Tween Media Handler

Created: September 11, 1995
Last Update: March 31, 1996

Introduction

The tween media handler acts as a modifier track, sending algorithmically generated values to other tracks. The tween media handler compliments the capabilities of the base media handler, which is also used as a modifier track, but only for sending a discrete set of values. The tween media handler never displays its media data.

Data Types Overview

The tween media handler generates a value to send to another track from a pair of values and a percentage. In the most simple example, the pair of values are integers and the percentage is used to linearly interpolate between the two integers. The tween media handler currently only supports linear interpolation. The following data types are currently supported by the tween media handler: short integer, long integer, Fixed, QuickDraw point, QuickDraw rectangle, QuickTime 3x3 Matrix, and QuickDraw RGB Color. Rather than supporting all possible data types that might be tweened, the tween media handler provides a mechanism for tween components to work with the tween media handler to generate other types of values as well as non-linear interpolations.

Creating a Tween Media Track

The tween media handler is based on the modifier track support introduced with QuickTime 2.1. Refer to the 2.1 documentation for basic information.

To add the tween media to an existing movie, use the follow code:

```
      Track t;
      Media md;
      SampleDescriptionHandle desc
            =(SampleDescriptionHandle)NewHandleClear(sizeof(SampleDescri
            ption));

      t = NewMovieTrack(m, 0, 0, kNoVolume);
      md = NewTrackMedia(t, TweenMediaType, 600, nil, 0);

      (**desc).descSize = sizeof(SampleDescription);
```

The sample data for the tween media handler is based on the QuickTime AtomData structure introduced with QuickTime 2.1. By using this structure, a single tween media can be generating several values in parallel. The following example creates a tween sample which interpolates a short integer from 512 to 0.

```
      QTAtomContainer container = nil;
      short tweenDataShort[2];
      QTAtomType tweenType;

      tweenDataShort[0] = 512;
      tweenDataShort[1] = 0;
```

```
QTNewAtomContainer(&container);

tweenType = tweenTypeShort;
QTInsertChild(container, kParentAtomIsContainer, kTweenEntry, 1,
      0, 0, nil, &tweenAtom);
QTInsertChild(container, tweenAtom, kTweenType, 1, 0,
      sizeof(tweenType), &tweenType, nil);
QTInsertChild(container, tweenAtom, kTweenData, 1, 0,
      sizeof(short) * 2, tweenDataShort, nil);

BeginMediaEdits(md);
AddMediaSample(md, container, 0,
      GetHandleSize(container),kSampleDuration,
      sampleDescriptionH, 1, 0, nil);
EndMediaEdits(md);

DisposeHandle((Handle)sampleDescriptionH);
QTDisposeAtomContainer(container);

InsertMediaIntoTrack(t, 0, 0, kSampleDuration, kFix1);

SetTrackEnabled(t, true);
```

A tween doesn’t have to start at the beginning of a tween sample, not does it have to end at the end of the tween sample. You can specify the starting offset into the sample for the tween to begin by adding kTweenStartOffset atom to the tween entry. The start offset atom contains a time value indicating the number of units into the tween sample that the tween should be delayed. The time value is expressed in the media’s time coordinate system. Similarly, you can specify the duration of the tween by adding a kTweenDuration atom to the tween entry. The duration atom contains a time value indiciating the number of units of the time the tween should last. The time value is also expressed in the media’s time coordinate system. If the start offset atom is not specified, the tween is assumed to start at the beginning of the sample. If the duration atom is not specified, the tween is assumed to have a duration of (sample duration - start offset). The start offset can be negative number, just as the duration can end after the end of the sample.

In the following example, a start off and duration are specified so that the tween will start 1/2 second into the sample, and last for one second. This example assumes the media’s time scale is 1000.

```
TimeValue time;

time = 500;
QTInsertChild(container, tweenAtom, kTweenStartOffset, 1, 0,
      sizeof(time), &time, nil);
time = 1000;
QTInsertChild(container, tweenAtom, kTweenDuration, 1, 0,
      sizeof(time), &time, nil);
```

