

PlayStation®Edge
オフラインツール用アニメーション
ライブラリ リファレンス

目次

はじめに	3
このドキュメントについて	4
ツールのデータ型	5
Joint	6
AnimationKeyframe	7
JointAnimation	8
UserChannelAnimation	9
Animation	10
UserChannelInfo	11
Skeleton	12
CompressionInfo	13
libedgeanimtool の関数	15
ComputePeriod	16
GenerateAdditiveAnimation	17
ExportAnimation	18
ExtractSkeleton	20
ExportSkeleton	21
ParseUserChannels	22
カスタムデータ	23
CustomDataCallback	24
CustomDataTable	25

はじめに

このドキュメントについて

目的

このドキュメントは、Edge ライブラリのオフラインツール用アニメーションコンポーネントである libedgeanimtool の API リファレンスです。このコンポーネントをアセットパイプラインに組込むと、データを効率的に管理しつつ、高い実行時パフォーマンスを維持することができます。

edgeanimcompiler は、Edge アニメーションコンパイラのサンプルツールで、libedgeanimtool をアセットパイプラインの中でどのように利用できるかを示しています。具体的には、COLLADA™フォーマットのデータを取り込んで、Edge ライブラリに渡し、ランタイムサンプルにロードできるバイナリファイルを作成する方法を示します。edgeanimcompiler は FCollada を使用しているため、現在のところ、Windows でしかサポートされていません。

対象読者、および前提条件

このドキュメントは、PlayStation®3 用の高性能アプリケーションを書こうとしている PlayStation®3 デベロッパのため書かれたもので、デベロッパが以下の知識を持っていることを前提としています。

- C および C++
- PlayStation®3 のハードウェア
- SCE の標準ライブラリ関数

関連ドキュメント

このリファレンスと以下のドキュメントを併用することにより、Edge ライブラリの使用法やリファレンスについての完全な情報を得ることができます。

- 「PlayStation® Edge ライブラリ 概要」
- 「PlayStation® Edge ジオメトリライブラリ クイックスタートガイド」
- 「PlayStation® Edge ジオメトリライブラリ リファレンス」
- 「PlayStation® Edge オフラインツール用ジオメトリライブラリ リファレンス」
- 「PlayStation® Edge アニメーションライブラリ リファレンス」
- 「PlayStation® Edge Zlib ライブラリ リファレンス」
- 「PlayStation® Edge LZMA ライブラリ リファレンス」
- 「PlayStation® Edge LZ0 ライブラリ リファレンス」
- 「PlayStation® Edge DXT ライブラリ リファレンス」
- 「PlayStation® Edge Post ライブラリ リファレンス」

表記法

このドキュメントでは、以下のような表記法を使います。

記法	意味
等幅フォント	プログラミングコードおよびリテラル（処理命令、レジスタ名、データ型、イベント、ファイル名など）を表します。また、関数、構造体、マクロなどの名前を表すこともあります。
等幅フォント+太字	構造体や関数の定義の中でのみ、構造体や関数の名前を示します。
等幅フォント+斜体	引数、パラメータ、変数を表します。
青色+下線のテキスト	ハイパーリンクを表します（青色で表示されるのはカラープリンタやオンラインの場合だけです）。

ツールのデータ型

Joint

単一のジョイント変換を記述する

定 義

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

メ ン バ

<code>m_rotation</code>	変換のうちの回転部分（クォータニオン）
<code>m_translation</code>	変換のうちの平行移動部分
<code>m_scale</code>	変換のうちの拡大縮小部分

解 説

この構造体は、特定のジョイント変換を記述します。

AnimationKeyframe

単一のキーフレームを含む構造体

定 義

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

メ ン バ

<i>m_keyTime</i>	キーフレーム時間（秒）
<i>m_keyData</i>	キーフレームデータ

解 説

この構造体には、特定チャネルの単一のキーフレームデータが含まれています。このキーフレームデータは、回転、平行移動、スケール、ユーザチャネル値などを表すことができます。ユーザチャネルの場合、*m_keyData* の最初の要素だけが使われ、それ以外の要素はゼロに設定されます。

