zBuilder Documentation

Release 1.0.4

Ziva Dynamics

Apr 16, 2018

Contents

1 Introduction 3

2 Installation 5

3 zBuilder 7

4 zUI 19

5 Release Notes 21

6 Glossary 23

7 Module index 25

Python Module Index 53

Welcome to Ziva's VFX Utilities, a set of python utilities that have been designed to help leverage the benefits of the Ziva Character Simulation paradigm. You can find the BitBucket repo for these tools here.

Ziva Dynamics

zBuilder is a tool for extracting part of a Maya scene into a Python representation that's easy to manipulate, save to disk, and re-apply to a Maya scene. It has lots of support for manipulating Ziva creatures and can be easily extended. It's designed to be used by Technical Directors who are rigging characters with ZivaVFX. It supports use cases such as updating a rig when new geometry is published and transferring a Ziva rig from one character to another.

zUI is a beta UI to help navigate the Ziva setup in Maya. Among other things, it can help list objects in the simulation, find their attachments, quickly navigate and select Ziva DG nodes, and drop into painting maps.

CHAPTER 1

Introduction

Welcome to Ziva's VFX Utilities, a set of python utilities that have been designed to help leverage the benefits of the Ziva Character Simulation paradigm. You can find the BitBucket repo for these tools here.

Ziva Dynamics

zBuilder is a tool for extracting part of a Maya scene into a Python representation that's easy to manipulate, save to disk, and re-apply to a Maya scene. It has lots of support for manipulating Ziva creatures and can be easily extended. It's designed to be used by Technical Directors who are rigging characters with ZivaVFX. It supports use cases such as updating a rig when new geometry is published and transferring a Ziva rig from one character to another.

zUI is a beta UI to help navigate the Ziva setup in Maya. Among other things, it can help list objects in the simulation, find their attachments, quickly navigate and select Ziva DG nodes, and drop into painting maps.

CHAPTER 2

Installation

Either download a zip of the repository

https://bitbucket.org/zivadynamics/ziva-vfx-utils/downloads

or clone it

https://bitbucket.org/zivadynamics/ziva-vfx-utils.git

Put the zBuilder folder and all of its contents in a Maya scripts directory.

If you need to update your Python path for Maya you can place this in your userSetup.py:

```
#!python
import sys
sys.path.append('/path/to/download')
```

CHAPTER 3

zBuilder

At a high level, zBuilder is a python framework to help serialize and deserialize content from Maya scenes. The basic idea is that you first interactively build a scene in Maya, and then with zBuilder you can save that state and save it out and re-build it.

If we want to save a Ziva rig, for example, zBuilder can write out a json file representing the Ziva nodes in the scene. That file would contain the nodes and attribute values and some basic information about their relationships. zBuilder then gives you the ability to load that file into a scene with new geometry and re-build the rig there. If the new geometry's topology (ie: triangles, number of vertices) is different from the original rig, zBuilder allows you to interpolate data (eg: maps) to work on the new geometry.

3.1 Tutorials

3.1.1 Tutorial – Basics

Let's explore some example usage of zBuilder by trying it out on the anatomical arm demo that ships with the ZivaVFX Maya plugin. First, set the Python path to zBuilder as explained in the *Installation* section.

First, run the anatomical arm demo, which can be found in the 'Ziva Tools' menu: Ziva Tool > Examples > Run Demo > Anatomical Arm. Now that you have the arm *setup* in your scene, let's start playing with zBuilder.

Retrieving the Ziva rig from the Maya scene

Retrieving a whole Ziva rig

In order to interact with a Ziva scene with zBuilder, we first need to create a Ziva *builder* object:

```
import zBuilder.builders.ziva as zva
z = zva.Ziva()
```

You can use the help command to get some information about the Ziva builder.

```
help(z)
```

We can use our builder to capture the Ziva arm rig by running a method to retrieve the current state of the Ziva Maya scene:

```
z.retrieve_from_scene()
```

