- Basic Concepts
- Segmentation
- Playback Speed
- Segmented Parametric Animation
    - Ambiguity
    - Possible Solution
- Animation with Only One Key
- Direction of Time
- Time within Controllers

Basic Concepts

Depending upon the situation CryAnimation uses different units of 'time'. How those units of time compare is best explained using an example.

First off a definition of 'frames' - CryAnimation uses a fixed rate of 30 Frames per Second (fps). Of course games can run at higher frame rates, but some operations in the Editor that use the concept of 'frames' or operations that clamp the animation duration to 'one frame' assume a frame rate of 30 fps.

Let's say then, that we have an animation with a duration of 1.5 seconds. This means the animation has 46 frames (note this includes the final frame). so, in the case of Real Time we can assume an animation starts at time 0, has no segmentation and is played back at normal speed. However, rather than using Real Time, CryAnimation typically uses Animation Normalized Time - this is compared with Real Time in the table below:

Frame Index	Real Time	Animation Normalized Time
0	0.0 s	0.0
1	0.033.. s = 1/30 s	0.022.. = 1/45
..	..	..
30	1.0 s	0.666.. = 30/45
..	..	..
44	1.466.. s = 44/30 s	0.977.. = 44/45
45	1.5 s = 45/30 s	1.0

- **Real Time**
    - Time in seconds

- **Animation Normalized Time**
    - Time relative to the total length of the animation
    - Starts at 0 at the beginning of the animation and ends at 1 (= RealTime/Duration = Keytime/LastKeyTime)
    - Used by functions such as `ISkeletonAnim::GetAnimationNormalizedTime()` and `ISkeletonAnim::SetAnimationNormalizedTime()`
    - Is not well defined for parametric animations with examples that have differing numbers of segments. See Segmentation below

Real Time is used to define Duration

- **Duration**
    - Duration = lastFrame.realTime - firstFrame.realTime. That's 1.5s in our example
    - `IAnimationSet::GetDuration_sec()` returns the duration of an animation. **Note:** for a parametric animation this returns only a crude approximation: the average duration of all its examples, ignoring parameters or speed scaling
    - `CAnimation::GetExpectedTotalDurationSeconds()` returns the duration of an animation that is currently playing back. **Note:** for a parametric animation this returns only a crude approximation, assuming the parameters are the ones that are currently set and never change throughout the animation
    - There is no function that returns the real time of an animation, for that you need to manually multiply Animation Normalized Time with the duration

Segmentation

In practice CryAnimation doesn't really use Animation Normalized Time (this terminology has been used to make the introduction easier to understand), typically it uses Segment Normalized Time and for this you need to understand Segmentation.

For time warping (phase matching) purposes animations can be split into multiple segments, for example you want to time warp from a walk animation with 2 cycles to a walk animation with 1 cycle. In this case you have to annotate the first animation and split it into two - this is what we mean by Segments. This segmentation is achieved by adding a segment1 animevent at the border between the cycles.

Note: an animation without segmentation has exactly 1 segment which runs from beginning to end.

Segmentation introduces a new unit for time, Segment Normalized Time which is time relative to the current segment duration.

So, extending our example, see how the table now looks when a segment1 animevent at 1.0s has been added to split the animation into two segments.

Frame Index	Real Time	AnimEvents	(Animation) Normalized Time	Segment Index	Segment Normalized Time
0	0.0 s		0.0	0	0.0
1	0.033.. s		0.022..	0	0.033.. = 1/30
..	..		..	..	..
30	1.0 s	segment1	0.666..	1	0.0
..	..		..	..	..
44	1.466.. s		0.977..	1	0.933.. = 14/15
45	1.5 s		1.0	1	1.0

- **Segment Index**
  - Identifies which segment you are currently in - runs from 0 to the total number of segments minus 1
  - While an animation is playing you can use `CAnimation::GetCurrentSegmentIndex()` to retrieve it
  - When using ca_debugtext or es_debuganim, then this index is displayed after "seg:"
- **Segment Normalized Time**
  - Time relative to the current segment's duration
  - 0 at the beginning of the segment, 1 at the end (only 1 for the last segment as you can see in the table above)
  - While an animation is playing you can use `CAnimation::Get/SetCurrentSegmentNormalizedTime()` to get/set the Segment Normalized Time
  - As the names suggest, `CAnimation::GetCurrentSegmentIndex()` retrieves the current segment index and `CAnimation::GetCurrentSegmentExpectedDurationSeconds()` retrieves the duration of the current segment
  - Segment Normalized Time is more convenient to use when representing time within parametric animations and is used at runtime rather than Animation Normalized Time.
  - AnimEvent time is specified using Animation Normalized Time (except for the special case of parametric animation, see below)
  - When using ca_debugtext or es_debuganim, Segment Normalized Time is displayed after "ATime:". Real time within the segment and the segment duration is displayed after that within parentheses

Playback Speed

Playback speed does not impact the functions that compute duration of playing animations such as `CAnimation::GetExpectedTotalDurationSeconds()` or `ISkeletonAnim::CalculateCompleteBlendSpaceDuration()`.