JointAnimation

単一ジョイントのアニメーションを含む構造体

定 義

```
#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;
};
```

メ ン バ

<i>m_jointName</i>	ジョイント名のハッシュ値
<i>m_jointWeight</i>	ジョイントの重み (0~1)
<i>m_rotationAnimation</i>	回転キーフレームの配列 (時間増加順)
<i>m_translationAnimation</i>	平行移動キーフレームの配列 (時間増加順)
<i>m_scaleAnimation</i>	スケールキーフレームの配列 (時間増加順)

解 説

単一ジョイントのアニメーションを記述する構造体です。

m_jointWeight は、実行時の重みつきブレンディングに使われる重みを表します。

関 連 項 目

[AnimationKeyframe](#)

UserChannelAnimation

単一のユーザチャネルのアニメーションを含む構造体

定 義

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

メ ン バ

<i>m_nodeName</i>	チャネルの親ノード名のハッシュ値
<i>m_channelName</i>	チャネル名のハッシュ値
<i>m_weight</i>	チャネルの重み (0~1)
<i>m_animation</i>	キーフレーム配列 (時間増加順)

解 説

この構造体は、単一ユーザチャネルのアニメーションを記述します。
m_weight は、実行時の重みつきブレンディングに使われる重みを表します。

関 連 項 目

[AnimationKeyframe](#)

Animation

一連のジョイントやユーザチャネルのアニメーションを記述する構造体

定 義

```
#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;
};
```

メ ン バ

<code>m_startTime</code>	アニメーション開始時間 (秒)
<code>m_endTime</code>	アニメーション終了時間 (秒)
<code>m_period</code>	アニメーションのサンプリング周期 (秒)
<code>m_numFrames</code>	全フレームの数
<code>m_enableLocoDelta</code>	true の場合、移動差分が出力される
<code>m_locoDeltaQuat</code>	移動回転差分 (最後-最初)
<code>m_locoDeltaTrans</code>	移動平行移動差分 (最後-最初)
<code>m_jointAnimations</code>	ジョイントアニメーションの配列
<code>m_userChannelAnimations</code>	ユーザチャネルアニメーションの配列

解 説

この構造体は、一連のジョイントやユーザチャネルのアニメーションを記述します。

`m_numFrames` は、サンプリング周波数の初期値における全フレームの数 (30Hz で 2 秒のクリップだったら 60) です。

関 連 項 目

[JointAnimation](#)、[UserChannelAnimation](#)

UserChannelInfo

ユーザチャンネルに関する情報を含んだ構造体

定 義

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

メ ン バ

<code>m_nodeNameHash</code>	チャンネルの親ノード名のハッシュ値
<code>m_channelNameHash</code>	チャンネル名のハッシュ値
<code>m_componentIndex</code>	成分のインデックス (0~3)
<code>m_flags</code>	フラグ (下記参照)

`m_flags` には、以下を組合せたものを設定することができます。

マクロ	値	解説
<code>EDGE_ANIM_USER_CHANNEL_FLAG_CLAMP01</code>	0	チャンネルは、(0,1) の範囲に固定される
<code>EDGE_ANIM_USER_CHANNEL_FLAG_MINMAX</code>	1	チャンネルは、最小/最大タイプである

解 説

この構造体には、単一のユーザチャンネルに関する情報が含まれています。

チャンネルを一意に特定するには、ハッシュ名が使われます。`m_nodeNameHash` はアニメーション属性の親ノードの名前、`m_channelNameHash` は属性自体の名前です。たとえば、チャンネルが「FaceShapes (顔の形状)」ノードの子ノードで、「Frown (眉をひそめる)」というブレンド形状重みを表しているといった具合です。

`m_componentIndex` は、このチャンネルに関連付けられた成分を特定するために使われます。チャンネルがベクトル属性の x 成分を表している場合、この値は 0 に設定されます。y 成分を表している場合は 1、z 成分を表している場合は 2、そして w 成分を表している場合は 3 になります。スカラー属性の場合には、0 に設定されます。

`m_flags` は、実行時のブレンド動作に影響します。詳しくは、ランタイムのマニュアルを参照してください。

Skeleton

ジョイントの階層や一連のユーザチャンネルを記述する構造体

定 義

```
#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;
};
```

メ ン バ

<i>m_locoJointIndex</i>	移動に使われるジョイントのインデックス
<i>m_numJoints</i>	ジョイントの数
<i>m_numUserChannels</i>	ユーザチャンネルの数
<i>m_parentIndices</i>	各ジョイントの親インデックスの配列（インデックスが-1 の場合には親がない）
<i>m_basePose</i>	スケルトンのベースポーズを定義する、各ジョイントの変換の配列
<i>m_scaleCompensateFlags</i>	各ジョイントのスケール補償フラグの配列
<i>m_jointNameHashes</i>	各ジョイントのハッシュ名の配列
<i>m_userChannelInfoArray</i>	各ユーザチャンネルを記述する構造体の配列

