

---

# **STLIB Documentation**

***Release 1.0***

**damien.marchal@univ-lille1.fr**

**Aug 22, 2023**



---

## Contents

---

<b>1</b>	<b>splib.algorithms</b>	<b>3</b>
<b>2</b>	<b>splib.animation</b>	<b>5</b>
2.1	Functions: . . . . .	5
2.2	Modules: . . . . .	6
<b>3</b>	<b>splib.loaders</b>	<b>7</b>
3.1	splib.loaders.loadPointListFromFile . . . . .	7
<b>4</b>	<b>splib.objectmodel</b>	<b>9</b>
<b>5</b>	<b>splib.scenegraph</b>	<b>11</b>
5.1	splib.scenegraph.find . . . . .	11
5.2	splib.scenegraph.get . . . . .	11
<b>6</b>	<b>splib.units</b>	<b>13</b>
6.1	splib.units.time . . . . .	13
6.2	splib.units.material . . . . .	13
6.3	splib.units.units . . . . .	14
<b>7</b>	<b>Sofa Template Library</b>	<b>15</b>
7.1	stlib.physics . . . . .	16
7.2	stlib.visuals . . . . .	20
7.3	stlib.solver . . . . .	20
7.4	stlib.scene . . . . .	21
7.5	stlib.tools . . . . .	23
	<b>Python Module Index</b>	<b>25</b>
	<b>Index</b>	<b>27</b>



Python side of the framework [Sofa](#).

Example:

TODO
------

<i>splib.algorithms</i>	Algorithms we often use.
<i>splib.animation</i>	Animation framework focusing in ease of use.
<i>splib.caching</i>	
<i>splib.debug</i>	
<i>splib.geometric</i>	
<i>splib.loaders</i>	Utility function to ease the writing of scenes.
<i>splib.numerics</i>	
<i>splib.objectmodel</i>	
<i>splib.scenegraph</i>	Algorithms we often use.
<i>splib.units</i>	Toolbox to manipulates units (time, mechanics, material)



# CHAPTER 1

---

splib.algorithms

---

Algorithms we often use.

**Content:**





Animation framework focusing in ease of use.

## 2.1 Functions:

<code>animate(onUpdate, params, duration[, mode, ...])</code>	Construct and starts an animation
<code>AnimationManager(node)</code>	A Controller to manage all animations in the scene
<code>AnimationManagerController</code>	MagicMock is a subclass of Mock with default implementations of most of the magic methods.

### 2.1.1 splib.animation.animate

`splib.animation.animate` (*onUpdate*, *params*, *duration*, *mode*='once', *onDone*=None)

Construct and starts an animation

Build a new animation from a callback function that computes the animation value, a set of parameters, the animation duration and the type of animation repetition pattern.

Animation can be added from any code location (`createScene`, `PythonScriptController`)

#### Parameters

- **duration** (*float*) – duration of the animation in seconds.
- **mode** (*str*) – once, loop, pingpong

#### Example:

```
def myAnimate(target, factor):
    print("I should do something on: "+target.name)

def createScene(rootNode)
```

(continues on next page)

(continued from previous page)

```
AnimationManager(rootNode)
animate(myAnimate, {"target" : rootNode }, 10)
```

## 2.1.2 splib.animation.AnimationManager

`splib.animation.AnimationManager` (*node*)

A Controller to manage all animations in the scene

Before using the animation framework an AnimationManager must be added to the scene. It has in charge, at each time step to update all the running animations.

**Returns:** AnimationManagerController

**Example:**

```
def createScene(rootNode)
    AnimationManager(rootNode)
```

## 2.1.3 splib.animation.AnimationManagerController

`splib.animation.AnimationManagerController`

## 2.2 Modules:

---

*splib.animation.easing*

Easing function to use in animation

---

### 2.2.1 splib.animation.easing

Easing function to use in animation

---

*LinearRamp*([beginValue, endValue, scale])

Linear interpolation between two values

---

#### `splib.animation.easing.LinearRamp`

`splib.animation.easing.LinearRamp` (*beginValue=0.0, endValue=1.0, scale=0.5*)

Linear interpolation between two values

**Examples:** LinearRamp(10, 20, 0.5)

Utility function to ease the writing of scenes.