The root level of the AtomData structure contains atoms of type kTweenEntry. The id’s of these atoms are used to bind them to their target tracks, as is shown in the following example. In this code, the tween media is attached to a sound track, to modify the sound track's volume.

```
	Track soundTrack;

	soundTrack = GetMovieIndTrackType(theMovie , 1,
		AudioMediaCharacteristic, movieTrackCharacteristic |
		movieTrackEnabledOnly);
	if (soundTrack) {
		long referenceIndex;
		if (AddTrackReference(soundTrack, t,
			kTrackModifierReference, &referenceIndex) == noErr) {
			QTAtomContainer inputMap = nil;

			if (QTNewAtomContainer(&inputMap) == noErr) {
				QTAtom inputAtom;
				OSType inputType;
				long tweenID = 1;

				QTInsertChild(inputMap, kParentAtomIsContainer,
					kTrackModifierInput, referenceIndex,
					        0, 0, nil, &inputAtom);

				inputType = kTrackModifierTypeVolume;
				QTInsertChild(inputMap, inputAtom,
					kTrackModifierType, 1, 0,
				   sizeof(inputType), &inputType, nil);
				QTInsertChild(inputMap, inputAtom,
					kInputMapSubInputID, 1, 0,
				   sizeof(tweenID), &tweenID, nil);

				SetMediaInputMap(GetTrackMedia(soundTrack),
					inputMap);

				QTDisposeAtomContainer(inputMap);
			}
		}
	}
```

The only addition to the above code since QuickTime 2.1 is the addition of the "sub input id" which contains a long word indicating the id of the tween entry to use on the specified input.

Data Types Details

Within each tween entry atom (type kTweenEntry) in the tween media samples, there is information that is used by the specific tweener type to generate values. The type of data varies depending on the particular type of tweening that is to take place. All currently defined tween types put their basic (and often only) data into an atom of type "kTweenData". Other data atoms may be present as necessary (such as for the region tweener), depending on the type of tweening.

kTweenTypeShort
The data atom contains two short integers (16 bit).

kTweenTypeLong
The data atom contains two long integers (32 bit).

kTweenTypeFixed
The data atom contains two Fixed point values (32 bit).

kTweenTypePoint
The data atom contains two QuickDraw points. Each value in the point is interpolated separately.

kTweenTypeQDRect
The data atom contains two QuickDraw rectangles. Each value in the rectangle is interpolated separately.

kTweenTypeQDRegion
The data atom contains two QuickDraw rectangles. A separate region atom (kTweenRegionData) contains a single QuickDraw region. The rectangles are interpolated as in tweenTypeQDRect and then the region is mapped to the resulting rectangle.

kTweenTypeMatrix
The data atom contains two QuickTime 3x3 matrices. Each value in the matrix is interpolated separately. Note: this works well for translation and scaling. Varying other parameters will produce unexpected results.

kTweenTypeRGBColor
The data atom contains two QuickDraw RGB Colors. Each value in the RGB Color is interpolated separately.

kTweenType3dScale
This tween type is based on QuickDraw 3D's scale transform. The data atom contains a QuickTime 3D TQ3Vector3D record.

kTweenType3dTranslate
This tween type is based on QuickDraw 3D's translation transform. The data atom contains a QuickTime 3D TQ3Vector3D record.

kTweenType3dRotate
This tween type is based on QuickDraw 3D's rotate transform. The data atom contains a QuickTime 3D TQ3RotateTransformData record. Only the angle field is interpolated.

kTweenType3dRotateAboutPoint
This tween type is based on QuickDraw 3D's rotate about point transform. The data atom contains a QuickTime 3D TQ3RotateAboutPointTransformData record. Only the angle field is interpolated.

kTweenType3dRotateAboutAxis
This tween type is based on QuickDraw 3D's rotate about axis transform. The data atom contains a QuickTime 3D TQ3RotateAboutAxisTransformData record. Only the angle field is interpolated.

kTweenType3dQuaternion

This tween type is based on QuickDraw 3D’s quaternion transform. The data atom contains a QuickTime 3D TQ3Quaternion record.

kTweenType3dMatrix
This tween type is based on QuickDraw 3D’s 4x4 matrix transform. The data atom contains a QuickTime 3D TQ3Matrix4x4 record.

Tweener Components

Tweener components provide a mechanism for extending the kinds of values that can be interpolated. While tweener components have been created for use by the tween media handler, the component API itself is designed to allow tweener components to be used independently of the tween media handler. The subtype of the tweener component is used to match tween components to the “kTweenType” provided in the tween entry field in the media sample.

