCallback</i>	Pointer to user callback function for unrecognized commands (0 by default)

## Return Values

None.

## Description

This function processes a blend tree command list. It is called internally by [edgeAnimProcessBlendTree\(\)](#).

## See Also

[EdgeAnimCommand](#), [edgeAnimProcessBlendTree](#)



# edgeAnimLocalJointsToWorldMatrices3x4

Converts an array of local space joint transforms to world space 3x4 matrices.

## Definition

```
#include <edge/anim/edgeanim_spu.h>
void edgeAnimLocalJointsToWorldMatrices3x4 (
    void* outputMatrices,
    const EdgeAnimJointTransform* inputJoints,
    const EdgeAnimJointTransform* rootJoint,
    const int16_t* jointLinkage,
    uint32_t count
)
```

## Arguments

<i>outputMatrices</i>	Pointer to output world matrix array
<i>inputJoints</i>	Pointer to input local joint transform array
<i>rootJoint</i>	Pointer to root joint transform
<i>jointLinkage</i>	Pointer to single instruction, multiple data (SIMD) hierarchy
<i>count</i>	Number of entries in SIMD hierarchy

## Return Values

None.

## Description

This function converts an array of local space joint transforms to world space 3x4 matrices.

Unlike the [edgeAnimLocalJointsToWorldJoints\(\)](#) function, the calculation ensures that the local scale component of each joint in the hierarchy is correctly transformed in its own world orientation as opposed to the world orientation of the final joint.

Due to the SIMD nature of the processing, the size of the *outputMatrices* array must be a multiple of four entries.

For *jointLinkage*, pass the address of the *simdHierarchy* array in the [EdgeAnimSkeleton](#).

*count* specifies the number of entries in the hierarchy array, which can be calculated from the [EdgeAnimSkeleton](#) as  $4 * \text{numSimdHierarchyQuads}$ .

You can optionally use this function to perform parent scale compensation. If the parent joint's index has the appropriate bit set, the inverse of the parent joint's local-space scale is multiplied with the child's local-space scale to cancel out any scaling in the parent joint.

## Note

SPU and SSE implementations require all pointers to be 16-byte aligned.

## See Also

[EdgeAnimSkeleton](#)



# edgeAnimLocalJointsToWorldMatrices4x4

Converts an array of local space joint transforms to world space 4x4 matrices.

## Definition

```
#include <edge/anim/edgeanim_spu.h>
void edgeAnimLocalJointsToWorldMatrices4x4 (
    void* outputMatrices,
    const EdgeAnimJointTransform* inputJoints,
    const EdgeAnimJointTransform* rootJoint,
    const int16_t* jointLinkage,
    uint32_t count
)
```

## Arguments

<i>outputMatrices</i>	Pointer to output world matrix array
<i>inputJoints</i>	Pointer to input local joint transform array
<i>rootJoint</i>	Pointer to root joint transform
<i>jointLinkage</i>	Pointer to single instruction, multiple data (SIMD) hierarchy
<i>count</i>	Number of entries in SIMD hierarchy

## Return Values

None.

## Description

This function converts an array of local space joint transforms to world space 4x4 matrices.

Unlike the [edgeAnimLocalJointsToWorldJoints\(\)](#) function, the calculation ensures that the local scale component of each joint in the hierarchy is correctly transformed in its own world orientation as opposed to the world orientation of the final joint.

Due to the SIMD nature of the processing, the size of the *outputMatrices* array must be a multiple of four entries.

For *jointLinkage*, pass the address of the *simdHierarchy* array in the [EdgeAnimSkeleton](#).

*count* specifies the number of entries in the hierarchy array, which can be calculated from the [EdgeAnimSkeleton](#) as  $4 * \text{numSimdHierarchyQuads}$ .

This function can optionally perform parent scale compensation if the parent joint's index has the appropriate bit set. If set, the inverse of the parent joint's local-space scale is multiplied with the child's local-space scale to cancel out any scaling in the parent joint.

SPU and SSE implementations require all pointers to be 16-byte aligned.

## See Also

[EdgeAnimSkeleton](#)



# edgeAnimLocalJointsToWorldJoints

Converts an array of local space joint transforms to world space.