<code>loadPointListFromFile(s)</code>	Load a set of 3D point from a json file
<code>getLoadingLocation(filename[, source])</code>	Compute a loading path for the provided filename relative to a given source location

---

### 3.1 splib.loaders.loadPointListFromFile

`splib.loaders.loadPointListFromFile(s)`

Load a set of 3D point from a json file

`splib.loaders.getLoadingLocation(filename, source=None)`

Compute a loading path for the provided filename relative to a given source location

**Examples:**

```
getLoadingLocation("myfile.json") # returns "myfile.json"
getLoadingLocation("myfile.json", "toto") #returns "/fullpath/to/toto/myfile.json"
getLoadingLocation("myfile.json", __file__) #returns "/fullpath/to/toto/myfile.json"
```

The latter is really usefull to make get the path for a file relative to the 'current' python source.



## CHAPTER 4

---

splib.objectmodel

---



Algorithms we often use.

**Content:**

<code>find(node, path)</code>	Query a node or an object by its path from the provided node.
<code>get(node, path)</code>	Query a node, an object or a data by its path from the provided node.

## 5.1 splib.scenegraph.find

`splib.scenegraph.find (node, path)`

Query a node or an object by its path from the provided node.

**Example:** `find(node, "/root/rigidcube1/visual/OglModel")`

## 5.2 splib.scenegraph.get

`splib.scenegraph.get (node, path)`

Query a node, an object or a data by its path from the provided node.

**Example:** `find(node, "/root/rigidcube1/visual/OglModel.position")`





Toolbox to manipulates units (time, mechanics, material)

### Content:

<i>time</i>	Converting SI units to/from specific units Every units are given as a ratio to the SI units (m/kg/s)
<i>material</i>	Converting SI units to/from specific units Every units are given as a ratio to the SI units (m/kg/s)
<i>units</i>	Converting SI units to/from specific units Every units are given as a ratio to the SI units (m/kg/s)

## 6.1 splib.units.time

Converting SI units to/from specific units Every units are given as a ratio to the SI units (m/kg/s)

For each conversion a local unit can be given OR variables local\_time/local\_length/local\_mass are used by default (these variables can be set by the user to define local units)

Note that Poisson ratio and strain have no units

## 6.2 splib.units.material

Converting SI units to/from specific units Every units are given as a ratio to the SI units (m/kg/s)

For each conversion a local unit can be given OR variables local\_time/local\_length/local\_mass are used by default (these variables can be set by the user to define local units)

Note that Poisson ratio and strain have no units

## 6.3 splib.units.units

Converting SI units to/from specific units Every units are given as a ratio to the SI units (m/kg/s)

For each conversion a local unit can be given OR variables `local_time/local_length/local_mass` are used by default (these variables can be set by the user to define local units)

Note that Poisson ratio and strain have no units

- `genindex`
- `modindex`
- `search`

## Sofa Template Library

Utility functions and scene templates for the real-time simulation framework *Sofa*. The different templates are organized in types and abstract the complexity of object creation with *Sofa*.

The library is hosted on *github* <https://github.com/SofaDefrost/STLIB/> and it can be used with scenes written in python and PSL.

Example:

```
from stlib3.scene import MainHeader
from stlib3.solver import DefaultSolver
from stlib3.physics.rigid import Cube, Sphere, Floor
from stlib3.physics.deformable import ElasticMaterialObject

def createScene(rootNode):
    MainHeader(rootNode)
    DefaultSolver(rootNode)

    Sphere(rootNode, name="sphere", translation=[-5.0, 0.0, 0.0])
    Cube(rootNode, name="cube", translation=[5.0,0.0,0.0])

    ElasticMaterialObject(rootNode, name="dragon",
                          volumeMeshFileName="mesh/liver.msh",
                          surfaceMeshFileName="mesh/dragon.stl"
                          translation=[0.0,0.0,0.0])

    Floor(rootNode, name="plane", translation=[0.0, -1.0, 0.0])
```

### Content:

<i>stlib.physics</i>	Templates for physical simulation.
<i>stlib.visuals</i>	Templates for rendering.
<i>stlib.solver</i>	Templates for most of the common time integration setups.

Continued on next page

Table 1 – continued from previous page

<code>stlib.scene</code>	Templates for most of the common scene setups.
<code>stlib.tools</code>	

## 7.1 stlib.physics

Templates for physical simulation.