The tween component interface consists of three routines: Initialize, DoTween, and Reset.

```
pascal ComponentResult TweenerInitialize(TweenerComponent tc,
QTAtomContainer container, QTAtom tweenAtom, QTAtom dataAtom)
```

On initialize, the tweener component is given the QTAtomContainer that the tween data is stored in. The tweenAtom is the container atom for the particular tween entry to use. The dataAtom is provided as a convenience, since most tweeners will need this value. The tweener component should use this call to extract any data from the QTAtomContainer it will need during tweening. The QTAtomContainer is provided during the DoTween call, but for performance reasons it is better to extract the data once during initialize.

```
pascal ComponentResult TweenerDoTween(TweenerComponent tc,
TweenRecord *tr)
```

The DoTween routine performs the actual tween operation. A TweenRecord is supplied which contains all the information required for tweening.

```
typedef struct {
      long                    version;

      QTAtomContainer         container;
      QTAtom                  tweenAtom;
      QTAtom                  dataAtom;
      Fixed                   percent;

      TweenerDataProc         dataProc;

      void                    *private1;
      void                    *private2;
} TweenRecord;
```

The container, tweenAtom, and dataAtom fields have the same meaning as the parameters to Initialize. The percent field contains a fixed point number from 0.0 to 1.0 indicating how far into the tweening interpolation the generated value should be. The dataProc field contains a routine pointer to call to send interpolated value to.

```
typedef pascal ComponentResult (*TweenerDataProc)(struct
      TweenRecord *tr, void *tweenData, long tweenDataSize, long
      dataDescriptionSeed, Handle dataDescription,
      ICMCompletionProcRecordPtr asyncCompletionProc, ProcPtr
      transferProc, void *refCon);
```

The TweenerDataProc is nearly identical to the MediaSetNonPrimarySourceData routine. The only difference is the first parameter, which should simply be the “dataProc” passed in to TweenerDoTween.

```
pascal ComponentResult TweenerReset(TweenerComponent tc)
```

On Reset, the tweener should dispose of any data it allocated during Initialize.

An simple example tweener component for tweening a pair of short integers is shown below.

```
pascal ComponentResult TweenShortInitialize(Handle storage,
      QTAtomContainer container, QTAtom tweenAtom, QTAtom
      dataAtom)
{
      return noErr;
}

pascal ComponentResult TweenShortDoTween(Handle storage,
      TweenRecord *tr)
{
#pragma unused(storage)
      short *data;
      short tFrom, tTo, tValue;

      QTGetAtomDataPtr(tr->container, tr->dataAtom, nil,
            (Ptr *)&data);

      tFrom = data[0];
      tTo = data[1];
      tValue = tFrom + FixMul(tTo - tFrom, tr->percent);

      (tr->dataProc)((struct TweenRecord *)tr, &tValue,
            sizeof(tValue), 1, nil, nil, nil, nil);

      return noErr;
}

pascal ComponentResult TweenShortReset(Handle storage)
{
      return noErr;
}
```

Reference

Stuff that is in “Movies.h” somewhere...

```
enum {
      tweenTypeShort = 1,
      tweenTypeLong,
      tweenTypeFixed,
      tweenTypePoint,
      tweenTypeQDRect,
      tweenTypeQDRegion,
      tweenTypeMatrix,
      tweenTypeRGBColor,
      kTweenTypeGraphicsModeWithRGBColor,

      kTweenType3dScale = '3sca',
      kTweenType3dTranslate = '3tra',
      kTweenType3dRotate = '3rot',
      kTweenType3dRotateAboutPoint = '3rap',
      kTweenType3dRotateAboutAxis = '3rax',
      kTweenType3dQuaternion = '3qua',
      kTweenType3dMatrix = '3mat'
};

#define kTweenEntry 'twen'
#define kTweenData 'data'
#define kTweenType 'twnt'
#define kTweenStartOffset 'twst'
#define kTweenDuration 'twdu'

#define kInputMapSubInputID 'subi'

#define kTweenRegionData 'qdrg'

#define TweenMediaType 'twen'

#define TweenComponentType 'twen'

typedef ComponentInstance TweenerComponent;
```