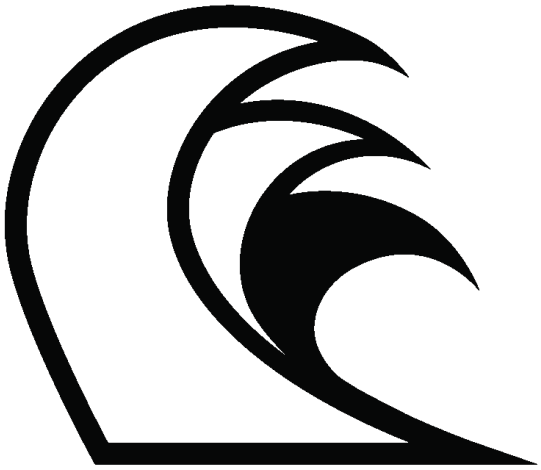
Programming in MotionBuilder || Focusing on Python
Autodesk MotionBuilder 2013

Autodesk Developer Network
May 2012
Module 7: Animation



AUTODESK®
MOTIONBUILDER

Contents

Programming in MotionBuilder || Focusing on Python

Autodesk Developer Network

Module 7: Animation

Agenda

- Animation Nodes
- Keying Animation
- FCurve
- Time
- Player Control (aka Transport Control)
- Filters

7.0 Animation Nodes

An animation node is the node of a Box in MotionBuilder. It represents a channel through which data flows (in vector or number format) in either direction. An animation node is a connector that represents a flow of data between two elements. Boxes, models, and devices are all Boxes. This functionality is also in constraints that deal with connecting boxes together.

They are data ports or plugs from which devices, models, constraints, and operators read and write to and from the data. Use animation nodes both in the real-time computation of data and for recording and filtering.

In particular, they are used in the following contexts:

- Animation nodes access a model's position, orientation, and scaling data.
- Input devices stream data into output animation nodes, and output devices read input animation nodes.
- Model constraints modify a model's position, rotation, and scaling data though a model's animation node (FBModel::AnimationNode).
- Filters modify data by accessing recorded or keyframed animation stored in FCurves found in animation nodes.

Animation nodes contain animation/keyframe data. Generally, they are structures which can serve as senders or receivers data flowing within entities like models, relations constraint boxes, devices, constraints, or filters.

Animation nodes are organized in a hierarchy. Certain nodes won't directly contain animation data. Use the Nodes list to access sub-nodes.

For example, an FBModel will have a root animation node. Among the elements in the root node's Nodes list is Lcl Translation. Lcl Translation has 3 subnodes (in its Nodes list): X, Y, and Z, which will have the animation data.

```
from pyfbsdk import *

lCube = FBFindModelByLabelName("Cube")
lFBPropertyAnimatable = lCube.Translation
print "Property Name: ", lFBPropertyAnimatable.Name

lAnimationNode = lFBPropertyAnimatable.GetAnimationNode()
print "Animation Node Name: ", lAnimationNode.Name
```

Note: Starting MB2012, the translation, rotation and scale of a model object is not animated by default, so if you didn't manually set your translation to be animated, "lFBPropertyAnimatable.GetAnimationNode()" will return "None", the above code will not work correctly.

```
lListAnimationNodes = lAnimationNode.Nodes
for lListAnimationNode in lListAnimationNodes:
    print lListAnimationNode.Name,
```

The above approach is a long way to access animation node, here is the short version using a helper function 'FindAnimationNode"

```
from pyfbsdk import *

def FindAnimationNode( pParent, pName ):
    lResult = None
    for lNode in pParent.Nodes:
        if lNode.Name == pName:
            lResult = lNode
            break
    return lResult

lCube = FBFindModelByLabelName("Cube")

lCubeLocalRot = FindAnimationNode(lCube.AnimationNode, 'Lcl Translation' )
lCubeLocalRotX = FindAnimationNode( lCubeLocalRot, 'X' )
lCubeLocalRotY = FindAnimationNode( lCubeLocalRot, 'Y' )
lCubeLocalRotZ = FindAnimationNode( lCubeLocalRot, 'Z' )

print "SubValues: ", lCubeLocalRotX.Name
```

7.1 Keying Animation

Keying becomes to be a bit of a complex thing to discuss in MotionBuilder because there are numerous different object/properties you want to key and there are also several ways to do it.