When you run this command, you should see output that looks something like this in your script editor:

```
# zBuilder.bundle : zTissue 7 #
# zBuilder.bundle : map 68 #
# zBuilder.bundle : zAttachment 21 #
# zBuilder.bundle : zMaterial 7 #
# zBuilder.bundle : zEmbedder 1 #
# zBuilder.bundle : zBone 4 #
# zBuilder.bundle : zTet 7 #
# zBuilder.bundle : mesh 11 #
# zBuilder.bundle : zSolver 1 #
# zBuilder.bundle : zSolverTransform 1 #
# zBuilder.bundle : zFiber 6 #
# zBuilder.builder : Finished: ---Elapsed Time = 0:00:00.087000 #
```

These are the stats about which *scene items* were retrieved by the builder. In this case you can see there are 7 zTissues, 4 zBones, etc. Scene items typically fall into two categories:

- *Nodes*, which are the Maya dependency graph nodes in the scene.
- *Parameters*, which are the relevant pieces of data associated with nodes, like meshes and *maps*.

To give you a sense of the complexity that zBuilder is handling for you here, the scene items captured in this case are:

- All the Ziva nodes. (zTissue, zTet, zAttachment, etc..)
- Order of the nodes so we can re-create material layers reliably.
- Attributes and values of the nodes. (Including weight maps)
- Sub-tissue information.
- User defined tet mesh reference. (Not the actual mesh)
- Any embedded mesh reference. (Not the actual mesh)
- Curve reference to drive zLineOfAction. (Not actual curve)
- Relevant zSolver for each node.
- Mesh information used for world space lookup to interpolate maps if needed.

Fortunately, zBuilder handles all this data for you, allowing you to treat all the complexity of a Ziva *rig* as a single logical object. You can then save it out to a text file, and/or restore the rig to the captured state at a later time. You can also manipulate the information in the builder before re-applying it. This is useful for mirroring, for example, which we'll describe later.

Retrieving parts of a Ziva rig

Above, we retrieved Ziva builder data from the entire Maya scene. However, if you only want to capture a part of the scene, you can select the items you are interested in and call retrieve_from_scene_selection(). This comes in handy if you want to mirror the setup, for example.

```
import maya.cmds as mc
mc.select('r_bicep_muscle')
import zBuilder.builders.ziva as zva
z = zva.Ziva()
z.retrieve_from_scene_selection()
```

By default retrieve_from_scene_selection() grabs all items that are connected to the selected items. In this example, therefore, it grabs the fibers and attachments connected to the muscle in addition to the muscle itself. Your script editor output should have looked something like this:

```
# zBuilder.bundle : zTissue 1 #
# zBuilder.bundle : map 12 #
# zBuilder.bundle : zAttachment 4 #
# zBuilder.bundle : zMaterial 1 #
# zBuilder.bundle : zEmbedder 1 #
# zBuilder.bundle : zBone 3 #
# zBuilder.bundle : zTet 1 #
# zBuilder.bundle : mesh 5 #
# zBuilder.bundle : zSolver 1 #
# zBuilder.bundle : zSolverTransform 1 #
# zBuilder.bundle : zFiber 1 #
# zBuilder.builder : Finished: ---Elapsed Time = 0:00:00.166000 #
```

Notice now we are only retrieving 1 tissue.

Building

Building takes the data stored in a builder object, and applies it to the Maya scene, equipping it with the Ziva rig stored in the builder object.

Note: zBuilder does not currently re-create geometry. The expectation is that any geometry required by the rig will already exist in the scene, and the builder will then apply the rig onto it. It's fine if the geometry is already being used in a Ziva rig, just as long as the geometry is already in scene.

With the exception of geometry, building restores the state of all the nodes and parameters in the builder. Each scene item is first checked to see if it exists in the Maya scene. If it doesn't exist, it is created. If it does exist, its data values are set to what is stored in the builder.

