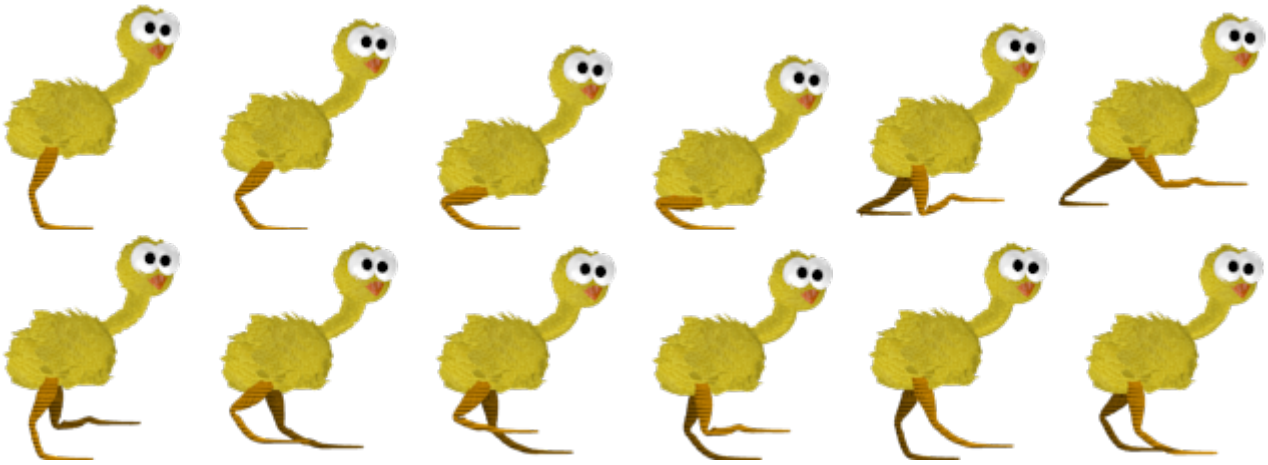
Zero Length Frames

You can use a zero length frame in order to send a key frame event without any visual indication in the animation itself. Essentially a sequence trigger.

As in the Animation Walkthrough, we'll use the chicken sprite sheet again:



Basic Config Setup

Create a standard orx project.

Start with some basic config to set up the chicken, the animation and the frames:

```
[Scene]
ChildList       = Chicken

[Chicken]
Graphic       = @
Texture       = chicken-animation-sheet.png
TextureOrigin = (0, 0, 0)
TextureSize   = (108, 115, 0)
Pivot         = top left
AnimationSet  = ChickenAnimationSet

[ChickenAnimationSet]
Texture     = chicken-animation-sheet.png
FrameSize   = (108, 115, 0)
StartAnim   = SitDownAnim
KeyDuration = 2 ; frame every two seconds
SitDownAnim = 0 ; five frames are specified, so need to keep this total up
to date.

[SitDownAnim1]
TextureOrigin = (0, 0, 0)
```

```
[SitDownAnim2]
TextureOrigin = (108, 0, 0)
KeyEvent      = SITTING_FRAME_2 # 100

[SitDownAnim3]
KeyDuration = 0
KeyEvent    = SITTING_FRAME_3 # 900

[SitDownAnim4]
TextureOrigin = (216, 0, 0)
KeyEvent      = SITTING_FRAME_4 # 230

[SitDownAnim5]
TextureOrigin = (324, 0, 0)
```

Animation Event Handler

We'll need an animation event handler so that we can print out the Key Event names and values.

At the bottom to the `init()`:

```
orxEvent_AddHandler(orxEVENT_TYPE_ANIM, AnimationEventHandler);
```

And the `AnimationEventHandler` function itself:

```
orxSTATUS orxFASTCALL AnimationEventHandler(const orxEVENT *_pstEvent){
    orxANIM_EVENT_PAYLOAD *pstPayload;
    pstPayload = (orxANIM_EVENT_PAYLOAD *)_pstEvent->pstPayload;

    switch(_pstEvent->eID){
        case orxANIM_EVENT_CUSTOM_EVENT: {
            orxLOG("<%s> / <%s> event was fired. Value: %f ",
pstPayload->zAnimName, pstPayload->stCustom.zName,
pstPayload->stCustom.fValue );
            break;
        }
    }

    return orxSTATUS_SUCCESS;
}
```

Compile and run, and we should have a basic chicken on the screen.

Analysis

The chicken will start to slowly sit down. Frames change every two seconds due to the default `KeyDuration`, which helps you see what's going on. Each frame will be 2 seconds unless we say otherwise in a specific frame.

The log should be something like:

```
[22:36:41] [LOG] <SitDownAnim> / <SITTING_FRAME_2> event was fired. Value:
100.000000
[22:36:45] [LOG] <SitDownAnim> / <SITTING_FRAME_3> event was fired. Value:
900.000000
[22:36:45] [LOG] <SitDownAnim> / <SITTING_FRAME_4> event was fired. Value:
230.000000
```

And the above repeats forever. You'll note the following interesting things:

There is no SITTING_FRAME_1. This is because frame 1 is defined as just a graphic change, there is no KeyEvent and KeyValue:

```
[SitDownAnim1]
TextureOrigin = (0, 0, 0)
```

However we go to next frame straight away, within the same 'tick', and, as such, this 'ghost' frame doesn't require a proper graphic to exist.

Frames 2 fires and then 3, being the next frame, fires within the same 'tick' (due to the KeyDuration of 0). And this is the real point of this tutorial. Frame 2 changes the graphic texture and fires the SITTING_FRAME_2 KeyEvent name and value of 100.

```
[SitDownAnim2]
TextureOrigin = (108, 0, 0)
KeyEvent      = SITTING_FRAME_2 # 100

[SitDownAnim3]
KeyDuration = 0
KeyEvent    = SITTING_FRAME_3 # 900
```

Frame 4 then shows for 2 seconds, SITTING_FRAME_4 fires with a value of 230.

```
[SitDownAnim4]
TextureOrigin = (216, 0, 0)
KeyEvent    = SITTING_FRAME_4 # 230
```

Frame 5 will change graphic texture, but there is no event fired:

```
[SitDownAnim5]
TextureOrigin = (324, 0, 0)
```

That's pretty much it. This is handy for firing an event on a certain frame and being able to trigger some game activity. Examples could be:

1. Footstep sounds being played when a foot contacts the ground.
2. A sword sound when the sword swings past.
3. Bullets fire when a frame is shown.