You can add keys directly to animation node (KeyAdd() or KeyCandidate() of FBAnimationNode) or through the FCurve (KeyAdd() or KeyInsert() of FBFCurve). The respective classes have a KeyRemove() function for removing keys.

- Key () : FBPropertyAnimatable , FBPlayerControl , FBEventInput
  - FBPropertyAnimatable.Key()takes the candidate and transforms it into a key.
  - From the FBPlayerControl everything is keys, not control of what properties are keyed
- KeyAdd () : FBAnimationNode , FBFCurve
  - Keying on a specific property or the FCurve of that property, more precise
- KeyCandidate () : FBAnimationNode
- KeyCount: FBAnimationNode
- KeyInsert () : FBFCurve
  - One use is for gap keys
- KeyRemove () : FBFCurve , FBAnimationNode
- Keys : FBFCurve

What is the difference between SetCandidate/KeyCandidate vs. KeyAdd?

SetCandidate: Just sets up keyable positions of the object, so that when you key it, it's ready if you want to key them, for example when you move a sphere in the UI it creates all these candidate keys behind the scenes so then when you key it will have done all the computation behind the scene, adds this at the current time.

```
from pyfbsdk import *

lCube = FBFindModelByLabelName("Cube")
lAnimationNode = lCube.Translation.GetAnimationNode()
if lAnimationNode:
        lListAnimationNodes = lAnimationNode.Nodes
        if lListAnimationNodes:
                lListAnimationNodes[0].SetCandidate(10.0)
```

Currently there is a bug with MotionBuilder 2013 python interface that SetCandidate doesn't work correctly.

KeyCandidate: Key's the candidate keys you set up with SetCandidate at the current time.

```
from pyfbsdk import *

lCube = FBFindModelByName("Cube")
lAnimationNode = lCube.Translation.GetAnimationNode()
if lAnimationNode:
        lListAnimationNodes = lAnimationNode.Nodes
        if lListAnimationNodes:
                lListAnimationNodes[0].SetCandidate(10.0)

                lListAnimationNodes[0].KeyCandidate()
```

KeyAdd: Adds a key to the animation node at current time

Keying Visibility

This is so show you an example that there are lots of ways to do things, some are better than others, for example this first example, is not a recommended solution as it

```
from pyfbsdk import *

#Find Model
lCube=FBFindModelByLabelName("Cube")
if not lCube.Visibility.IsAnimated():
        lCube.Visibility.SetAnimated(True)

if not lCube.Visibility.IsFocused():
        lCube.Visibility.SetFocus(True)

#Key 1
FBPlayerControl().Goto(FBTime(0,0,0,0))
lanimnode = lCube.Visibility.GetAnimationNode ()
lanimnode.KeyAdd(True)
FBSystem().Scene.Evaluate()

#Key 2
FBPlayerControl().Goto(FBTime(0,0,0,10))
lanimnode = lCube.Visibility.GetAnimationNode ()
lanimnode.KeyAdd(True)
FBSystem().Scene.Evaluate()
```

```
#Key 3
FBPlayerControl().Goto(FBTime(0,0,0,11))
lanimnode = lCube.Visibility.GetAnimationNode ()
lanimnode.KeyAdd(False)
FBSystem().Scene.Evaluate()

#Key 4
FBPlayerControl().Goto(FBTime(0,0,0,20))
lanimnode = lCube.Visibility.GetAnimationNode ()
lanimnode.KeyAdd(False)
FBSystem().Scene.Evaluate()

#Key 5
FBPlayerControl().Goto(FBTime(0,0,0,21))
lanimnode = lCube.Visibility.GetAnimationNode ()
lanimnode.KeyAdd(True)
FBSystem().Scene.Evaluate()

#Key 6
FBPlayerControl().Goto(FBTime(0,0,0,30))
lanimnode = lCube.Visibility.GetAnimationNode ()
lanimnode.KeyAdd(True)
FBSystem().Scene.Evaluate()

#Key 7
FBPlayerControl().Goto(FBTime(0,0,0,31))
lanimnode = lCube.Visibility.GetAnimationNode ()
lanimnode.KeyAdd(False)
FBSystem().Scene.Evaluate()

#Key 8
FBPlayerControl().Goto(FBTime(0,0,0,40))
lanimnode = lCube.Visibility.GetAnimationNode ()
lanimnode.KeyAdd(False)
FBSystem().Scene.Evaluate()
```

