

# **PlayStation®Edge Animation Library for Offline Tool: Reference**

---

## Table of Contents

---

<b>Preface.....</b>	<b>3</b>
About This Document.....	4
<b>Data Types for Tools.....</b>	<b>5</b>
Joint.....	6
AnimationKeyframe .....	7
JointAnimation.....	8
UserChannelAnimation .....	9
Animation .....	10
UserChannelInfo .....	11
Skeleton .....	12
CompressionInfo .....	13
<b>libedgeanimtool Functions .....</b>	<b>15</b>
ComputePeriod .....	16
GenerateAdditiveAnimation .....	17
ExportAnimation .....	18
ExtractSkeleton .....	19
ExportSkeleton .....	20
ParseUserChannels .....	21
<b>Custom Data.....</b>	<b>22</b>
CustomDataCallback.....	23
CustomDataTable.....	24

# Preface

---

# About This Document

---

## Purpose

This document provides an API reference for the offline-tool animation component of the Edge library, `libedgeanimtool`. Integrate this component into your asset-pipeline to efficiently manage the data and to maintain high performance in the runtime.

The Edge animation compiler sample tool, `edgeanimcompiler`, is a sample tool to demonstrate how `libedgeanimtool` can be used in an asset pipeline. Specifically, it demonstrates how to take data in COLLADA™ format and pass it through the Edge library, producing a binary file that can be loaded into the runtime sample. Due to the usage of FCollada, `edgeanimcompiler` now only supports Windows.

## 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 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
<i>fixed-width font</i>	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><i>fixed-width font + bold</i></b>	In structure/function definitions only, indicates names of structures and functions.
<i>fixed-width font + italics</i>	Indicates arguments, parameters, and variables.
<u>blue + underlined text</u>	Indicates a hyperlink (blue displays in color printers or online only).

# Data Types for Tools

---

# Joint

---

Describes a single joint transform.

## Definition

---

```
#include <edge/anim/edgeanim_structs.h>
struct Joint
{
    Float4 m_rotation;
    Float4 m_translation;
    Float4 m_scale;
};
```

## Members

---

<i>m_rotation</i>	Rotational part of transform (quaternion).
<i>m_translation</i>	Translational part of transform.
<i>m_scale</i>	Scaling part of transform.

## Description

---

This structure describes a single joint transform.

---

# AnimationKeyframe

---

Structure containing a single keyframe.

## Definition

---

```
#include <libedgeanimtool_animation.h>
struct AnimationKeyframe
{
    float m_keyTime;
    Float4 m_keyData;
};
```

## Members

---

<i>m_keyTime</i>	Keyframe time (seconds).
<i>m_keyData</i>	The keyframe data.

## Description

---

This structure contains the data for a single keyframe for a channel. It can represent a rotation, translation, scale, or user channel value. In the case of a user channel, only the first entry of *m\_keyData* is used and the remaining entries are set to zero.

---

# JointAnimation

---

Structure that contains the animation of a single joint.

## Definition

---

```
#include <libedgeanimtool_animation.h>
struct JointAnimation
{
    unsigned m_jointName;
    float m_jointWeight;
    std::vector<AnimationKeyframe> m_rotationAnimation;
    std::vector<AnimationKeyframe> m_translationAnimation;
    std::vector<AnimationKeyframe> m_scaleAnimation;
};
```

## Members

---

<i>m_jointName</i>	Hashed name of joint.
<i>m_jointWeight</i>	Joint weight (0 to 1).
<i>m_rotationAnimation</i>	Rotation keyframe array (in order of increasing time).
<i>m_translationAnimation</i>	Translation keyframe array (in order of increasing time).
<i>m_scaleAnimation</i>	Scale keyframe array (in order of increasing time).

## Description

---

This structure describes the animation of a single joint.

*m\_jointWeight* specifies the weight used for weighted blending in the runtime.

## See Also

---

[AnimationKeyframe](#)



---

# UserChannelAnimation

---

Structure that contains the animation of a single user channel.

## Definition

---

```
#include <libedgeanimtool_animation.h>
struct UserChannelAnimation
{
    unsigned m_nodeName;
    unsigned m_channelName;
    float m_weight;
    std::vector<AnimationKeyframe> m_animation;
};
```

## Members

---

<i>m_nodeName</i>	Hashed name of channel's parent node.
<i>m_channelName</i>	Hashed name of channel.
<i>m_weight</i>	Channel weight (0 to 1).
<i>m_animation</i>	Keyframe array (in order of increasing time).

