SofaPython plugin

Bruno Carrez

January 28, 2014

Abstract

The goal of this plugin is to provide python integration into SOFA.

Therefore, it is possible to add new features in SOFAwithout modifying SOFAcodebase. Moreover, python scripts are stored in the same place as the corresponding scene, for a cleaner integration of scene-specific code without polluting the rest of SOFAcodebase.

This plugin provides a Sofa module available from python scripts, to interact with SOFA.

1 SofaPython graph components

1.1 PythonScriptController

This controller is an empty Controller, its behavior is driven in the Python script it loads. This is the simplest and the most polyvalent way to imbed Python scripting in a SOFAscene graph.

2 SofaPython scene files

It is possible, once the plugin is charged in runSofa, to load python script *.py files directly instead of xml *.scn files. This way, you can create scene graphs in a procedural way instead of a descriptive way.
An example of this is provided in the examples/PythonScene.py sample scene of the plugin.

3 Creating graph objects in Python

Creating objects in Python is as simple as describing them in XML scene files. You can use either Sofa.createObject(typename,...) or Sofa.BaseContext.createObject(typename,...) to do it (the second creates the object THEN adds it to the context, whereas the first creates an "orphan" object taht you must add later to a context.)
Both functions take a string for the type name of the object to create, then any optional named parameters neede to complete the object description.
Here is an example of xml file, and its equivament in SofaPython code: (xml is commented)

```
#        <EulerImplicit name="cg_odesolver" printLog="false" />
    node.createObject('EulerImplicit',name='cg_odesolver',printLog='false')
#        <CGLinearSolver iterations="25" name="linear solver" tolerance="1.0e-9"
  threshold="1.0e-9" />
    node.createObject('CGLinearSolver',name='linear
        solver', iterations=25,tolerance=2.0e-9,threshold=1.0e-9)
#        <MechanicalObject />
    node.createObject('MechanicalObject')
#        <UniformMass totalmass="10" />
    node.createObject('UniformMass',name='mass',totalmass=10)
#        <RegularGrid nx="6" ny="5" nz="3" xmin="-11" xmax="11" ymin="-7" ymax="7"
  zmin="-4" zmax="4" />
    node.createObject('RegularGrid', name='RegularGrid1', nx=6, ny=5, nz=3, xmin=-11,
        xmax=11, ymin=-7, ymax=7, zmin=-4, zmax=4)
#        <RegularGridSpringForceField name="Springs" stiffness="350" damping="1" />
    node.createObject('RegularGridSpringForceField',name='Springs', stiffness='350', damping='1')

#        <Node name="VisuDragon" tags="Visual">
    VisuNode = node.createChild('VisuDragon')
#            <OglModel name="Visual" filename="mesh/dragon.obj" color="red" />
    VisuNode.createObject('OglModel',name='Visual',filename='mesh/dragon.obj', color=color)
#            <BarycentricMapping object1="../.." object2="Visual" />
    VisuNode.createObject('BarycentricMapping',object1='../..', object2='Visual' )
#        </Node>

#        <Node name="Surf">
    SurfNode = node.createChild('Surf')
#            <MeshObjLoader name="loader" filename="mesh/dragon.obj" />
    SurfNode.createObject('MeshObjLoader', name='loader', filename='mesh/dragon.obj')
#            <Mesh src="@loader" />
    SurfNode.createObject('Mesh',src='@loader')
#            <MechanicalObject src="@loader" />
    SurfNode.createObject('MechanicalObject', name='meca', src='@loader')
#            <Triangle />
    SurfNode.createObject('Triangle')
#            <Line />
    SurfNode.createObject('Line')
#            <Point />
    SurfNode.createObject('Point')
#            <BarycentricMapping />
    SurfNode.createObject('BarycentricMapping')
#        </Node>
```

Remark: the values given for each optional named parameter can be a string, a scalar or an integer; this way you can flawlessly use computed variables, or even method calls to initialize your components.

4 SOFA Python API

4.1 The Sofa module

The core of this plugin is the Sofa Python module available to python scripts from within the SofaPython components (they are not available outside Sofa environment, in the command-line python binary for example).
Therefore, each python script to be imbedded in SOFAshould include the following line if it wants to interact with the SOFAframework:

```
1 import Sofa
```

This module provides a wide range of methods and types, bound to essential SOFA framework features.
These can be used from python within scripts loaded by the components provided by this plugin.