Opposed to this example which does the same thing but is more optimized, and you do not use the Evaluate function:

```
from pyfbsdk import *

#Find Model
lCube=FBFindModelByLabelName("Cube")
if not lCube.Visibility.IsAnimated():
        lCube.Visibility.SetAnimated(True)

if not lCube.Visibility.IsFocused():
        lCube.Visibility.SetFocus(True)
```

```
#Key 1
lCube.Visibility.GetAnimationNode ().KeyAdd(FBTime(3,3,3,0),True)
#Key 2
lCube.Visibility.GetAnimationNode ().KeyAdd(FBTime(0,0,0,10),True)
#Key 3
lCube.Visibility.GetAnimationNode ().KeyAdd(FBTime(0,0,0,11),False)
#Key 4
lCube.Visibility.GetAnimationNode ().KeyAdd(FBTime(0,0,0,20),False)
#Key 5
lCube.Visibility.GetAnimationNode ().KeyAdd(FBTime(0,0,0,21),True)
#Key 6
lCube.Visibility.GetAnimationNode ().KeyAdd(FBTime(0,0,0,30),True)
#Key 7
lCube.Visibility.GetAnimationNode ().KeyAdd(FBTime(0,0,0,31),False)
#Key 8
lCube.Visibility.GetAnimationNode ().KeyAdd(FBTime(0,0,0,40),False)
```

You're not able to read the right values even though you have updated them

A lot time you update values and you try and read them back but you are not getting the correct values

```
from pyfbsdk import *

lCube = FBFindModelByLabelName('Cube')

lCube.Translation = FBVector3d(10,10,10)
ltrans = lCube.Translation
print "Cube Translation: %f,%f,%f" % (ltrans[0], ltrans[1], ltrans[2])

lTime = FBTime(0,0,0,0)
lCube.Translation.SetAnimated(True)
lAnimationNode = lCube.Translation.GetAnimationNode()
lPositionV = FBVector3d(
    lAnimationNode.Nodes[0].FCurve.Evaluate(lTime),
    lAnimationNode.Nodes[1].FCurve.Evaluate(lTime),
    lAnimationNode.Nodes[2].FCurve.Evaluate(lTime))

print "current translation is %f,%f,%f" % (lPositionV[0], lPositionV[1],
lPositionV[2])
```

7.2 FCurve

Nodes may have an FCurve (FBFCurve) associated with them. FCurves let you have more control over you animation to modify the animation by adjusting its individual key frames and their interpolation, extrapolation, and tangents.

This example shows you how to work with animation and manipulate it to get the data you need:

```
from pyfbsdk import *

lModel = FBFindModelByLabelName('Cube')
for lAnim in lModel.AnimationNode.Nodes:
    if lAnim.Name == 'Lcl Translation':
        print lAnim.Name

        for lAnimComp in lAnim.Nodes:
            lMsg = lAnimComp.Name
            lCurve = lAnimComp.FCurve

            if lCurve:
                lNumKeys = len(lCurve.Keys)
                lMsg += ' has an FCurve with '+str(lNumKeys)+' keys'
            else:
                lMsg += ' has no FCurve'
            print '   ' + lMsg
```

7.3 Time

This class is used very frequently, however can be very misleading, because you think that it would be just as easy as reading the time on a clock, however due to all the different formats of time, this class tends not be as straight forward as users expect.