解 説

この構造体は、ジョイントの階層や一連のユーザチャンネルを記述します。

m_scaleCompensateFlags の中の特定のジョイントに対応する要素が真であるということは、そのジョイントが親空間に変換されるときに、親のスケールを計算に入れないということを表します。

関 連 項 目

[Joint](#)、[UserChannelInfo](#)

CompressionInfo

チャンネル圧縮モードを指定するために使われる構造体

定 義

```
#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
};
```

メ ン バ

<i>m_compressionTypeRotation</i>	回転に使われる圧縮の種類（下記参照）
<i>m_compressionTypeTranslation</i>	平行移動に使われる圧縮の種類（下記参照）
<i>m_compressionTypeScale</i>	スケールに使われる圧縮の種類（下記参照）
<i>m_compressionTypeUser</i>	ユーザチャンネルに使われる圧縮の種類（下記参照）
<i>m_defaultToleranceRotation</i>	ビットパックされたデフォルトの回転許容範囲
<i>m_defaultToleranceTranslation</i>	ビットパックされたデフォルトの平行移動許容範囲
<i>m_defaultToleranceScale</i>	ビットパックされたデフォルトのスケール許容範囲
<i>m_defaultToleranceUser</i>	ビットパックされたデフォルトのユーザチャンネル許容範囲
<i>m_jointTolerancesRotation</i>	各ジョイントの回転許容範囲
<i>m_jointTolerancesTranslation</i>	各ジョイントの平行移動許容範囲
<i>m_jointTolerancesScale</i>	各ジョイントのスケール許容範囲
<i>m_userChannelTolerances</i>	各ジョイントユーザチャンネル許容範囲

compressionTypeXXXX には、以下の値を設定することができます。

マクロ	値	解説
COMPRESSION_TYPE_NONE	0	圧縮なし
COMPRESSION_TYPE_SMALLEST_3	1	最小 3 成分圧縮
COMPRESSION_TYPE_BITPACKED	2	ビットパック圧縮

解 説

この構造体は、どのように各種類のチャンネルを圧縮すべきか、また、ビットパックの場合、どのぐらいのエラー許容範囲を使うべきかを指定するために使われます。この構造体は、ユーザーによって設定されて、[ExportAnimation\(\)](#) に引数として渡されます。

圧縮の種類の中には、一部のチャンネルではサポートされていない、もしくは無効なものがあることに注意してください。回転では、COMPRESSION_TYPE_NONE はサポートされていません。回転以外では、COMPRESSION_TYPE_SMALLEST_3 は圧縮の種類として有効ではありません。

許容範囲は、`COMPRESSION_TYPE_BITPACKED` という種類の圧縮を行うために必要です。許容範囲は、必要があれば、ジョイント/ユーザチャネルごとに指定することもできます。けれども、このレベルの制御を必要としない場合には、関連する配列は空のままにしてください。そうすれば、指定されたデフォルトの許容範囲が使われます。

libedgeanimtool の関数

ComputePeriod

指定されたキー時間リストから周期を計算する

定 義

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

引 数

<i>keyTimes</i>	キーフレーム時間値の入力ベクトルへの参照
<i>pHints</i>	周期ヒントの入力配列へのポインタ
<i>hintCount</i>	周期ヒント配列の要素数

返 り 値

計算された周期を返します。エラーのフラグは返しません。

解 説

この関数は、指定されたキー時間配列から周期を計算します。サンプリング周波数は、「1/周期」です。この関数は、入力配列に極端に大きな時間値が含まれていると、使用する浮動小数点表現の性質により、数値的に不安定になり、不正確な結果を返すことがあります。この関数に周期ヒントを渡すことは、そのような不正確な結果を回避するために役立ちます。

具体的には、ヒント配列を使うと、*hintCount* 個の「周期ヒント」を渡すことができます。周期ヒントは、「1/サンプリングレート」として指定されます。一般的なサンプリングレートは、25、50、100、15、30、60、120 です。