## Description

---

This structure describes the animation of a single user channel.

*m\_weight* specifies the weight used for weighted blending in the runtime.

## See Also

---

[AnimationKeyframe](#)

# Animation

---

Structure that describes the animation of a set of joints and/or user channels.

## Definition

---

```
#include <libedgeanimtool_animation.h>
struct Animation
{
    float m_startTime;
    float m_endTime;
    float m_period;
    unsigned m_numFrames;
    bool m_enableLocoDelta;
    Float4 m_locoDeltaQuat;
    Float4 m_locoDeltaTrans;
    std::vector<JointAnimation> m_jointAnimations;
    std::vector<UserChannelAnimation> m_userChannelAnimations;
};
```

## Members

---

<i>m_startTime</i>	Start time of the animation (seconds).
<i>m_endTime</i>	End time of the animation (seconds).
<i>m_period</i>	Sampling period of the animation (seconds).
<i>m_numFrames</i>	Total number of frames.
<i>m_enableLocoDelta</i>	If true, locomotion deltas will be output.
<i>m_locoDeltaQuat</i>	Locomotion rotation delta (end-start).
<i>m_locoDeltaTrans</i>	Locomotion translation delta (end-start).
<i>m_jointAnimations</i>	Array of joint animations.
<i>m_userChannelAnimations</i>	Array of user-channel animations.

## Description

---

This structure describes the animation of a set of joints and/or user channels.

*m\_numFrames* is the total number of frames at the initial sampling frequency (for example, 60 for a two-second clip at 30hz).

## See Also

---

[JointAnimation](#), [UserChannelAnimation](#)

# UserChannelInfo

Structure that contains information about a user channel.

## Definition

```
#include <libedgeanimtool_skeleton.h>
struct UserChannelInfo
{
    unsigned int m_nodeNameHash;
    unsigned int m_channelNameHash;
    unsigned char m_componentIndex;
    unsigned char m_flags;
};
```

## Members

<i>m_nodeNameHash</i>	Hashed name of channel's parent node.
<i>m_channelNameHash</i>	Hashed name of channel.
<i>m_componentIndex</i>	Component index (0 to 3).
<i>m_flags</i>	Flags (see below).

A combination of the following can be set for *m\_flags*:

Macro	Value	Description
EDGE_ANIM_USER_CHANNEL_FLAG_CLAMP01	0	Channel is clamped to range (0,1).
EDGE_ANIM_USER_CHANNEL_FLAG_MINMAX	1	Channel is of type min/max.

## Description

This structure contains information about a single user channel.

The hashed names are used to uniquely identify the channel; *m\_nodeNameHash* is the name of the parent node of the animated attribute, and *m\_channelNameHash* is the name of the attribute itself. For example, the channel could represent a blend shape weight called "Frown" that is a child of the node "FaceShapes".

*m\_componentIndex* is used to identify the component associated with this channel. For example, if the channel represents the z component of a vector attribute, this should be set to 2. For scalar attributes, it should be set to 0.

*m\_flags* affects the runtime blending behavior – see the runtime documentation for details.

# Skeleton

Structure describing a hierarchy of joints and/or a set of user channels.

## Definition

```
#include <libedgeanimtool_skeleton.h>
struct Skeleton
{
    unsigned int m_locoJointIndex;
    unsigned int m_numJoints;
    unsigned int m_numUserChannels;
    std::vector<int> m_parentIndices;
    std::vector<Joint> m_basePose;
    std::vector<bool> m_scaleCompensateFlags;
    std::vector<unsigned int> m_jointNameHashes;
    std::vector<UserChannelInfo> m_userChannelInfoArray;
};
```

## Members

<i>m_locoJointIndex</i>	Index of the joint used for locomotion.
<i>m_numJoints</i>	Joint count.
<i>m_numUserChannels</i>	User-channel count.
<i>m_parentIndices</i>	Array of parent indices per joint (an index of -1 indicates no parent).
<i>m_basePose</i>	Array of transforms per joint defining the base pose of the skeleton.
<i>m_scaleCompensateFlags</i>	Array of scale compensate flags per joint.
<i>m_jointNameHashes</i>	Array of hashed names per joint.
<i>m_userChannelInfoArray</i>	Array of structures describing each user channel.