### Content:

<code>stlib.physics.constraints</code>	Templates for external constraints.
<code>stlib.physics.collission</code>	Templates to ease collision and contact handling.
<code>stlib.physics.deformable</code>	Templates for deformable objects.
<code>stlib.physics.mixedmaterial</code>	
<code>stlib.physics.rigid</code>	This package is focused on implementing a standard rigid object for sofa.

### 7.1.1 stlib.physics.constraints

Templates for external constraints.

#### Content

<code>FixedBox([applyTo, atPositions, name, ...])</code>	Constraint a set of degree of freedom to be at a fixed position.
<code>PartiallyFixedBox([attachedTo, box, ...])</code>	Constraint a set of degree of freedom to be at a fixed position.
<code>SubTopology([attachedTo, containerLink, ...])</code>	Args:

#### stlib.physics.constraints.FixedBox

`stlib.physics.constraints.FixedBox` (*applyTo=None, atPositions=[-1.0, -1.0, -1.0, 1.0, 1.0, 1.0], name='FixedBox', doVisualization=False, position=None, constraintStrength='1e12', doRecomputeDuringSimulation=False*)

Constraint a set of degree of freedom to be at a fixed position.

**Args:** `applyTo` (Sofa.Node): Node where the constraint will be applied

`atPosition` (vec6f): Specify min/max points of the font.

`name` (str): Set the name of the FixedBox constraint.

`doVisualization` (bool): Control whether or not we display the boxes.

Structure:

```
Node : {
  name : "fixedbox",
```

(continues on next page)

(continued from previous page)

```

BoxROI,
RestShapeSpringsFroceField
}

```

### stlib.physics.constraints.PartiallyFixedBox

```

stlib.physics.constraints.PartiallyFixedBox (attachedTo=None, box=[-1.0, -1.0, -1.0, 1.0, 1.0, 1.0], fixedAxis=[1, 1, 1], name='PartiallyFixedBox', drawBoxes=False, fixAll=False, doUpdate=False)

```

Constraint a set of degree of freedom to be at a fixed position.

**Args:** attachedTo (Sofa.Node): Node where the constraint will be applied

box (vec6f): Specify min/max points of the font.

fixedAxis (vec3bool): Specify which axis should be fixed (x,y,z)

name (str): Set the name of the FixedBox constraint.

drawBoxes (bool): Control whether or not we display the boxes.

fixAll (bool): If true will apply the partial fixed to all the points.

doUpdate (bool): If true

Structure:

```

Node : {
    name : "fixedbox",
    BoxROI,
    PartialFixedConstraint
}

```

### stlib.physics.constraints.SubTopology

```

stlib.physics.constraints.SubTopology (attachedTo=None, containerLink=None, boxRoiLink=None, linkType='tetrahedron', name='modelSubTopo', poissonRatio=0.3, youngModulus=18000)

```

**Args:**

attachedTo (Sofa.Node): Where the node is created.

containerLink (str): path to container with the dataField to link, ie: '@../container.position'

boxRoiLink (str): path to boxRoi with the dataField to link,

**linkType (str): indicate of which type is the subTopo**, ei: 'triangle' -> TriangleSetTopologyContainer & TriangleFEMForceField

name (str): name of the child node

youngModulus (float): The young modulus.

poissonRatio (float): The poisson parameter.

Structure:

```
Node : {  
    name : "modelSubTopo",  
    TetrahedronSetTopologyContainer,  
    TetrahedronFEMForceField  
}
```

## 7.1.2 stlib.physics.collision

Templates to ease collision and contact handling.

**Content:**

---

*CollisionMesh*([attachedTo,...])

---

### stlib.physics.collision.CollisionMesh

`stlib.physics.collision.CollisionMesh` (*attachedTo=None, surfaceMeshFileName=None, name='collision', rotation=[0.0, 0.0, 0.0], translation=[0.0, 0.0, 0.0], collisionGroup=None, mappingType='BarycentricMapping'*)

## 7.1.3 stlib.physics.deformable

Templates for deformable objects.

**Content:**

---

*ElasticMaterialObject* Creates an object composed of an elastic material.