## Definition

```
#include <edge/anim/edgeanim_spu.h>
void edgeAnimLocalJointsToWorldJoints (
    EdgeAnimJointTransform* outputJoints,
    const EdgeAnimJointTransform* inputJoints,
    const EdgeAnimJointTransform* rootJoint,
    const int16_t* jointLinkage,
    uint32_t count
)
```

## Arguments

<i>outputJoints</i>	Pointer to output world joint transform array
<i>inputJoints</i>	Pointer to input local joint transform array
<i>rootJoint</i>	Pointer to root joint transform
<i>jointLinkage</i>	Pointer to single instruction, multiple data (SIMD) hierarchy
<i>count</i>	Number of entries in SIMD hierarchy

## Return Values

None.

## Description

This function converts an array of local space joint transforms to world space.

Due to the SIMD nature of the processing, the size of the *outputJoints* array must be a multiple of four entries.

For *jointLinkage*, pass the address of the *simdHierarchy* array in the [EdgeAnimSkeleton](#).

*count* specifies the number of entries in the hierarchy array, which can be calculated from the [EdgeAnimSkeleton](#) as  $4 * \text{numSimdHierarchyQuads}$ .

You can optionally use this function to perform parent scale compensation. If the parent joint's index has the appropriate bit set, the inverse of the parent joint's local-space scale is multiplied with the child's local-space scale to cancel out any scaling in the parent joint.

## Note

SPU and SSE implementations require all pointers to be 16-byte aligned.

## See Also

[EdgeAnimSkeleton](#)



# edgeAnimWorldJointsToLocalJoints

Converts an array of world space joint transforms to local space.

## Definition

```
#include <edge/anim/edgeanim_spu.h>
void edgeAnimWorldJointsToLocalJoints(
    EdgeAnimJointTransform* outputJoints,
    const EdgeAnimJointTransform* inputJoints,
    const EdgeAnimJointTransform* rootJoint,
    const int16_t* jointLinkage,
    uint32_t count
)
```

## Arguments

<i>outputJoints</i>	Pointer to output local joint transform array
<i>inputJoints</i>	Pointer to input world joint transform array
<i>rootJoint</i>	Pointer to root joint transform
<i>jointLinkage</i>	Pointer to single instruction, multiple data (SIMD) hierarchy
<i>count</i>	Number of entries in SIMD hierarchy

## Return Values

None.

## Description

This function converts an array of world space joint transforms to local space.

Due to the SIMD nature of the processing, the size of the *outputJoints* array must be a multiple of four entries.

For *jointLinkage*, pass the address of the *simdHierarchy* array in the skeleton.

*count* specifies the number of entries in the hierarchy array, which can be calculated from the skeleton as  $4 * \text{numSimdHierarchyQuads}$ .



---

# edgeAnimJointsToMatrices4x4

---

Converts an array of joint transforms to 4x4 matrices.

## Definition

---

```
#include <edge/anim/edgeanim_spu.h>
void edgeAnimJointsToMatrices4x4(
    Vectormath::Aos::Transform3* outputMatrices,
    const EdgeAnimJointTransform* inputJoints,
    uint32_t count
)
```

## Arguments

---

<i>outputMatrices</i>	Pointer to output matrix array
<i>inputJoints</i>	Pointer to input joint transform array
<i>count</i>	Joint count

## Return Values

---

None.

## Description

---

This function converts an array of joint transforms to 4x4 matrices.

Due to the SIMD nature of the processing, the size of the *outputMatrices* array must be a multiple of four entries.

## Note

---

SPU and SSE implementations require all pointers to be 16-byte aligned.



---

# edgeAnimMatrices4x4ToJoints

---

Converts an array of 4x4 matrices to joint transforms.

## Definition

---

```
#include <edge/anim/edgeanim_spu.h>
void edgeAnimMatrices4x4ToJoints(
    EdgeAnimJointTransform* outputJoints,
    const Vectormath::Aos::Transform3* inputMatrices,
    uint32_t count
)
```

## Arguments

---

<i>outputJoints</i>	Pointer to output joint transform array
<i>inputMatrices</i>	Pointer to input matrix array
<i>count</i>	Joint count

## Return Values

---

None.

## Description

---

This function converts an array of 4x4 matrices to joint transforms.

Due to the SIMD nature of the processing, the size of the *outputJoints* array must be a multiple of four entries.

## Note

---

SPU and SSE implementations require all pointers to be 16-byte aligned.



---

# edgeAnimJointsToMatrices3x4

---

Converts an array of joint transforms to 3x4 matrices.