Restoring a Ziva rig to a previous state

This simple example demonstrates how to revert the Ziva rig to a previous state. First, load the Anatomical Arm Demo. Then, let's capture the whole scene, so that we can later restore it.

```
import zBuilder.builders.ziva as zva
z = zva.Ziva()
z.retrieve_from_scene()
```

Now, the builder object "z" contains the Ziva rig. Let's make a change to the arm. For example, paint a muscle attachment to all white, something that is easy to identify in viewport. Now let's apply our builder to it, to revert the rig to the previous state.

```
z.build()
```

In the viewport, you should see that the state of the arm rig jumped back to the way it was when you retrieved it, as well as this output in the script editor:

```
# zBuilder.builders.ziva : Building.... #
# zBuilder.builder : Finished: ---Elapsed Time = 0:00:01.139000 #
```

Building a Ziva rig from scratch

It is also possible to build a Ziva rig into a Maya scene that doesn't contain any Ziva nodes or data. The command is exactly the same as before, but we'll start from a "clean" scene containing only geometry.

First, clean out the entire Ziva rig with the following command:

```
import zBuilder.zMaya as mz
mz.clean_scene()
```

clean_scene() is a utility function to remove all of the Ziva footprint in the scene. If you look in the scene the Ziva solver nodes should now be gone.

Now that we have a scene with just geometry in it, let's see what happens when we apply that same builder.

```
z.build()
```

The full Ziva rig should now be restored and acting on the scene's geometry. zBuilder built all of the Ziva maya nodes for us.

Building with differing topologies

In production a common occurrence (unfortunately) is the geometry that goes into your rig will change and you will be the one who has to deal with it.

Let's show how zBuilder can accommodate changes to geometry.

First thing, let's clean the scene to represent brand new geometry coming in.

```
import zBuilder.zMaya as mz
mz.clean_scene()
```

Now change the bicep for example. A quick way is to apply a mesh smooth. Once the bicep has a different topology simply build the same way as before again.

```
z.build()
```

This time your script editor output will be slightly different. It should be as below:

```
# zBuilder.builders.ziva : Building.... #
# zBuilder.parameters.maps : interpolating map:  r_bicep_muscle_zTet.weightList[0].
↪weights #
# zBuilder.parameters.maps : interpolating map:  r_bicep_muscle_zMaterial.
↪weightList[0].weights #
# zBuilder.parameters.maps : interpolating map:  r_bicep_muscle_r_brachialis_muscle.
↪weightList[0].weights #
# zBuilder.parameters.maps : interpolating map:  r_bicep_muscle_r_brachialis_muscle.
↪weightList[1].weights #
# zBuilder.parameters.maps : interpolating map:  r_bicep_muscle_r_humerus_bone.
↪weightList[0].weights #
```

```
# zBuilder.parameters.maps : interpolating map:  r_bicep_muscle_r_humerus_bone.
↪weightList[1].weights #
# zBuilder.parameters.maps : interpolating map:  r_bicep_muscle_r_radius_bone.
↪weightList[0].weights #
# zBuilder.parameters.maps : interpolating map:  r_bicep_muscle_r_radius_bone.
↪weightList[1].weights #
# zBuilder.parameters.maps : interpolating map:  r_bicep_muscle_r_scapula_bone.
↪weightList[0].weights #
# zBuilder.parameters.maps : interpolating map:  r_bicep_muscle_r_scapula_bone.
↪weightList[1].weights #
# zBuilder.parameters.maps : interpolating map:  r_bicep_muscle_zFiber.weightList[0].
↪weights #
# zBuilder.parameters.maps : interpolating map:  r_bicep_muscle_zFiber.endPoints #
# zBuilder.builder : Finished: ---Elapsed Time = 0:00:03.585000 #
```

You will notice above that it listed out a bunch of maps that got interpolated. This shows that zBuilder noticed the change in topology between the mesh in the original rig and the new rig. Furthermore, the call to build() modified all the maps painted onto the old mesh and re-applied them to the new mesh by interpolation.

Note: When the maps get interpolated it is currently done in world space of the stored geometry. So, if a muscle's new geometry is in a significantly different position in world space, the interpolation may not work very well. However, it should be fine in cases where the position and shape of the muscle only make relatively small changes.

With this feature, you can manage bringing in any new geometry and building a previously-captured Ziva scene on it. Typically you will import the desired geometry into a scene from an external source instead of editing it directly in Maya (also ensure that it's given the same name as the original mesh it's replacing in the rig).

Reading/Writing Files

Writing to disk

Once we have the arm Ziva rig saved into a builder object in memory, we can write it out to disk. All we need to do is:

```
# replace path with a working temp directory on your system
z.write('C:\\Temp\\test.ziva')
```

This writes out a json file of all the information so it can be retrieved later.

