

SofaPython plugin

Bruno Carrez

July 18, 2012

Abstract

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

Therefore, it is possible to add new features in SOFA without 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 SOFA Python API

2.1 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 SOFA should include the following line if it wants to interact with the SOFA framework:

```
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.

2.2 Module methods

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

- `sendGUIMessage(msgType,msgValue)`

2.3 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.

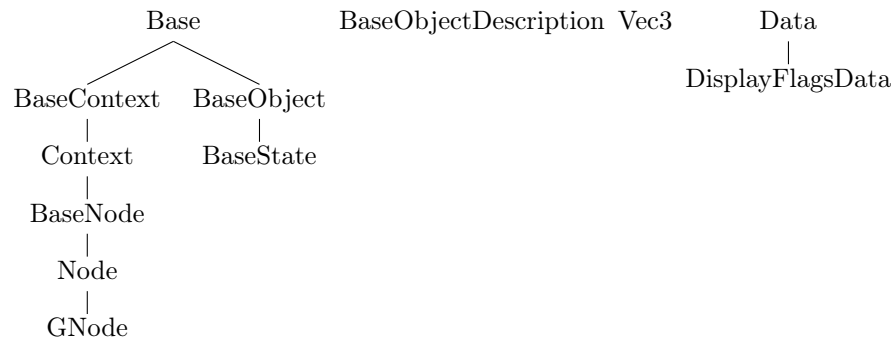


Figure 1: Sofa python types hierarchy

2.3.1 Sofa.BaseObjectDescription

Used to create objects with the Method `Sofa.createObject` :

```

1 desc = Sofa.BaseObjectDescription(' particle ', 'MechanicalObject')
2 particle = Sofa.createObject(node,desc)

```

Methods :

- `setAttributes`
- `getAttributes`

Constructor :

- `BaseObjectDescription(name,type)`

Attributes :

- `name`

2.3.2 Sofa.Vec3

Attributes :

- `x`
- `y`
- `z`

2.3.3 Sofa.Base

Methods :

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

Attributes:

- `name`

2.3.4 Sofa.BaseObject

(empty)

2.3.5 Sofa.BaseState

Methods :

- `resize(size)`

2.3.6 Sofa.BaseContext

Methods :

- `getRootContext()`
- `getTime()`
- `getDt()`
- `getGravity()` returns a `Sofa.vec3` object
- `setGravity(Vec3)`
- `createObject(BaseObjectDescription)`
- `getObject(path)`

Attributes:

- `active` (boolean)
- `animate` (boolean)

2.3.7 Sofa.Context

(empty)

2.3.8 Sofa.Node

- `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.

2.3.9 Sofa.Gnode

Methods :

- createChild(childName)
- addChild(childNode)
- removeChild(childNode)
- moveChild(childNode)
- addObject(object) objets created with Sofa.createObject method (see example scene)
- removeObject(object)
- detachFromGraph()

2.3.10 Sofa.Data

Attributes :

- value
- name

Methods :

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

2.3.11 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 hierarchically 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.

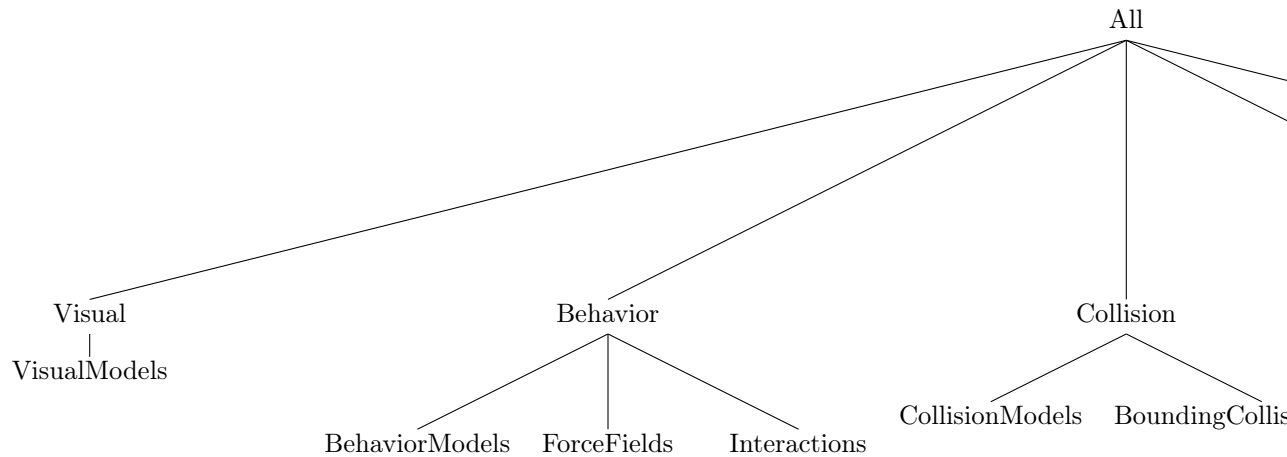


Figure 2: display flags hierarchy

Simple example of use:

```

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

```

2.3.12 Data members: the most important thing in SOFAPython API

The most important class is `Sofa.Base`, and its associated method `findData`. ALMOST EVERYTHING in SOFA is 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:

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

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

```
1 [0.0, -9.81, 0.0]
```

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

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

or by the text version:

```
1 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 :

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


3 Visitors

3.1 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)

```
1 class SofaVisitor (object):
2     def __init__(self,name):
3         print 'SofaVisitor constructor name='+name
4         self.name = name
5
6     def processNodeTopDown(self,node):
7         print 'SofaVisitor "' +self.name+" processNodeTopDown node='"+node.findData('name').value
8         return True
9
10    def processNodeBottomUp(self,node):
11        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:

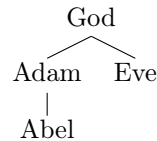


Figure 3: Scene graph of the Visitor sample.

The Visitor is executed on the "God" node:

```
1 v = SofaVisitor('PythonVisitor')
2 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
```

4 Sofa "PythonScriptController" component

4.1 Example scene

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

4.2 Component data

- filename is the name of the .py script file.

4.3 Python script entry points

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