## Definition

---

```
#include <edge/anim/edgeanim_spu.h>
void edgeAnimJointsToMatrices3x4(
    void* outputMatrices,
    const EdgeAnimJointTransform* inputJoints,
    uint32_t count
)
```

## Arguments

---

<i>outputMatrices</i>	Pointer to output matrix array
<i>inputJoints</i>	Pointer to input joint transform array
<i>count</i>	Joint count

## Return Values

---

None.

## Description

---

This function converts an array of joint transforms to 3x4 matrices.

Due to the SIMD nature of the processing, the size of the *outputMatrices* array must be a multiple of four entries.

## Note

---

SPU and SSE implementations require all pointers to be 16-byte aligned.



---

# edgeAnimMatrices3x4ToJoints

---

Converts an array of 3x4 matrices to joint transforms.

## Definition

---

```
#include <edge/anim/edgeanim_spu.h>
void edgeAnimMatrices3x4ToJoints(
    EdgeAnimJointTransform* outputJoints,
    const void* inputMatrices,
    uint32_t count
)
```

## Arguments

---

<i>outputJoints</i>	Pointer to output joint transform array
<i>inputMatrices</i>	Pointer to input matrix array
<i>count</i>	Joint count

## Return Values

---

None.

## Description

---

This function converts an array of 3x4 matrices to joint transforms.

Due to the SIMD nature of the processing, the size of the *outputJoints* array must be a multiple of four entries.

## Note

---

SPU and SSE implementations require all pointers to be 16-byte aligned.



# edgeAnimBlendJointsLinear

Performs weighted blend between two poses.

## Definition

```
#include <edge/anim/edgeanim_spu.h>
void edgeAnimBlendJointsLinear(
    EdgeAnimJointTransform* outputJoints,
    uint8_t* outputWeights,
    const EdgeAnimJointTransform* leftJoints,
    const uint8_t* leftWeights,
    const EdgeAnimJointTransform* rightJoints,
    const uint8_t* rightWeights,
    float alpha,
    uint32_t count
)
```

## Arguments

<i>outputJoints</i>	Pointer to output joint transform array
<i>outputWeights</i>	Pointer to output joint weights array
<i>leftJoints</i>	Pointer to left joint transform array
<i>leftWeights</i>	Pointer to left joint weights array
<i>rightJoints</i>	Pointer to right joint transform array
<i>rightWeights</i>	Pointer to right joint weights array
<i>alpha</i>	Blend factor (0-1)
<i>count</i>	Joint count

## Return Values

None.

## Description

This function performs weighted blend between two poses. It is called internally by [edgeAnimProcessCommandList\(\)](#) and [edgeAnimProcessBlendTree\(\)](#).

- Rotations are blended using spherical linear interpolation.
- Translation and scale are blended using linear interpolation.

Partial animation logic, described in the *PlayStation®Edge Library Overview*, is summarized in the following table:

Joints Defined	Output
None	Undefined
Left	Left
Right	Undefined
Left & Right	Blend(Left, Right)

## Notes

Due to the SIMD nature of the processing, the size of the *outputJoints* array must be a multiple of four entries.

*alpha* must be in the [0..1] range. Behavior is undefined outside of this range.

If *leftWeights* is set to NULL, all left weights are assumed to be 0xFF (1.0).



If *rightWeights* is set to `NULL`, all right weights are assumed to be `0xFF` (1.0).

A higher-level interface is available with [edgeAnimBlendPose\(\)](#).

SPU and SSE implementations require the joint pointers to be 16-byte aligned and the weight pointers to be 4-byte aligned.

#### **See Also**

---

[edgeAnimBlendPose](#), [edgeAnimBlendUserLinear](#)



# edgeAnimBlendJointsRelative

Performs weighted additive/subtractive blending and composition between two poses, depending on the value of *blendMode*.

## Definition

```
#include <edge/anim/edgeanim_spu.h>
void edgeAnimBlendJointsRelative(
    EdgeAnimJointTransform* outputJoints,
    uint8_t* outputWeights,
    const EdgeAnimJointTransform* baseJoints,
    const uint8_t* baseWeights,
    const EdgeAnimJointTransform* deltaJoints,
    const uint8_t* deltaWeights,
    float alpha,
    EdgeAnimRelativeBlendMode blendMode,
    uint32_t count
)
```

## Arguments

<i>outputJoints</i>	Pointer to output joint transform array
<i>outputWeights</i>	Pointer to output joint weights array
<i>baseJoints</i>	Pointer to base joint transform array
<i>baseWeights</i>	Pointer to base joint weights array
<i>deltaJoints</i>	Pointer to delta joint transform array
<i>deltaWeights</i>	Pointer to delta joint weights array
<i>alpha</i>	Blend factor (0-1)
<i>blendMode</i>	Relative blend/compose mode (see below)
<i>count</i>	Joint count

## Return Values

None.