Reading from disk

To test that writing worked properly let's setup the scene with just the geometry again. Run the Anatomical Arm demo again, then run mz.clean_scene().

Once we have a scene with just the arm geometry, let's retrieve the Ziva rig from the file on disk.

```
import zBuilder.builders.ziva as zva
z = zva.Ziva()
# Use the same path here that you used above.
z.retrieve_from_file('C:\\Temp\\test.zBuilder')
```

You should see something like this in your script editor:

```
z.retrieve_from_file('C:\\Temp\\test.zBuilder')
# zBuilder.builder : reading parameters. 134 nodes #
# zBuilder.builder : reading info #
# zBuilder.bundle : zTissue 7 #
# zBuilder.bundle : map 68 #
# zBuilder.bundle : zAttachment 21 #
# zBuilder.bundle : zMaterial 7 #
# zBuilder.bundle : zEmbedder 1 #
# zBuilder.bundle : zBone 4 #
# zBuilder.bundle : zTet 7 #
# zBuilder.bundle : mesh 11 #
# zBuilder.bundle : zSolver 1 #
# zBuilder.bundle : zSolverTransform 1 #
# zBuilder.bundle : zFiber 6 #
# zBuilder.builder : Read File: C:\Temp\test.zBuilder in 0:00:00.052000 #
```

Like before, this is a simple printout to give you a hint of what has been loaded from the file. Now we can build:

```
z.build()
```

If you have been following along the output should look like this again as there would have been no map interpolation.

```
# zBuilder.builders.zBuilder : Building.... #
# zBuilder.builder : Finished: ---Elapsed Time = 0:00:03.578000 #
```

The Anatomical Arm rig should now be completely restored back to its original state.

String Replacing

You can do basic string replace operations on the information stored in a builder. This is very useful if you have name changes of the geometry you are dealing with, or even to create a basic mirroring of the rig.

When you do a string replace you provide a search term and a replace term. In the context of the Ziva builder it will search and replace:

- node names
- map names (zAttachment1.weights for example)
- curve names for zLineOfAction
- any mesh name (embedded, user tet)

This works with regular expressions as well. For example you can search for occurrences of "r_" at the beginning of a name.

Changing geometry name

As before, let's build the Anatomical Arm demo from the Ziva menu and retrieve the Ziva rig into a builder object.

```
import zBuilder.builders.ziva as zva
z = zva.Ziva()
z.retrieve_from_scene()
```

To represent a model name change let's clean the scene and change the name of one of the muscles.

```
import zBuilder.zMaya as mz
mz.clean_scene()

mc.rename('r_bicep_muscle', 'r_biceps_muscle')
```

Now the information in the builder is out of sync with the geometry in the scene. We can update it by doing the following:

```
z.string_replace('r_bicep_muscle','r_biceps_muscle')
```

Now when we build you see that the newly-named muscle is correctly integrated into the rig, and all the maps painted on that mesh have had their names corrected as well.

```
z.build()
```

Mirroring a setup

We can also use string replace to mirror half of a Ziva rig into a full symmetric Ziva rig.

In order for this to work the geometry needs to be already-mirrored, with r_* and l_* prefixes used to distinguish between each pair of mirrored meshes. Assuming you have already created a rig on the right-side of the character, you will then tell the builder to replace r_muscle with l_muscle (note that all zBuilder will be doing here is changing names, so it expects all of the l_muscle meshes to already be in the scene).

Let's run a little test scene that sets up 2 spheres and a cube with 1 attachment.

```
import zBuilder.tests.utils as utl

utl.build_mirror_sample_geo()
utl.ziva_mirror_sample_geo()
```

You should see a cube and 2 spheres in your scene. The right-side sphere "r_muscle" is a tissue and the cube is a bone, and they are connected by a single attachment. We want to mirror this so the "l_muscle" gets a tissue and attachment as well. To do this we can just create and initialize a builder, perform a string replace, and then rebuild.

```
import zBuilder.builders.ziva as zva

z = zva.Ziva()
z.retrieve_from_scene()
z.string_replace('^r_','l_')
```

Notice the ^ in the search field. This is a regular expression to tell it to search just for an "r_" at the beginning of a name.