## Description

This structure describes a hierarchy of joints and/or a set of user channels.

If a joint's entry in *m\_scaleCompensateFlags* is true, it indicates that the joint will not take into account its parent's scale when it is transformed into parent space.

## See Also

[Joint](#), [UserChannelInfo](#)

# CompressionInfo

Structure used to specify channel compression modes.

## Definition

```
#include <libedgeanimtool_animation.h>
struct CompressionInfo
{
    EdgeAnimCompressionType m_compressionTypeRotation;
    EdgeAnimCompressionType m_compressionTypeTranslation;
    EdgeAnimCompressionType m_compressionTypeScale;
    EdgeAnimCompressionType m_compressionTypeUser;
    float m_defaultToleranceRotation,
    float m_defaultToleranceTranslation,
    float m_defaultToleranceScale,
    float m_defaultToleranceUser,
    std::vector<float> m_jointTolerancesRotation,
    std::vector<float> m_jointTolerancesTranslation,
    std::vector<float> m_jointTolerancesScale,
    std::vector<float> m_userChannelTolerances
};
```

## Members

<i>m_compressionTypeRotation</i>	Compression type used for rotations (see below).
<i>m_compressionTypeTranslation</i>	Compression type used for translations (see below).
<i>m_compressionTypeScale</i>	Compression type used for scales (see below).
<i>m_compressionTypeUser</i>	Compression type used for user channels (see below).
<i>m_defaultToleranceRotation</i>	Default bit-packed rotation tolerance.
<i>m_defaultToleranceTranslation</i>	Default bit-packed translation tolerance.
<i>m_defaultToleranceScale</i>	Default bit-packed scale tolerance.
<i>m_defaultToleranceUser</i>	Default bit-packed user channel tolerance.
<i>m_jointTolerancesRotation</i>	Per-joint rotation tolerances.
<i>m_jointTolerancesTranslation</i>	Per-joint translation tolerances.
<i>m_jointTolerancesScale</i>	Per-joint scale tolerances.
<i>m_userChannelTolerances</i>	Per-joint user channel tolerances.

The following values can be set for *compressionTypeXXXX*:

Macro	Value	Description
COMPRESSION_TYPE_NONE	0	No compression.
COMPRESSION_TYPE_SMALLEST_3	1	Smallest-Three compression.
COMPRESSION_TYPE_BITPACKED	2	Bit-packed compression.

## Description

This structure is used to specify how individual channel types should be compressed and, in the case of bit-packing, what error tolerances should be used. It is filled in by the user and passed as an argument to [ExportAnimation\(\)](#).

Note that some compression types are not supported/valid for certain channels. For rotations, COMPRESSION\_TYPE\_NONE is not supported. For non-rotations, COMPRESSION\_TYPE\_SMALLEST\_3 is not a valid compression type.

The tolerances are required for compression type `COMPRESSION_TYPE_BITPACKED`. They can optionally be specified per joint/user channel. However, if this level of control is not required, the associated array should be left empty and the specified default tolerance will be used.

# **libedgeanimtool Functions**

---

# ComputePeriod

---

Calculates the period for a given list of key times

## Definition

---

```
#include <libedgeanimtool_animation.h>
float ComputePeriod(
    const std::vector<float>& keyTimes,
    const double* pHints = NULL,
    unsigned int hintCount = 0
)
```

## Arguments

---

<i>keyTimes</i>	Reference to input vector of keyframe time values.
<i>pHints</i>	Pointer to input array of period hints.
<i>hintCount</i>	Number of elements in period hints array.

## Return Values

---

Returns the computed period. Errors are not flagged.

## Description

---

This function computes the period for a given array of key times. The sampling frequency is then  $1/\text{period}$ .

Due to the nature of the floating-point representation that is used, this function is not numerically stable and can produce an incorrect result if the input array contains very large time values. Period hints can be passed into this function to help avoid such incorrect results.

Specifically, the hint array can be used to pass *hintCount* number of “period hints”. A period hint is given as  $1/\text{sampling rate}$  where typical sampling rates are 25, 50, 100, 15, 30, 60, 120.

After computing the period from the *keyTimes* argument, `ComputePeriod()` will compare each hint against the computed period and select the best match. The best match is defined as the largest period that does not cause a greater absolute error than the computed value.