---

### stlib.physics.deformable.ElasticMaterialObject

`class stlib.physics.deformable.ElasticMaterialObject` (*attachedTo=None, volumeMeshFileName=None, name='ElasticMaterialObject', rotation=[0.0, 0.0, 0.0], translation=[0.0, 0.0, 0.0], scale=[1.0, 1.0, 1.0], surfaceMeshFileName=None, collisionMesh=None, withConstrain=True, surfaceColor=[1.0, 1.0, 1.0], poissonRatio=0.3, youngModulus=18000, totalMass=1.0, solver=None*)

Creates an object composed of an elastic material.

```

createPrefab (volumeMeshFileName=None, name='ElasticMaterialObject', rotation=[0.0, 0.0,
0.0], translation=[0.0, 0.0, 0.0], scale=[1.0, 1.0, 1.0], surfaceMeshFileName=None,
collisionMesh=None, withConstrain=True, surfaceColor=[1.0, 1.0, 1.0], poissonRa-
tio=0.3, youngModulus=18000, totalMass=1.0, solver=None)

addCollisionModel (collisionMesh, rotation=[0.0, 0.0, 0.0], translation=[0.0, 0.0, 0.0], scale=[1.0,
1.0, 1.0])

addVisualModel (filename, color, rotation, translation, scale=[1.0, 1.0, 1.0])

cls
    alias of ElasticMaterialObject

definedloc = ('/home/docs/checkouts/readthedocs.org/user_builds/stlib/checkouts/latest

```

## 7.1.4 stlib.physics.rigid

This package is focused on implementing a standard rigid object for sofa. The object is made described by its surface mesh.

### Content:

<i>RigidObject</i> (node[, name, ...])	Creates and adds rigid body from a surface mesh.
<i>Cube</i> (node, **kwargs)	Create a rigid cube of unit dimension
<i>Sphere</i> (node, **kwargs)	Create a rigid sphere of unit dimension
<i>Floor</i> (node, **kwargs)	Create a rigid floor of unit dimension

### stlib.physics.rigid.RigidObject

```

stlib.physics.rigid.RigidObject (node, name='RigidObject', surfaceMeshFileName=None,
translation=[0.0, 0.0, 0.0], rotation=[0.0, 0.0, 0.0], uniform-
Scale=1.0, totalMass=1.0, volume=1.0, inertiaMatrix=[1.0,
0.0, 0.0, 0.0, 1.0, 0.0, 0.0, 0.0, 1.0], color=[1.0, 1.0, 0.0],
isAStaticObject=False)

```

Creates and adds rigid body from a surface mesh.

**Args:** surfaceMeshFileName (str): The path or filename pointing to surface mesh file.

totalMass (float): The mass is distributed according to the geometry of the object.

color (vec3f): The default color used for the rendering of the object.

translation (vec3f): Apply a 3D translation to the object.

rotation (vec3f): Apply a 3D rotation to the object in Euler angles.

uniformScale (vec3f): Apply a uniform scaling to the object.

isAStaticObject (bool): The object does not move in the scene (e.g. floor, wall) but react to collision.

### Structure:

```

Node : {
    name : "rigidobject"
    MechanicalObject,
    UniformMass,
    UncoupledConstraintCorrection,
    *EulerImplicit,
    *SparseLDLSolver,

```

(continues on next page)

(continued from previous page)

```
Node : {
    name : "collision",
    Mesh,
    MechanicalObject,
    Triangle,
    Line,
    Point,
    RigidMapping
}
Node : {
    name : "visual"
    OglModel,
    RigidMapping
}
}
```

### stlib.physics.rigid.Cube

stlib.physics.rigid.**Cube** (*node*, *\*\*kwargs*)  
Create a rigid cube of unit dimension

### stlib.physics.rigid.Sphere

stlib.physics.rigid.**Sphere** (*node*, *\*\*kwargs*)  
Create a rigid sphere of unit dimension

### stlib.physics.rigid.Floor

stlib.physics.rigid.**Floor** (*node*, *\*\*kwargs*)  
Create a rigid floor of unit dimension

## 7.2 stlib.visuals

Templates for rendering.

## 7.3 stlib.solver

Templates for most of the common time integration setups.