## Description

This function is used to perform weighted additive/subtractive blending and composition between two poses, depending on the value of *blendMode*. It is called internally by [edgeAnimProcessCommandList\(\)](#) and [edgeAnimProcessBlendTree\(\)](#).

### “Delta” Blend Modes:

```
blendMode = EDGE_ANIM_RELATIVE_BLEND_ADD_DELTA
```

In these modes, this function performs weighted blend between two poses, either left and left+right.

- Rotations are blended using spherical linear interpolation.
- Translation and scale are blended using linear interpolation.

### “Compose” Modes:

```
blendMode = EDGE_ANIM_RELATIVE_COMPOSE_ADD
blendMode = EDGE_ANIM_RELATIVE_COMPOSE_SUB
```

In these modes, this function composes, without blending (alpha is ignored), either left+right(ADD) or left-right(SUB). The major difference is the behavior if the left joint is undefined.



---

Partial animation logic, described in the *PlayStation®Edge Library Overview*, is summarized in the following table:

Defined	BLEND_ADD_DELTA	COMPOSE_ADD	COMPOSE_SUB
None	Undefined	Undefined	Undefined
Left	Left	Left	Undefined
Right	Undefined	Right	Undefined
Both	Blend(Left, Left+Right)	Left+Right	Left-Right

## Notes

---

Due to the SIMD nature of the processing, the size of the *outputJoints* array must be a multiple of four entries.

*alpha* must be in the [0..1] range. Behavior is undefined outside of this range.

If *leftWeights* is set to NULL, all left weights are assumed to be 0xFF (1.0).

If *rightWeights* is set to NULL, all right weights are assumed to be 0xFF (1.0).

A higher-level interface is available with [edgeAnimBlendPose\(\)](#).

SPU and SSE implementations require the joint pointers to be 16-byte aligned and the weight pointers to be 4-byte aligned.

## See Also

---

[edgeAnimBlendPose](#), [edgeAnimBlendUserRelative](#)



# edgeAnimBlendPose

---

Performs a blend between slots (both joints and user channels) of the pose stack.

## Definition

---

```
#include <edge/anim/edgeanim_spu.h>
void edgeAnimBlendPose (
    EdgeAnimSpuContext* spuContext,
    unsigned int poseDestDepth,
    unsigned int poseLeftDepth,
    unsigned int poseRightDepth,
    EdgeAnimBlendOp blendOp,
    float alpha,
    const EdgeAnimSkeleton* skeleton
)
```

## Arguments

---

<i>spuContext</i>	Pointer to SPU context
<i>poseDestDepth</i>	Depth of destination pose in the stack
<i>poseLeftDepth</i>	Depth of left pose in the stack
<i>poseRightDepth</i>	Depth of right pose in the stack
<i>blendOp</i>	Blend operation
<i>alpha</i>	Blend factor (0-1)
<i>skeleton</i>	Pointer to skeleton definition

## Return Values

---

None.

## Description

---

This function performs a blend between slots (both joints and user channels) of the pose stack.

## Notes

---

If any DMA operation is currently affecting any of these slots, the function waits until DMA is completed

Slots can alias - the *poseDestDepth* can either be the same as *poseLeftDepth* or *poseRightDepth* to do in-place processing.

## See Also

---

[edgeAnimBlendJointsLinear](#), [edgeAnimBlendJointsRelative](#), [edgeAnimBlendUserLinear](#), [edgeAnimBlendUserRelative](#)



# edgeAnimBlendUserLinear

Performs linear blending between two user channel arrays.