4.2 Module methods

SofaPython provides several module methods, for general purpose (not linked to a particular node or component, for example).

- sendGUIMessage(msgType,msgValue)
- createObject(type[, datafield=value [,...]]) creates an object. ex: node.createObject('UniformMass',totalmass=1) See "Creating graph objects in Python" section for more info.

4.3 Types hierarchy

The class hierarchy in the Sofa module is quite different from the C++ SOFA class hierarchy. Not all SOFA classes are bound in Python, and some levels in the hierarchy are skipped.
Despite the ability of Python to support multi-heritage, this feature has not been implemented in the Sofa module, for code simplicity.

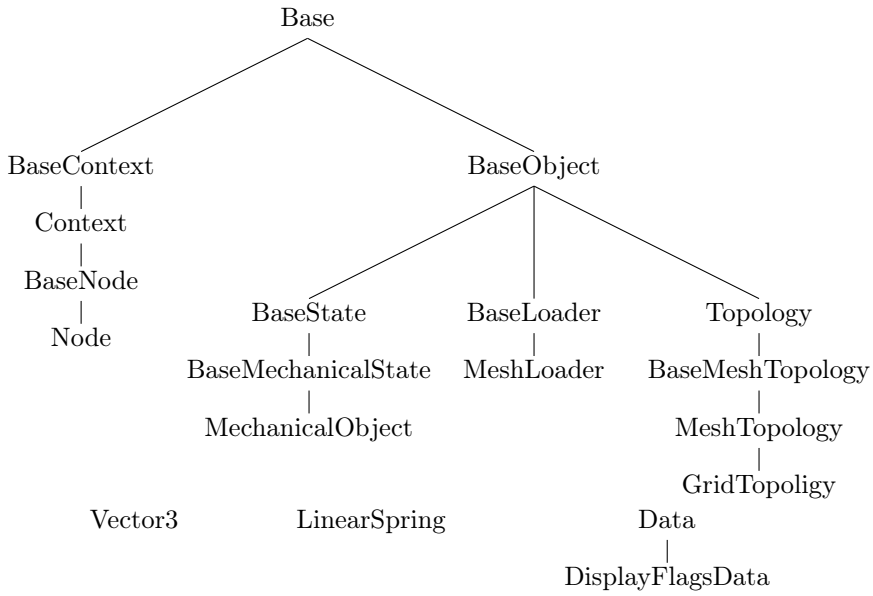

Figure 1: Sofa python types hierarchy

4.3.1 Sofa.Vector3

Attributes :

- x
- y
- z

4.3.2 Sofa.LinearSpring

Attributes :

- Index1 : the first extremity of the spring
- Index2 : the second extremity of the spring
- Ks : spring stiffness
- Kd : damping factor
- L : rest length of the spring

4.3.3 Sofa.Topology

Methods :

- haspos()
- getNbPoints()
- setNbPoints(n)
- getPX(i)
- getPY(i)
- getPZ(i)

4.3.4 Sofa.BaseMeshTopology

Methods :

- getNbEdges()
- getNbTriangles()
- getNbQuads()
- getNbTetrahedra()
- getNbHexahedra()

4.3.5 Sofa.GridTopology

Methods :

- setSize(nx,ny,nz)
- setNumVertices(nx,ny,nz)
- getNx()
- getNy()
- getNz()
- setNx(n)
- setNy(n)
- setNz(n)

4.3.6 Sofa.RegularGridTopology

Methods :

- setPos(xmin,xmax,ymin,ymax,zmin,zmax))

4.3.7 Sofa.Base

Methods :

- findData(name) returns a Sofa.Data object (if it exists)

Attributes:

- name

4.3.8 Sofa.BaseObject

Methods :

- init()
- bwdInit()
- reinit()
- storeResetState()
- reset()
- cleanup()
- getContext() returns a Sofa.BaseContext object
- getMaster() returns a Sofa.BaseObject object
- setSrc(valueString, loader) use a Sofa.BaseLoader to initialize the object

4.3.9 Sofa.BaseState

Methods :

- resize(size)

4.3.10 Sofa.BaseMechanicalState

Methods :

- applyTranslation(x,y,z)
- applyRotation(x,y,z) Euler angles in degrees.
- applyScale(x,y,z)

4.3.11 Sofa.MechanicalObject

Methods :

- setTranslation(x,y,z) Initial transformations accessor.
- setRotation(x,y,z) Initial transformations accessor. Euler angles in degrees.
- setScale(x,y,z) Initial transformations accessor.
- getTranslation() returns a Sofa.Vector3 object. Initial transformations accessor.
- getRotation() returns a Sofa.Vector3 object. Initial transformations accessor. Euler angles in degrees.
- getScale() returns a Sofa.Vector3 object. Initial transformations accessor.