---

# GenerateAdditiveAnimation

---

Generates an additive (delta) animation from two input animations.

## Definition

---

```
#include <libedgeanimtool_animation.h>
void GenerateAdditiveAnimation(
    Animation& outAnimation,
    const Animation& baseAnimation,
    const Animation& animation,
    const Skeleton& bindSkeleton
)
```

## Arguments

---

<i>outAnimation</i>	The resultant delta animation gets stored here.
<i>baseAnimation</i>	Base animation.
<i>animation</i>	Target animation.
<i>bindSkeleton</i>	Bind skeleton associated with the animations.

## Return Values

---

None.  
Function modifies the parameters to contain the results of the operation.

## Description

---

This function generates a delta animation, which is computed as:

$$outAnimation = animation - baseAnimation$$

# ExportAnimation

Outputs an animation to a binary file.

## Definition

```
#include <libedgeanimtool_animation.h>
void ExportAnimation(
    const std::string outputAnimFilename,
    const Animation& animationIn,
    const Skeleton& bindSkeleton,
    bool bigEndian,
    bool stats,
    const CompressionInfo& compressionInfo,
    bool optimize,
    float optimizerTolerance,
    CustomDataCallback customDataCallback,
    void* customData
)
```

## Arguments

<i>outputAnimFilename</i>	File name to be output.
<i>animationIn</i>	The animation to export.
<i>bindSkeleton</i>	Bind skeleton associated with the animation.
<i>bigEndian</i>	If true, write data in big-endian format. Otherwise, write data in little-endian format.
<i>stats</i>	If true, output statistics to a text file.
<i>compressionInfo</i>	User-specified compression settings.
<i>optimize</i>	If true, optimize the animation (true by default).
<i>optimizerTolerance</i>	Error tolerance used by optimizer (0.001 by default).
<i>customDataCallback</i>	Callback for adding custom data (NULL by default).
<i>customData</i>	Custom data pointer (NULL by default).

## Return Values

None.

All failures are triggered through the `EDGEERROR` macro, which by default throws an exception of type `(char*)`.

## Description

This function writes out an animation to a binary file, in a format that can be used by the Edge Animation runtime.

If *stats* is true, various statistics about the animation data get written to a text file. This file has the same name as the binary file, with the extension “`.txt`” appended to it.

If *optimize* is true, the animation is first optimized to remove any keyframes that can be approximated by interpolation, within the specified tolerance.

*customDataCallback* allows users to add their own custom data to the file. If specified, the callback will get called with a pointer to the current `FileWriter` object, and the *customData* pointer. Data written by this callback will get appended to the end of the file, and its offset written to the *offsetCustomData* member of `EdgeAnimAnimation`.

---

# ExtractSkeleton

---

Creates a skeleton structure from a runtime skeleton binary structure.

## Definition

---

```
#include <libedgeanimtool_skeleton.h>
void ExtractSkeleton(
    const EdgeAnimSkeleton* pSkel,
    Skeleton& skeleton
)
```

## Arguments

---

<i>pSkel</i>	Input skeleton structure.
<i>skeleton</i>	Empty structure to be filled by this function.

## Return Values

---

None.  
Function modifies the parameters to contain the results of the operation.

## Description

---

This function takes a binary skeleton structure that was created by [ExportSkeleton\(\)](#) and converts it to a skeleton structure. The endianness of the file is determined from the skeleton structure's version tag.

---

# ExportSkeleton

---

Outputs an skeleton to a binary file.

## Definition

---

```
#include <libedgeanimtool_skeleton.h>
void ExportSkeleton(
    const std::string skelBinFilename,
    const Skeleton& skeleton,
    bool big-Endian,
    CustomDataCallback customDataCallback,
    void* customData
)
```

## Arguments

---

<i>skelBinFilename</i>	File name to be output.
<i>skeleton</i>	The skeleton to export.
<i>bigEndian</i>	If true, write data in big-endian format; otherwise, write data in little-endian format.
<i>customDataCallback</i>	Callback for adding custom data (NULL by default).
<i>customData</i>	Custom data pointer (NULL by default).

## Return Values

---

None.

All failures are triggered through `EDGEERROR` macro, which by default throws an exception of type `(char*)`.

## Description

---

This function writes out a skeleton to a binary file, in a format that can be used by the Edge Animation runtime.