## Definition

```
#include <edge/anim/edgeanim_spu.h>
void edgeAnimBlendUserLinear(
    float* outputChannels,
    uint8_t* outputWeights,
    const float* leftChannels,
    const uint8_t* leftWeights,
    const float* rightChannels,
    const uint8_t* rightWeights,
    const uint8_t* channelFlags,
    float alpha,
    uint32_t count
)
```

## Arguments

<i>outputChannels</i>	Pointer to output user channel array
<i>outputWeights</i>	Pointer to output weights array
<i>leftChannels</i>	Pointer to left user channel array
<i>leftWeights</i>	Pointer to left weights array
<i>rightChannels</i>	Pointer to right user channel array
<i>rightWeights</i>	Pointer to right weights array
<i>channelFlags</i>	Pointer to channel flags array
<i>alpha</i>	Blend factor (0-1)
<i>count</i>	User channel count

## Return Values

None.

## Description

This function performs linear blending between two user channel arrays. It is called internally by [edgeAnimProcessCommandList\(\)](#) and [edgeAnimProcessBlendTree\(\)](#).

Flags defined in *channelFlags* can alter the behavior of this blend function per-channel:

- If `EDGE_ANIM_USER_CHANNEL_FLAG_CLAMP01` is set, output value is clamped to the `[0.0f...1.0f]` range.
- Other flags are ignored.

Partial animation logic, described in the *PlayStation®Edge Library Overview*, is summarized in the following table:

Joins Defined	Output
None	Undefined
Left	Left
Right	Undefined
Left & Right	Blend(Left, Right)



**Notes**

---

Due to the SIMD nature of the processing, the size of the *outputChannels* array must be a multiple of four entries.

*alpha* must be in the [0..1] range. Behavior is undefined outside of this range.

If *leftWeights* is set to NULL, all left weights are assumed to be 0xFF (1.0).

If *rightWeights* is set to NULL, all right weights are assumed to be 0xFF (1.0).

If *channelFlags* is set to NULL, all flags are assumed to be 0x00 (the default behavior).

A higher-level interface is available with [edgeAnimBlendPose\(\)](#).

SPU and SSE implementations require the joint pointers to be 16-byte aligned and the weight pointers to be 4-byte aligned.

**See Also**

---

[edgeAnimBlendPose](#), [edgeAnimBlendJointsLinear](#)



# edgeAnimBlendUserRelative

Performs additive blending between two user channel arrays.

## Definition

```
#include <edge/anim/edgeanim_spu.h>
void edgeAnimBlendUserRelative(
    float* outputChannels,
    uint8_t* outputWeights,
    const float* leftChannels,
    const uint8_t* leftWeights,
    const float* rightChannels,
    const uint8_t* rightWeights,
    const uint8_t* channelFlags,
    float alpha,
    EdgeAnimRelativeBlendMode blendMode,
    uint32_t count
)
```

## Arguments

<i>outputChannels</i>	Pointer to output user channel array
<i>outputWeights</i>	Pointer to output weights array
<i>leftChannels</i>	Pointer to base user channel array
<i>leftWeights</i>	Pointer to base weights array
<i>rightChannels</i>	Pointer to delta user channel array
<i>rightWeights</i>	Pointer to delta weights array
<i>channelFlags</i>	Pointer to channel flags array
<i>alpha</i>	Blend factor (0-1)
<i>blendMode</i>	Relative blend/compose mode
<i>count</i>	User channel count

## Return Values

None.

## Description

This function performs additive blending between two user channel arrays. It is called internally by [edgeAnimProcessCommandList\(\)](#) and [edgeAnimProcessBlendTree\(\)](#).

Flags defined on *channelFlags* can alter the behavior of this blend function per-channel:

- If `EDGE_ANIM_USER_CHANNEL_FLAG_CLAMP01` is set, output value is clamped to the [0.0f...1.0f] range.
- If `EDGE_ANIM_USER_CHANNEL_FLAG_MINMAX` is set, all addition and subtraction operations in “compose” mode are replaced respectively by max and min operations.
- Other flags are ignored.

### “Delta” Blend Modes:

```
blendMode = EDGE_ANIM_RELATIVE_BLEND_ADD_DELTA
```

In this mode, this function performs weighted blend between two poses, either left and left+right.

- Rotations are blended using spherical linear interpolation.
- Translation and scale are blended using linear interpolation.



**“Compose” Modes:**

```
blendMode = EDGE_ANIM_RELATIVE_COMPOSE_ADD
blendMode = EDGE_ANIM_RELATIVE_COMPOSE_SUB
```

In this mode, this function composes, without blending (alpha is ignored), either left+right (ADD) or left-right (SUB). The major difference is the behavior if the left joint is undefined.