4.3.12 Sofa.BaseContext

Methods :

- getRootContext()
- getTime()
- getDt()
- getGravity() returns a Sofa.vec3 object
- setGravity(Vec3)
- createObject(type[, datafield=value [,...]]) creates an object. ex: node.createObject('UniformMass',totalmass=1) See "Creating graph objects in Python" section for more info.
- getObject(path)

Attributes:

- active (boolean)
- animate (boolean)

4.3.13 Sofa.Context

(empty)

4.3.14 Sofa.Node

- createChild(childName)
- getRoot() returns the root node of the graph (if any)
- getChild(path) returns a child node, given its path (if any)
- getChildren() returns the list of children
- getParents() returns the list of parents
- executeVisitor(visitor) executes a python visitor from this node. See "Visitors" section for more infos.
- simulationStep(dt) executes ONE step of simulation, using dt (in seconds, float) as delta time.
- init() Initialize the scene
- reset() Reset the scene
- addObject(object)
- removeObject(object)
- addChild(childNode)
- removeChild(childNode)
- moveChild(childNode)
- detachFromGraph()
- sendScriptEvent(eventName, data) propagates a script event from this node; eventName is the name of the event (string) and data is whatever you want: scalar, float, array, ... as long as it is a python object or structure. Other PythonScriptController will then receive the correponding onScriptEvent(senderName, eventName, data). Useful for inter-script communications (see ScriptEvent.scn sample scene in the examples folder).

4.3.15 Sofa.Data

Attributes :

- value
- name

Methods :

- getValue(index)
- setValue(index,value)

4.3.16 Sofa.DisplayFlagsData

This class is always used in a "VisualStyle' object in the root node; see example below. It is used to control what should be displayed.
Attributes :

- showAll
- showVisual
- showVisualModels
- showBehavior
- showBehaviorModels
- showForceFields
- showInteractionForceFields
- showCollision
- showCollisionModels
- showBoundingCollisionModels
- showMapping
- showMappings
- showMechanicalMappings
- showOptions
- showWireFrame
- showNormals

Unlike in C++ code, all these attributes are boolean values for simplicity. Some attributes (like showAll) are hierarchicaly above others: setting them to True or False will set children attributes also; reading them will return True if their real C++ state is true or neutral.
Simple example of use:

```
1  style = node.getRoot().createObject('VisualStyle')
2  style.findData('displayFlags').showBehaviorModels = True
```