System time relating to actual system time

It's a bit challenging to discover the actual time of day it is, because you need to use the class FBReferenceTime, which isn't documented and not self explanatory, here is how one would get the actual time of day it is from the system time in FBSystem.

```
from pyfbsdk import *

#First you get the system time
sysTime = FBSystem().SystemTime;
```

```
#Second you get the Reference time using the system time which gives you the
actual //system time Index '0' for the FBReferenceTime is the System Time.
lReference = FBReferenceTime()
lRefTime = lReference.GetTime( 0, sysTime );

#I have chose to display in milliseconds...
print lRefTime.GetMilliSeconds()
```

My real life example:
So at 11:57 it prints out 86221242 which is correct if you do the math.

Break down:
11:57 = 57 minutes + (11 * 60) + (12 * 60) = 1437 minutes * 60 = 86220 seconds * 1000 = 86220000 milliseconds

86221242 = ~86220000 milliseconds

Note: they're not exactly the same because I am only calculating visual from my computer clock and its lowest value is minutes.

GetMilliSeconds () and SetMilliSeconds (long pMilliSeconds)

Want to work with time in a milli second format:

```
from pyfbsdk import *

# Get a timestamp before the sleep.
lTs1 = FBSystem().SystemTime

# Do sleep for a few milliseconds.
lSleepTimeMS = 1234
FBSleep( lSleepTimeMS )

# Get a second timestamp after the sleep.
lTs2 = FBSystem().SystemTime

# Now display the actual sleep time.
FBMessageBox( "Actual sleep time",
            "Sleep time requested: %.3f seconds\nActual sleep time   : %.3f
seconds" %(( lSleepTimeMS / 1000.0 ),
              (( lTs2.GetMilliSeconds() - lTs1.GetMilliSeconds() ) / 1000.0 )),
            "OK",
             None,
           None )
```

KLonglong is an _int64, so it's impossible to set and retrieve anything else than a whole number from the GetMilliseconds() and SetMilliseconds() functions.

You should use FBTime::GetSecondDouble () and FBTime::SetSecondDouble () function to achieve this level of precision.

GetSecondDouble () and SetSecondDouble

Want to work with time in a double format:

```
from pyfbsdk import *

print(FBSystem().LocalTime.GetSecondDouble())
```

GetTimeString () and SetTimeString ()

Want to work with time in a string format:

```
from pyfbsdk import *

lSystem = FBSystem().Scene

#Start and End Frame
print "StartFrame: " +
(lSystem.Takes[0].LocalTimeSpan.GetStart().GetTimeString() )
print "EndFrame: " +
(lSystem.Takes[0].LocalTimeSpan.GetStop().GetTimeString() )
```

7.4 Player Control (aka Transport Control)

This class serves as the interface to the transport controls.

This is typically used to change the time so that the value of a property at that time can be obtained. Before obtaining the value, you must call FBScene::Evaluate() so the values are up-to-date.

From this class you can set anything that you would set in the Transport Control to effect the animation of your scene. As well as step through your scene frame by frame.

Example

```
from pyfbsdk import *
```

```
lModel = FBFindModelByLabelName('Cube')
lControl = FBPlayerControl()

#Lets start at frame 1.
lStart = FBTime(0, 0, 0, 1)
lControl.Goto(lStart)

for i in range(1, 10):
    FBSystem().Scene.Evaluate()
    lTrans = FBVector3d()
    # Get global translation.
    lModel.GetVector(lTrans, FBModelTransformationType.kModelTranslation, True)
    print "%d: (%f, %f, %f)" % (i, lTrans[0], lTrans[1], lTrans[2])
    lControl.StepForward()
```

7.5 Filters

A filter is used to adjust or modify animation keyframes on a function curve.

When working with Filters there two main classes you need to take in to account FBFilterManager and FBFilter. FBFilterManager manages all the filters created in FBFilter.
The existing set of filters can be extended by deriving new filters from the FBFilter class. These filters can be applied either in the user interface or within the scope of another plug-in. Filters are objects which can be applied on an FCurve, or the animation node associated with an animated object property, to modify shape and number of keys. Filters can be created from the GUI, using the Filters tool, or programmatically with an instance of a FBFilterManager.
Instances of FBFilter should be created with the help of the class FBFilterManager. Only internal application code should call the FBFilter's class constructor.