Partial animation logic, described in the *PlayStation®Edge Library Overview*, is summarized in the following table:

Defined	BLEND_ADD_DELTA	COMPOSE_ADD	COMPOSE_SUB
None	Undefined	Undefined	Undefined
Left	Left	Left	Undefined
Right	Undefined	Right	Undefined
Both (1)	Blend(Left, Left+Right)	Left+Right	Left-Right
Both (2)	Blend(Left, Left+Right)	Max(Left,Right)	Min(Left,Right)

(1) “Default” mode – `EDGE_ANIM_USER_CHANNEL_FLAG_MINMAX` is not set.

(2) “Fuzzy” mode – `EDGE_ANIM_USER_CHANNEL_FLAG_MINMAX` is set.

**Notes**

Due to the SIMD nature of the processing, the size of the *outputChannels* array must be a multiple of four entries.

*alpha* must be in the [0..1] range. Behavior is undefined outside of this range.

If *leftWeights* is set to `NULL`, all left weights are assumed to be `0xFF` (1.0).

If *rightWeights* is set to `NULL`, all right weights are assumed to be `0xFF` (1.0).

If *channelFlags* is set to `NULL`, all flags are assumed to be `0x00` (the default behavior).

A higher-level interface is available with [edgeAnimBlendPose\(\)](#).

SPU and SSE implementations require the channel pointers to be 16-byte aligned and the weight pointers to be 4-byte aligned.

**See Also**

[edgeAnimBlendPose](#), [edgeAnimBlendJointsRelative](#)



---

# edgeAnimPoseStackPush

---

Adds one pose to the top of the stack.

## Definition

---

```
#include <edge/anim/edgeanim_spu.h>
void edgeAnimPoseStackPush (
    EdgeAnimSpuContext* spuContext
)
```

## Arguments

---

<i>spuContext</i>	Pointer to SPU context
-------------------	------------------------

## Return Values

---

None.

## Description

---

This function adds one pose to the top of the stack. If the local store stack space is full, it starts a DMA of the least recently used pose to the main memory external storage area (if any) associated with this SPU. This function asserts if there is insufficient space remaining in both local store and main memory.

## See Also

---

[edgeAnimPoseStackPop](#)



---

# edgeAnimPoseStackPop

---

Discards the pose at the top of the stack.

## Definition

---

```
#include <edge/anim/edgeanim_spu.h>
void edgeAnimPoseStackPop (
    EdgeAnimSpuContext* spuContext
)
```

## Arguments

---

<i>spuContext</i>	Pointer to SPU context
-------------------	------------------------

## Return Values

---

None.

## Description

---

This function discards the pose at the top of the stack. If any poses are stored in main memory, it uses DMA to write the most recently used pose to local store.

## See Also

---

[edgeAnimPoseStackPush](#)



---

# edgeAnimPoseStackGetPose

---

Retrieves the information associated with the pose at *depth* on the pose stack.

## Definition

---

```
#include <edge/anim/edgeanim_spu.h>
void edgeAnimPoseStackGetPose (
    EdgeAnimSpuContext* spuContext,
    EdgeAnimPoseInfo* pose,
    uint32_t depth
)
```

## Arguments

---

<i>spuContext</i>	Pointer to SPU context
<i>pose</i>	Address of pose information structure, which is filled by this function
<i>depth</i>	Stack depth of pose

## Return Values

---

None.

## Description

---

This function retrieves the information associated with the pose at *depth* on the pose stack.

## See Also

---

[EdgeAnimPoseInfo](#)



---

# edgeAnimCustomDataChunk

---

Retrieves a custom data entry from the specified custom data table, identified by the entry's hash name.

## Definition

---

```
#include <edge/anim/edgeanim_spu.h>
void* edgeAnimCustomDataChunk(
    const EdgeAnimCustomDataTable* pCustomDataTable,
    uint32_t chunkHash
)
```

## Arguments

---

<i>pCustomDataTable</i>	Pointer to custom data table
<i>chunkHash</i>	Hash name of the requested custom data entry

## Return Values

---

Returns a pointer to the data entry corresponding to the requested hash name.  
Returns NULL if the hash name does not exist in the table.

## Description

---

This function retrieves a custom data entry from the specified custom data table, identified by the entry's hash name. The pointer to the custom data table may be determined from the offset values in the [EdgeAnimSkeleton](#) and [EdgeAnimAnimation](#) structures.