`ComputePeriod()` は、*keyTimes* 引数から周期を計算した後で、計算した周期と各ヒントを比較して最もよく一致する値を選択します。その最もよく一致する値とは、計算された値よりも大きな絶対誤差を出さない最大の周期と定義します。

GenerateAdditiveAnimation

2つの入力アニメーションから加算的（差分）アニメーションを生成する

定 義

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

引 数

<i>outAnimation</i>	生成された差分アニメーションが格納される
<i>baseAnimation</i>	ベースアニメーション
<i>animation</i>	ターゲットアニメーション
<i>bindSkeleton</i>	アニメーションと関連するバインドスケルトン

返 り 値

なし。

この関数は、操作の結果を含むようにパラメータを変更します。

解 説

この関数は、次のように計算された差分アニメーションを生成します。

$$outAnimation = animation - baseAnimation$$

ExportAnimation

アニメーションをバイナリファイルに出力する

定 義

```
#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
)
```

引 数

<i>outputAnimFilename</i>	出力するファイル名
<i>animationIn</i>	エクスポートするアニメーション
<i>bindSkeleton</i>	アニメーションに関連したバインドスケルトン
<i>bigEndian</i>	真の場合、ビッグエンディアン形式でデータを書き込む。 それ以外の場合には、リトルエンディアン形式でデータを書き込む
<i>stats</i>	真の場合、統計値をテキストファイルに出力する
<i>compressionInfo</i>	ユーザ指定の圧縮設定
<i>optimize</i>	真の場合、アニメーションを最適化します（デフォルトでは真）
<i>optimizerTolerance</i>	オブティマイザに使われるエラー許容範囲（デフォルトでは 0.001）
<i>customDataCallback</i>	カスタムデータを追加するためのコールバック（デフォルトでは NULL）
<i>customData</i>	カスタムデータポインタ（デフォルトでは NULL）

返 り 値

なし。

エラーはすべて `EDGEERROR` マクロによってトリガされます。このマクロは、デフォルトでは型 `(char*)` の例外をスローします。

解 説

この関数は、Edge アニメーションランタイムから利用できる形式のバイナリファイルに、アニメーションを書き込みます。

stats が真の場合、アニメーションデータについてのさまざまな統計値がテキストファイルに書き込まれます。このファイルの名前は、バイナリファイルと同じ名前に、拡張子「.txt」を付加したものになります。

optimize が true である場合、まずアニメーションを最適化し、補間によって指定された許容範囲内で近似できるキーフレームを削除します。

`customDataCallback` は、ユーザがファイルに独自のカスタムデータを追加することを可能にします。指定された場合、コールバックを呼び出す際には、現在の `FileWriter` オブジェクトへのポインタ、および `customData` ポインタが渡されます。このコールバックによって書き込まれたデータは、ファイルの最後に追加され、そのオフセットが `EdgeAnimAnimation` の `offsetCustomData` メンバに書き込まれます。

ExtractSkeleton

実行時のバイナリスケルトン構造体からスケルトン構造体を作成する

定 義

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

引 数

<i>pSkel</i>	入力スケルトン構造体
<i>skeleton</i>	この関数によって書き込まれる空の構造体

返 り 値

なし。
この関数は、操作の結果を含むように、パラメータを変更します。

解 説

この関数は、[ExportSkeleton\(\)](#)によって作成されたバイナリスケルトン構造体を受け取って、スケルトン構造体に変換します。ファイルのエンディアンネスは、スケルトン構造体のバージョンタグから判定します。

ExportSkeleton

スケルトンをバイナリファイルに出力する

定 義

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

引 数

<i>skelBinFilename</i>	出力するファイル名
スケルトン	エクスポートするスケルトン
<i>big-Endian</i>	真の場合、ビッグエンディアン形式でデータを書き込む。 それ以外の場合には、リトルエンディアン形式でデータを書き込む。
<i>customDataCallback</i>	カスタムデータを追加するためのコールバック（デフォルトでは NULL）
<i>customData</i>	カスタムデータポインタ（デフォルトでは NULL）