*customDataCallback* allows users to add their own custom data to the file. If specified, the callback will get called with a pointer to the current `FileWriter` object and with the *customData* pointer. Data written by the callback will get appended to the end of the file, and its offset will be written to the *offsetCustomData* member of `EdgeAnimSkeleton`.

---

# ParseUserChannels

---

Parses a text file containing user-channel information.

## Definition

---

```
#include <libedgeanimtool_skeleton.h>
void ParseUserChannels (
    const std::string filename,
    std::vector<Edge::Tools::UserChannelInfo>& userChannelInfoArray
)
```

## Arguments

---

<i>filename</i>	File name to be input.
<i>userChannelInfoArray</i>	User-channel info array that gets filled by this function.

## Return Values

---

None.  
Function modifies the parameters to contain the results of the operation.

## Description

---

This function parses a text file containing user channel descriptors and creates an array of structures describing each channel.

Each line within the text file specifies a single user channel and must have the following form:

*node.attribute.component [flags]*

Where:

*node* specifies the name of the parent node of the animated attribute.

*attribute* specifies the name of the animated attribute.

*component* (optional) specifies the component of the attribute (x, y, z, or w)

*flags* (optional) is a comma separated list of flags. The following flags can be specified:

Flag	Description
clamp01	Channel is clamped to range (0,1).
minmax	Channel is of type min/max.

For example:

```
faceShapes.frown
lambert1.color.z [clamp01]
something.else.y [clamp01, minmax]
```

# Custom Data

---

# CustomDataCallback

---

Callback function for adding custom data to `EdgeAnimAnimation` and `EdgeAnimSkeleton`.

## Definition

---

```
#include <libedgeanimtool_common.h>
typedef void (*CustomDataCallback) (
    FileWriter& fileWriter,
    void* customData
)
```

## Arguments

---

<i>fileWriter</i>	The <code>FileWriter</code> object being used to write the current binary file.
<i>customData</i>	User-specified data pointer.

## Description

---

[CustomDataCallback\(\)](#) is an optional callback passed to [ExportAnimation\(\)](#) and [ExportSkeleton\(\)](#) that allows users to append their own custom data to animation and skeleton binary files.

## See Also

---

[ExportAnimation](#), [ExportSkeleton](#)

# CustomDataTable

Mechanism for adding multiple custom data entries within `EdgeAnimAnimation` and `EdgeAnimSkeleton` binary files.

## Definition

```
#include <libedgeanimtool_common.h>
class CustomDataTableEntry
{
public:
    virtual unsigned int GetHashValue() const = 0;
    virtual unsigned int GetDataAlignment() const = 0;
    virtual void ExportData( FileWriter& fileWriter ) const = 0;
}

struct CustomDataTable
{
    std::vector<CustomDataTableEntry*> m_CustomDataEntries;
};

void ExportCustomDataTable( FileWriter& fileWriter, void* customData );
```

## Class Methods

<i>GetHashValue</i>	The hash value by which this custom data entry is identified in the table.
<i>GetDataAlignment</i>	The data memory alignment before and after the custom data entry.
<i>ExportData</i>	Writes the custom data entry to file.

## Struct Members

*m\_CustomDataEntries* A vector list of `CustomDataTableEntry`

## Function Arguments

<i>fileWriter</i>	The <code>FileWriter</code> object being used to write the current binary file.
<i>customData</i>	A pointer to <code>CustomDataTable</code> .

## Description

The `CustomDataTable` is a basis to add multiple, independent custom data entries to the `EdgeAnimAnimation` and `EdgeAnimSkeleton` binary files. It is provided as an option, and any existing custom data components can still be used, either by using the [CustomDataCallback\(\)](#) function or by converting them into custom data table entries.

`ExportCustomDataTable()` conforms to the [CustomDataCallback\(\)](#) function, allowing it to be used in place of existing callbacks to [ExportAnimation\(\)](#) and [ExportSkeleton\(\)](#). The entries are exported in the order they are placed in the *m\_CustomDataEntries* vector list of `CustomDataTable`, with their hash value identifier, data size, and data all available at runtime. Each `CustomDataTableEntry` child class must implement the three abstract class methods. It is recommended that the hash value identifier be derived via the Edge Animation `edgeAnimGenerateNameHash()` method from a suitable string identifier, although this is not a requirement.



## **See Also**

---

[ExportAnimation](#), [ExportSkeleton](#), [CustomDataCallback](#)