## See Also

---

[EdgeAnimCustomDataTable](#)



# Callback Functions



# EdgeAnimLeafCallback

Called at each of the three pipeline stages of leaf processing.

## Definition

```
#include <edge/anim/edgeanim_spu.h>
void ( *EdgeAnimLeafCallback )(
    EdgeAnimSpuContext* spuContext,
    const EdgeAnimBlendLeaf const* leaf,
    const EdgeAnimAnimation* anim,
    const EdgeAnimSkeleton const* skel,
    const int32_t pipelineStage,
    const uint32_t dmaTag,
    void* userScratchBuffer
)
```

## Arguments

<i>spuContext</i>	Pointer to SPU context
<i>leaf</i>	Pointer to blend tree leaf
<i>anim</i>	Pointer to animation
<i>skel</i>	Pointer to skeleton
<i>pipelineStage</i>	Current pipeline stage (-2, -1, or 0)
<i>dmaTag</i>	User DMA tag
<i>userScratchBuffer</i>	Pointer to user scratch buffer space

## Return Values

None.

## Description

This callback function is called at each of the three pipeline stages of leaf processing. To register this callback, pass it as the *leafCallback* argument to [edgeAnimProcessBlendTree\(\)](#).

*leaf* contains a pointer to the currently processed leaf.

*anim* contains the pointer to the animation header if pipeline stage is 0; otherwise it contains NULL.

*pipelineStage* contains the current pipeline stage of the leaf. The possible values are (in order of occurrence) -2, -1, and 0. Stages -2 and -1 occur before evaluation and can be used to prefetch data. At stage 0 the leaf has been processed and the evaluated pose is accessible as the top pose on the pose stack.

*userScratchBuffer* points to the user memory associated with the current leaf, or NULL if none has been specified. The user scratch buffer size is specified in the call to [edgeAnimSpuInitialize\(\)](#).

*dmaTag* can be used to initiate DMAs to or from the user scratch buffer. It is the responsibility of the application to stall for any user DMAs that were started at an earlier pipeline stage, before reading the results.

## See Also

[edgeAnimProcessBlendTree](#), [EdgeAnimBranchCallback](#), [edgeAnimSpuInitialize](#)



# EdgeAnimBranchCallback

Called at each of the three pipeline stages of branch processing.

## Definition

```
#include <edge/anim/edgeanim_spu.h>
void ( *EdgeAnimBranchCallback) (
    EdgeAnimSpuContext* spuContext,
    const EdgeAnimBlendBranch const* branch,
    const EdgeAnimSkeleton const* skel,
    const int32_t pipelineStage,
    const uint32_t dmaTag,
    void* userScratchBuffer
)
```

## Arguments

<i>spuContext</i>	Pointer to SPU context
<i>branch</i>	Pointer to blend tree branch
<i>skel</i>	Pointer to skeleton
<i>pipelineStage</i>	Current pipeline stage (-2, -1, or 0)
<i>dmaTag</i>	User DMA tag
<i>userScratchBuffer</i>	Pointer to user scratch buffer space

## Return Values

None.

## Description

This callback function is called at each of the three pipeline stages of branch processing. To register this callback, pass it as the *branchCallback* argument to [edgeAnimProcessBlendTree\(\)](#).

*branch* contains a pointer to the currently processed branch.

*pipelineStage* contains the current pipeline stage of the branch. The possible values are (in order of occurrence) -2, -1, and 0. Stages -2 and -1 occur before blending and can be used to prefetch data. At stage 0 the branch has been processed and the blended pose is accessible as the top pose on the pose stack.

*userScratchBuffer* points to the user memory associated with the current branch, or NULL if none has been specified. The user scratch buffer size is specified in the call to [edgeAnimSpuInitialize\(\)](#).

*dmaTag* can be used to initiate DMAs to/from the user scratch buffer. It is the responsibility of the application to stall for any user DMAs that were started at an earlier pipeline stage, before reading the results.

## See Also

[edgeAnimProcessBlendTree](#), [EdgeAnimLeafCallback](#), [edgeAnimSpuInitialize](#)



# EdgeAnimUserCallback

Called at each of the three pipeline stages of command processing for any unrecognized command ID.