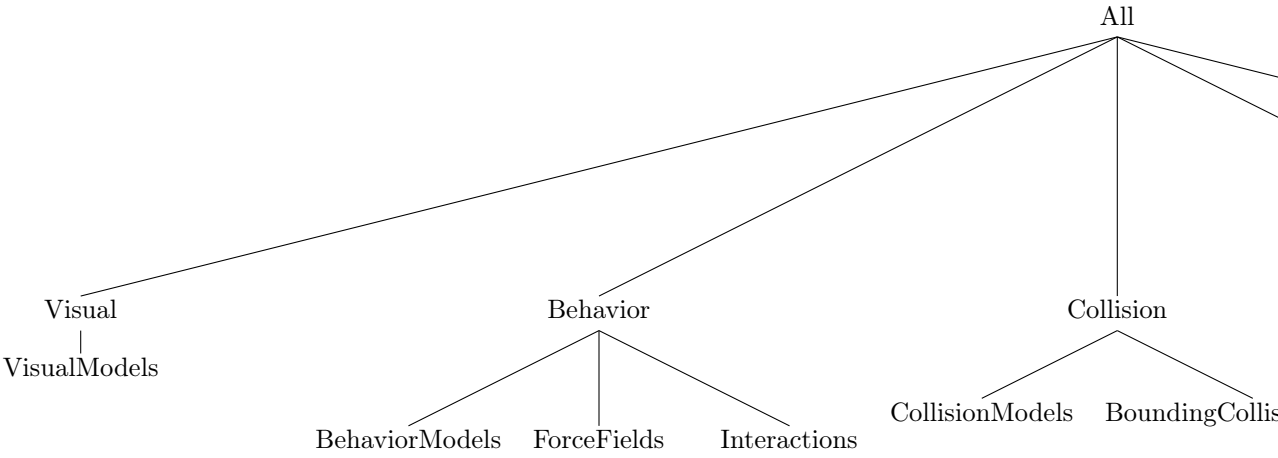

Figure 2: display flags hierarchy

4.3.17 Data members: the most important thing in SOFAPython API

The most important class is Sofa.Base, and its associated method findData. ALMOST EVERYTHING in SOFAis stored in Datas, and with the only Sofa.Base.findData method, almost everything is possible. Through the Sofa.Data class (returned by Sofa.Base.findData(name)) it is possible to read or write almost any object value, thus interact with the simulation in real-time. This way, even if a specific component isn't bound to python, it's possible to access it by its Sofa.Base heritage.
Data.value attribute has a versatile behavior, depending on the Data type. On read, Data.value can return either an integer, a float, a string, or even a list of one of these 3 types. On write, you have to set EXACTLY the proper type, or you can set a string (same format as in the *.scn xml files).
Examples:

```
print str(node.findData('gravity').value
```

will output the text conversion of a list of 3 floats :

```
[0.0, -9.81, 0.0]
```

You can set it in two ways ; the native version :

```
node.findData('gravity').value = [0.0, -9.81, 0.0]
```

or by the text version:

```
node.findData('gravity').value = '0.0 -9.81 0.0'
```

Use of any other type will result in an error. The following won't work for example :

```
node.findData('gravity').value = 9.81
```

5 Visitors

5.1 Example visitor implementation in Python

Following is a very simple visitor implementation in a python script: (it is provided in the examples/Visitor.scn sample scene of the plugin)

```
class SofaVisitor(object):
 def __init__(self,name):
  print 'SofaVisitor constructor name='+name
   self.name = name

 def processNodeTopDown(self,node):
  print 'SofaVisitor "'+self.name+'" processNodeTopDown node='+node.findData('name').value
  return True

 def processNodeBottomUp(self,node):
  print 'SofaVisitor "'+self.name+'" processNodeBottomUp node='+node.findData('name').value
```

Your visitor class MUST implement the two methods processNodeTopDown and processNodeBottomUp.
In this example it is only meant to go through the graph in both directions, listing all node names.
The scene graph is the following nodes tree:

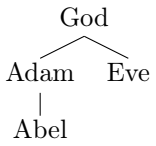

Figure 3: Scene graph of the Visitor sample.

The Visitor is executed on the "God" node:

```
v = SofaVisitor('PythonVisitor')
node.executeVisitor(v)
```

and the console output result is:

```
SofaVisitor constructor name=PythonVisitor SofaVisitor "PythonVisitor"
processNodeTopDown node=god SofaVisitor "PythonVisitor"
processNodeTopDown node=Adam SofaVisitor "PythonVisitor"
processNodeTopDown node=Abel SofaVisitor "PythonVisitor"
processNodeBottomUp node=Abel SofaVisitor "PythonVisitor"
processNodeBottomUp node=Adam SofaVisitor "PythonVisitor"
processNodeTopDown node=Eve SofaVisitor "PythonVisitor"
processNodeBottomUp node=Eve SofaVisitor "PythonVisitor"
processNodeBottomUp node=god
```

6 Sofa "PythonScriptController" component

6.1 Example scene

```
1 runSofa applications/plugins/SofaPython/examples/ExampleController.scn
```

6.2 Component data

- filename is the name of the .py script file.
- classname is the name of the class in the script to instanciate for the controller. NOTE: the class MUST inherit from Sofa.PythonScriptController. See example files.

6.3 Python script entry points

- onKeyPressed(self,c)
- onKeyReleased(self,c)
- onLoaded(self,node)
- onMouseButtonLeft(self,mouseX,mouseY,isPressed)
- onMouseButtonRight(self,mouseX,mouseY,isPressed)
- onMouseButtonMiddle(self,mouseX,mouseY,isPressed)
- onMouseWheel(self,mouseX,mouseY,wheelDelta)
- onGUIEvent(self,strControlID,valueName,strValue)
- onBeginAnimationStep(self,deltaTime)
- onEndAnimationStep(self,deltaTime)
- onScriptEvent(self,senderNode, eventName, data)
- createGraph(self,node)
- initGraph(self,node)
- bwdInitGraph(self,node)
- storeResetState(self)
- reset(self)
- cleanup(self)