### Content:

---

<code>DefaultSolver</code> ( <i>node</i> [, <i>iterative</i> ])	Adds EulerImplicit, CGLinearSolver
---	------------------------------------

---



### 7.3.1 stlib.solver.DefaultSolver

`stlib.solver.DefaultSolver` (*node*, *iterative=True*)

Adds EulerImplicit, CGLinearSolver

**Components added:** EulerImplicit CGLinearSolver

## 7.4 stlib.scene

Templates for most of the common scene setups.

**Content:**

<i>Scene</i>	Scene(SofaObject) Create a scene with default properties.
<i>MainHeader</i> (node[, gravity, dt, plugins, ...])	Args:
<i>ContactHeader</i> (applyTo, alarmDistance, ...[, ...])	Args:
<i>Node</i> (parentNode, name)	Create a new node in the graph and attach it to a parent node.
<i>Wrapper</i> (node, attachedFunction, datacache)	Args:

### 7.4.1 stlib.scene.Scene

**class** `stlib.scene.Scene` (*SofaObject*)

Create a scene with default properties.

Arg:

node (Sofa.Node) the node where the scene will be attached

gravity (vec3f) the gravity of the scene

dt (float) the dt time

plugins (list(str)) set of plugins that are used in this scene

repositoryPath (list(str)) set of path where to read the data from

doDebug (bool) activate debugging facility (to print text)

There is method to add default solver and default contact management on demand.

**addSolver** ()

**addContact** (*alarmDistance*, *contactDistance*, *frictionCoef=0.0*)

**cls**

alias of *Scene*

**definedloc** = ('/home/docs/checkouts/readthedocs.org/user\_builds/stlib/checkouts/latest

## 7.4.2 stlib.scene.Interaction

**class** `stlib.scene.Interaction` (*parent, targets*)

Store a list of mechanical object to interact with

Args:

`targets` ([objects]) the object to interact with and that don't have a solver

**cls**

alias of `Interaction`

**definedloc** = ('/home/docs/checkouts/readthedocs.org/user\_builds/stlib/checkouts/latest

## 7.4.3 stlib.scene.MainHeader

`stlib.scene.MainHeader` (*node, gravity=[0.0, -9.8, 0.0], dt=0.01, plugins=[], repositoryPaths=[], do-Debug=False*)

**Args:** `gravity` (vec3f): define the gravity vector.

`dt` (float): define the timestep.

`plugins` (list str): list of plugins to load

`repositoryPaths` (list str): list of path to the specific data repository

**Structure:**

```
rootNode : {
    gravity : gravity,
    dt : dt,
    VisualStyle,
    RepositoryPath,
    RequiredPlugin,
    OglSceneFrame,
    FreeMotionAnimationLoop,
    GenericConstraintSolver,
    DiscreteIntersection
}
```

## 7.4.4 stlib.scene.ContactHeader

`stlib.scene.ContactHeader` (*applyTo, alarmDistance, contactDistance, frictionCoef=0.0*)

**Args:** `applyTo` (Sofa.Node): the node to attach the object to

**alarmDistance** (float): define the distance at which the contact are integrated into the detection computation.

**contactDistance** (float): define the distance at which the contact response is integrated into the computation.

**frictionCoef** (float, default=0.0): optional value, set to non-zero to enable a global friction in your scene.

**Structure:**

```

rootNode : {
    CollisionPipeline,
    BruteForceBroadPhase,
    BVHNarrowPhase,
    RuleBasedContactManager,
    LocalMinDistance
}

```

### 7.4.5 stlib.scene.Node

`stlib.scene.Node` (*parentNode*, *name*)

Create a new node in the graph and attach it to a parent node.

### 7.4.6 stlib.scene.Wrapper

**class** `stlib.scene.Wrapper` (*node*, *attachedFunction*, *datacache*)

**Args:** *node* : the current node we are working on

**attachedFunction** [the function that will be called at each object creation ] to do some stuff replace/insert/delete ...

This function will take as arguments the node, the type of the object to create, datacache & also the current arguments of the object .