## Definition

```
#include <edge/anim/edgeanim_spu.h>
void ( *EdgeAnimUserCallback) (
    EdgeAnimSpuContext* spuContext,
    const EdgeAnimCommand const* cmd,
    const EdgeAnimSkeleton* skel,
    const int32_t pipelineStage,
    const uint32_t dmaTag,
    void* userScratchBuffer
)
```

## Arguments

<i>spuContext</i>	Pointer to SPU context
<i>cmd</i>	Pointer to the unrecognized user command
<i>skel</i>	Pointer to skeleton
<i>pipelineStage</i>	Current pipeline stage (-2, -1, or 0)
<i>dmaTag</i>	User DMA tag
<i>userScratchBuffer</i>	Pointer to user scratch buffer space

## Return Values

None.

## Description

This callback function is called at each of the three pipeline stages of command processing for any unrecognized command ID. To register this callback, pass it as the *userCallback* argument to [edgeAnimProcessCommandList\(\)](#).

*cmd* contains a pointer to the unrecognized user command.

*pipelineStage* contains the current pipeline stage of the branch. The possible values are (in order of occurrence) -2, -1, and 0. Stages -2 and -1 occur before blending and can be used to prefetch data. At stage 0, the branch has been processed and the blended pose is accessible as the top pose on the pose stack.

*userScratchBuffer* points to the user memory associated with the current command, or NULL if none has been specified. The user scratch buffer size is specified in the call to [edgeAnimSpuInitialize\(\)](#).

*dmaTag* can be used to initiate DMAs to/from the user scratch buffer. It is the responsibility of the application to stall for any user DMAs that were started at an earlier pipeline stage, before reading the results.

## See Also

[edgeAnimProcessCommandList](#), [edgeAnimSpuInitialize](#)



# Constants



# EdgeAnimBlendOp

Constants expressing the various different modes for blending operation.

## Definition

Macro	Value	Description
EDGE_ANIM_BLENDOP_BLEND_LINEAR	0x000	Blend between: Left [ $\alpha = 0.0$ ] Right [ $\alpha = 1.0$ ]
EDGE_ANIM_BLENDOP_BLEND_ADD_DELTA_RIGHT	0x001	Blend between: Left [ $\alpha = 0.0$ ] Add(Left, Right) [ $\alpha = 1.0$ ]
EDGE_ANIM_BLENDOP_BLEND_ADD_DELTA_LEFT	0x010	Blend between: Right [ $\alpha = 0.0$ ] Add(Right, Left) [ $\alpha = 1.0$ ]
EDGE_ANIM_BLENDOP_BLEND_SUB_DELTA_RIGHT	0x011	Blend between: Left [ $\alpha = 0.0$ ] Sub(Left, Right) [ $\alpha = 1.0$ ]
EDGE_ANIM_BLENDOP_BLEND_SUB_DELTA_LEFT	0x100	Blend between: Right [ $\alpha = 0.0$ ] Sub(Right, Left) [ $\alpha = 1.0$ ]
EDGE_ANIM_BLENDOP_COMPOSE_ADD	0x101	Compose: Add(Left,Right) [both defined] Otherwise Left or Right
EDGE_ANIM_BLENDOP_COMPOSE_SUB_RIGHT	0x110	Compose: Sub(Left,Right) [both defined] Otherwise Left or Inverse(Right)
EDGE_ANIM_BLENDOP_COMPOSE_SUB_LEFT	0x111	Compose: Sub(Right, Left) [both defined] Otherwise Right or Inverse(Left)

## Description

This is passed to [edgeAnimBlendPose\(\)](#) in *blendOp* argument.



---

# EdgeAnimRelativeBlendMode

---

Constants expressing the various different modes for relative-blending operation.

## Definition

---

Macro	Value	Description
EDGE_ANIM_RELATIVE_BLEND_ADD_DELTA	0x000	Blend between "Base" and "Base +Delta"
EDGE_ANIM_RELATIVE_BLEND_SUB_DELTA	0x001	Blend between "Base" and "Base - Delta"
EDGE_ANIM_RELATIVE_COMPOSE_ADD	0x010	Result "Base + delta" No blending preformed.
EDGE_ANIM_RELATIVE_COMPOSE_SUB	0x011	Result "Base - delta" No blending preformed.
EDGE_ANIM_RELATIVE_NONE	0x100	No operation. (Undefined)

## Description

---

This is passed to [edgeAnimBlendJointsRelative\(\)](#) and [edgeAnimBlendUserRelative\(\)](#) in the *blendMode* argument.