Segmented Parametric Animation

Ambiguity

Animation Normalized Time, Segment Index and Duration all create ambiguity for segmented parametric animations. This is because each example animation within the parametric animation can have its own number of segments, this complicates matters.

So to avoid ambiguity, Animevents in/on segmented parametric animations use Segment Normalized Time. **Note:** because of this an Animevent will be fired multiple times (once per segment) during the animation

`ISkeletonAnim::GetAnimationNormalizedTime()` uses a heuristic: it currently looks for the example animation with the largest number of segments and returns the animation normalized time within that example.

`ISkeletonAnim::GetCurrentSegmentIndex()` uses a different heuristic: it currently returns the segment index in the example animation which happens to be the first in the list.

Possible Solution

Given this we are considering redefining the above based on the following observation;

You can define the total number of segments in a parametric animation as the number of segments until repetition starts.

So, let's say you have a parametric animation consisting of 2 examples, one with 2 segments and the other with 3 segments. This will start to repeat after 6 segments (the lowest common multiple of 2 and 3). However, you can uniquely identify each possible combination of segments using any number from 0 to 5.

This method is used in the Character Tool to achieve a well defined duration. For this we implemented; `ISkeletonAnim::CalculateCompleteBlendSpaceDuration()`. This function calculates the duration until the parametric animation starts to repeat (assuming the parameters remain fixed). It reverts to the regular `GetExpectedTotalDurationSeconds()` implementation for non-parametric animations so the function can be used in more general situations.

Animation with Only One Key

Normally your animations have at least two keys. However, when you convert these into additive animations the first frame is interpreted as the base from which to calculate the additive, hence this leaves only 1 frame in the additive animation (currently this means that in respect to the asset both the start and end time of the asset are set to 1/30 s)

Functions retrieving the total duration of this animation will return 0.0. (e.g. `IAnimationSet::GetDuration_sec()`, `ISkeletonAnim::CalculateCompleteBlendSpaceDuration()` and `CAnimation::GetExpectedTotalDurationSeconds()`)

However, for playback purposes CryAnimation will handle these animations as if they have a duration of 1/30th of a second. For e.g. Animation Normalized Time will still progress from 0 to 1 while the time goes from 0 to 1/30th of a second. `CAnimation::GetCurrentSegmentExpectedDurationSecond()` will also return 1/30th of a second in this case.

Direction of Time

Time typically cannot run backwards when playing an animation.

You can only move time backwards if you do it manually. This can be achieved by setting the flag CA_MANUAL_UPDATE on the animation and using `CAnimation::SetCurrentSegmentNormalizedTime` - see the example `CProceduralClipManualUpdateList::UpdateLayerTimes()`.

Time within Controllers

This section is only relevant to full source code Licensees.

For the controllers that contain the actual key data and that are used for animation sampling then different units are used.

Frame Index	Real Time	I_CAF Ticks	Keytime
0	0.0 s	0	0.0
1	0.033.. s	160	1.0
..	..	..	..
30	1.0 s	4800	30.0
..	..	..	..
44	1.466.. s	7040	44.0
45	1.5 s	7200	45.0

- **I_CAF Ticks**
  - Used within I_CAF files to represent time
  - There are 4800 I_CAF ticks per second. (this is currently expressed by the fact that TICKS_CONVERT = 160 in Controller.h, which assumes 30 keys/second
- **Keytime**
  - Used at runtime to pass time to the controllers for sampling animation
  - Used within CAF files to represent time
  - A floating point version of 'frame index'
  - Can represent time in between frames
  - Use GlobalAnimationHeaderCAF::NTime2KTime( ) to convert from animation normalized time to keytime
  - All animation controllers in the runtime use Keytime

Animation assets can also have a StartTime other than 0.0s - this complicates matters a little, but only for the controllers. Typically, everywhere but the controllers, time is taken relative to this StartTime.