This function as to return a tuple containing parameters of the object we want to create (ie: his type and a dictionary with all the other arguments) or None

*datacache* : the data we will use in our attachedFunction as parameters or else

**createObject** (*type*, *\*\*kwargs*)

**createChild** (*name*)

**\_\_getattr\_\_** (*value*)

## 7.5 stlib.tools

- `genindex`
- `modindex`
- `search`



### S

- splib, [1](#)
- splib.algorithms, [3](#)
- splib.animation, [5](#)
- splib.animation.easing, [6](#)
- splib.loaders, [7](#)
- splib.objectmodel, [9](#)
- splib.scenegraph, [11](#)
- splib.units, [13](#)
- splib.units.material, [13](#)
- splib.units.time, [13](#)
- splib.units.units, [14](#)
- stlib, [15](#)
- stlib.physics, [16](#)
- stlib.physics.collission, [18](#)
- stlib.physics.constraints, [16](#)
- stlib.physics.deformable, [18](#)
- stlib.physics.rigid, [19](#)
- stlib.scene, [21](#)
- stlib.solver, [20](#)
- stlib.tools, [23](#)
- stlib.visuals, [20](#)



## Symbols

`__getattr__()` (*stlib.scene.Wrapper method*), 23

## A

`addCollisionModel()`  
     (*stlib.physics.deformable.ElasticMaterialObject method*), 19

`addContact()` (*stlib.scene.Scene method*), 21

`addSolver()` (*stlib.scene.Scene method*), 21

`addVisualModel()` (*stlib.physics.deformable.ElasticMaterialObject method*), 19

`animate()` (*in module splib.animation*), 5

`AnimationManager()` (*in module splib.animation*), 6

`AnimationManagerController` (*in module splib.animation*), 6

## C

`cls` (*stlib.physics.deformable.ElasticMaterialObject attribute*), 19

`cls` (*stlib.scene.Interaction attribute*), 22

`cls` (*stlib.scene.Scene attribute*), 21

`CollisionMesh()` (*in module stlib.physics.collision*), 18

`ContactHeader()` (*in module stlib.scene*), 22

`createChild()` (*stlib.scene.Wrapper method*), 23

`createObject()` (*stlib.scene.Wrapper method*), 23

`createPrefab()` (*stlib.physics.deformable.ElasticMaterialObject method*), 18

`Cube()` (*in module stlib.physics.rigid*), 20

## D

`DefaultSolver()` (*in module stlib.solver*), 21

`definedloc` (*stlib.physics.deformable.ElasticMaterialObject attribute*), 19

`definedloc` (*stlib.scene.Interaction attribute*), 22

`definedloc` (*stlib.scene.Scene attribute*), 21

## E

`ElasticMaterialObject` (class *in stlib.physics.deformable*), 18

## F

`find()` (*in module splib.scenegraph*), 11

`FixedBox()` (*in module stlib.physics.constraints*), 16

`Floor()` (*in module stlib.physics.rigid*), 20

## G

`get()` (*in module splib.scenegraph*), 11

`getLoadingLocation()` (*in module splib.loaders*), 7

## I

`Interaction` (class *in stlib.scene*), 22

## L

`LinearRamp()` (*in module splib.animation.easing*), 6

`loadPointListFromFile()` (*in module splib.loaders*), 7

## M

`MainHeader()` (*in module stlib.scene*), 22

## N

`Node()` (*in module stlib.scene*), 23

## P

`PendulumFixedBox()` (*in module stlib.physics.constraints*), 17

## R

`RigidObject()` (*in module stlib.physics.rigid*), 19

## S

`Scene` (class *in stlib.scene*), 21

`Sphere()` (*in module stlib.physics.rigid*), 20

`splib` (module), 1

`splib.algorithms` (module), 3

`splib.animation` (module), 5

`splib.animation.easing` (module), 6

splib.loaders (*module*), 7  
splib.objectmodel (*module*), 9  
splib.scenegraph (*module*), 11  
splib.units (*module*), 13  
splib.units.material (*module*), 13  
splib.units.time (*module*), 13  
splib.units.units (*module*), 14  
stlib (*module*), 15  
stlib.physics (*module*), 16  
stlib.physics.collission (*module*), 18  
stlib.physics.constraints (*module*), 16  
stlib.physics.deformable (*module*), 18  
stlib.physics.rigid (*module*), 19  
stlib.scene (*module*), 21  
stlib.solver (*module*), 20  
stlib.tools (*module*), 23  
stlib.visuals (*module*), 20  
SubTopology () (*in module stlib.physics.constraints*),  
17

## W

Wrapper (*class in stlib.scene*), 23