返 り 値

なし。

エラーはすべて EDGEERROR マクロによってトリガされます。このマクロは、デフォルトでは型（char*）の例外をスローします。

解 説

この関数は、Edge アニメーションランタイムから利用できる形式のバイナリファイルに、スケルトンを書き込みます。

customDataCallback は、ユーザがファイルに独自のカスタムデータを追加することを可能にします。指定された場合、コールバックを呼び出す際には、現在の FileWriter オブジェクトへのポインタ、および *customData* ポインタが渡されます。このコールバックによって書き込まれたデータは、ファイルの最後に追加され、そのオフセットが EdgeAnimSkeleton の *offsetCustomData* メンバに書き込まれます。

ParseUserChannels

ユーザチャンネル情報を含むテキストファイルを解析する

定 義

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

引 数

<i>filename</i>	入力するファイル名
<i>userChannelInfoArray</i>	この関数によって書き込まれるユーザチャンネル情報配列

返 り 値

なし。

この関数は、操作の結果を含むように、パラメータを変更します。

解 説

この関数は、ユーザチャンネル記述子を含むテキストファイルを解析して、各チャンネルを記述する構造体の配列を作成します。

テキストファイル中の各行は、特定のユーザチャンネルを表しており、以下の形式をとる必要があります。

```
node.attribute.component [flags]
```

node は、アニメーション属性の親ノードの名前を指定します。

attribute は、アニメーション属性の名前を指定します。

component (オプション) は、属性の成分 (x、y、z、w) を指定します

flags (オプション) は、コンマで区切られたフラグのリストです。フラグとして指定できるのは、以下の通りです。

フラグ	解説
clamp01	チャンネルは、(0,1) の範囲に固定される
minmax	チャンネルは、最小/最大タイプである

例：

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

カスタムデータ

CustomDataCallback

EdgeAnimAnimation、EdgeAnimSkeleton にカスタムデータを追加するためのコールバック関数

定 義

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

引 数

<i>fileWriter</i>	現在のバイナリファイルを書き込むために使われている FileWriter オブジェクト
<i>customData</i>	ユーザ指定のデータポインタ

解 説

[CustomDataCallback\(\)](#) は、オプションで [ExportAnimation\(\)](#) や [ExportSkeleton\(\)](#) に渡されるコールバックで、ユーザがアニメーションやスケルトンバイナリファイルに独自のカスタムデータを追加することを可能にします。

関 連 項 目

[ExportAnimation](#)、[ExportSkeleton](#)

CustomDataTable

EdgeAnimAnimation および EdgeAnimSkeleton バイナリファイルの中に、複数のカスタムデータ入力を追加するためのメカニズム

定 義

```
#include <libedgeanimtool_common.h>
class CustomDataTableEntry
{
public :
    virtual unsigned int GetHashCode() 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 );
```

ク ラ ス メ ソ ッ ド

<i>GetHashCode</i>	このカスタムデータ入力をテーブルの中から特定するためのハッシュ値
<i>GetDataAlignment</i>	カスタムデータ入力の前後のデータメモリアラインメント
<i>ExportData</i>	カスタムデータ入力をファイルに書き込む

構 造 体 の メ ン バ

m_CustomDataEntries CustomDataTableEntry のベクトルリスト

関 数 の 引 数

<i>fileWriter</i>	現在のバイナリファイルを書き込むために使われている FileWriter オブジェクト
<i>customData</i>	CustomDataTable へのポインタ

説 明

CustomDataTableは、EdgeAnimAnimationおよびEdgeAnimSkeletonバイナリファイルに、独立した複数のカスタムデータ入力を追加するための基盤になります。この構造体は、オプションとして提供されており、既存のカスタムデータ構成要素も、[CustomDataCallback\(\)](#)関数を利用したり、カスタムデータテーブルエントリに変換したりすることにより、そのまま利用できます。

ExportCustomDataTable() は [CustomDataCallback\(\)](#) 関数に準拠しているので、[ExportAnimation\(\)](#) や [ExportSkeleton\(\)](#) の既存のコールバックの代わりに利用できます。エントリは、ハッシュ値識別子、データサイズ、実行時に利用できる全データといっしょに、CustomDataTableの m_CustomDataEntriesベクトルリストに配置された順序でエクスポートされます。

CustomDataTableEntryの子クラスは、すべて、3つの抽象クラスメソッドを実装する必要があります。これは必須条件ではありませんが、ハッシュ値識別子は、適切な文字列識別子からEdgeアニメーションのedgeAnimGenerateNameHash()メソッドによって導出することをお勧めします。

関 連 項 目

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