FBFilteManager class provides list of all available filter types and a factory method in order to create an instance of the desired filter type. This manager will list both built-in and plug-in filters. Filter type names are not localized, and are the same as presented in the GUI.

Creating a Key Reducing Filter

```
from pyfbsdk import *

lModel = FBFindModelByLabelName("Cube")

lFilterManager = FBFilterManager()
lFilter = lFilterManager.CreateFilter( "key Reducing" )
```

```
lFilter.PropertyList.Find ( 'Start' ).Data = FBTime(0,0,0,0)
lFilter.PropertyList.Find ( 'Stop' ).Data = FBTime(0,0,0,120)

lFilter.PropertyList.Find("Precision").Data = 10.0
print lFilter.PropertyList.Find("Precision").Data

lFilter.Apply(lModel.Translation.GetAnimationNode(), 1)
```

Accessing Filter Properties for MotionBuilder Filters

The filter properties can be found in the object's PropertyList data member. They will use the same name, and be of the same type, as what can be seen in the GUI.

Previously, Start\Stop Public Attribute was not defined and Attribute "ResampleFrameRate" of Reinterpolate didn't work. For example, calling lFilter.PropertyList.Find ('Key Sync').Data = True did not set the Key Sync property to true.

The following filter properties now work correctly: Start\Stop (common)
lFilter.PropertyList.Find ('Start').Data = FBTime(0, 0, 0, 1)
lFilter.PropertyList.Find ('Stop').Data = FBTime(0, 0, 0, 10)

Butterworth
lFilter.PropertyList.Find ('Cut-off Frequency(Hz)').Data = 7.0
lFilter.PropertyList.Find ('Key On Frame') = True

Constant Key Reducer
lFilter.PropertyList.Find ('Keep At Least One Keyframe').Data = False

Key Reducing
lFilter.PropertyList.Find ('Key Sync').Data = True

KeysOnFrame
lFilter.PropertyList.Find ('Frame Rate').Data = 120.00

Reinterpolate
lFilter.PropertyList.Find ('Resample').Data = True
lFilter.PropertyList.Find ('Resample Frame Rate').Data = 120.0

Resample
lFilter.PropertyList.Find ('Frame Rate').Data = True
lFilter.PropertyList.Find ('Keys On Frame').Data = True
lFilter.PropertyList.Find ('With interpolation').Data = True
Smooth
lFilter.PropertyList.Find ('Width').Data = 4

```
lFilter.PropertyList.Find ( 'Sample Count' ).Data = 8.00
lFilter.PropertyList.Find ( 'Use Quaternions' ).Data = True
```

SmoothTranslation

```
lFilter.PropertyList.Find ( 'Width' ).Data = 8
lFilter.PropertyList.Find ( 'Factor' ).Data = 1.50
lFilter.PropertyList.Find ( 'Sample Count' ).Data = 8.00
```

Transformation

```
lFilter.PropertyList.Find ( 'Translation' ).Data = (0.00, 0.00, 0.00)
lFilter.PropertyList.Find ( 'Rotation' ).Data = (0.00, 0.00, 0.00)
lFilter.PropertyList.Find ( 'Scaling' ).Data = (0.00, 0.00, 0.00)
```

UnrollRotation

```
lFilter.PropertyList.Find ( 'Quality' ).Data = 0.25
lFilter.PropertyList.Find ( 'Path' ).Data = True
```

Creating an Custom OR SDK Filter

The following sample code shows how to use Python to create an instance of the orfilter_template filter and set one of its properties. For the sample code to work, the plug-in must have been compiled and copied in the plug-in folder prior to the application startup.

```
from pyfbsdk import *

# Create a filter of a known type. In this case the sample filter
# provided with the samples: orfilter_template.
lFilterManager = FBFilterManager()
lFilter = lFilterManager.CreateFilter( 'OR - Filter Template' );

# Set one of the filter property:
lPropDouble = lFilter.PropertyList.Find( 'Test Double' );
if lPropDouble: lPropDouble.Data = 2.0

# Now we can apply the filter on an FCurve.
# And when we are done, destroy it.
lFilter.FBDelete()
```