Now when you build you should have a mirrored setup:

```
z.build()
```

3.1.2 Tutorial – Advanced

Here we'll cover some of the more involved concepts.

Changing values before building

It's possible to inspect and modify the contents of the builder before you actually build. For example, maybe you are in a specific shot and want to build a Ziva rig with a value different than what was saved on disk. A common use case is to change the start frame of the Ziva solver based on the shot environment. Let's try doing that.

Build the Anatomical Arm demo again and retrieve the scene.

```
import zBuilder.builders.ziva as zva

z = zva.Ziva()
z.retrieve_from_scene()
```

Now we need to find the scene item we want to modify, in this case the solver. You can do that with the following code:

```
scene_items = z.get_scene_items(name_filter='zSolver1')
print scene_items[0]
```

Here we're using the name filter to search for the specific item we're interested in. You should see something like this in the script editor.

```
= zSolver1 <zBuilder.nodes.ziva.zSolverTransform SolverTransformNode>
↪==================================
    _builder_type - zBuilder.nodes
    solver - zSolver1Shape
    _DGNode__mobject - <maya.OpenMaya.MObject; proxy of <Swig Object of type 'MObject
↪*' at 0x000001E90F8E9690> >
    _name - |zSolver1
    _association - []
    attrs - {u'enable': {'locked': False, 'type': u'bool', 'value': True, 'alias':
↪None}, u'translateX': {'locked': False, 'type': u'doubleLinear', 'value': 0.0,
↪'alias': None}, u'translateY': {'locked': False, 'type': u'doubleLinear', 'value':
↪0.0, 'alias': None}, u'translateZ': {'locked': False, 'type': u'doubleLinear',
↪'value': 0.0, 'alias': None}, u'scaleX': {'locked': False, 'type': u'double', 'value
↪': 100.0, 'alias': None}, u'scaleY': {'locked': False, 'type': u'double', 'value':
↪100.0, 'alias': None}, u'visibility': {'locked': False, 'type': u'bool', 'value':
↪True, 'alias': None}, u'rotateX': {'locked': False, 'type': u'doubleAngle', 'value
↪': 0.0, 'alias': None}, u'rotateY': {'locked': False, 'type': u'doubleAngle', 'value
↪': 0.0, 'alias': None}, u'rotateZ': {'locked': False, 'type': u'doubleAngle', 'value
↪': 0.0, 'alias': None}, u'scaleZ': {'locked': False, 'type': u'double', 'value':
↪100.0, 'alias': None}, u'startFrame': {'locked': False, 'type': u'double', 'value':
↪1.0, 'alias': None}}
    _class - ('zBuilder.nodes.ziva.zSolverTransform', 'SolverTransformNode')
    type - zSolverTransform
    builder - <zBuilder.builders.ziva.Ziva object at 0x000001E90FDB97B8>
```

That's all the information that the builder has stored for the solver scene item. To query and change the attributes you go through the `attrs` dictionary like so:

```
print 'Before:', scene_item[0].attrs['startFrame']['value']
# set the value of startFrame to 10
scene_item[0].attrs['startFrame']['value'] = 10
print 'After:', scene_item[0].attrs['startFrame']['value']
```

In the above example we're printing the value of start frame before and after we change it.

Now if you apply the builder, the startFrame of the zSolver1 node will be given the new value you set. As before, the new value is applied whether or not the zSolver1 node already existed in the scene before the call to build().

```
z.build()
```

3.2 API Reference

class zBuilder.builders.ziva.**Ziva**
To capture a Ziva rig.

retrieve_from_scene(*get_parameters*)
This gets the scene items from the scene for further manipulation or saving. It works on selection or something passed in args. If nothing is selected it looks for a zSolver in the scene. If something is selected or passed it uses that specific solver to retrieve.

Items captured in this case are:

- All the Ziva nodes. (zTissue, zTet, zAttachment, etc..)
- Order of the nodes so we can re-create material layers reliably.
- Attributes and values of the nodes. (Including weight maps)
- Sub-tissue information.
- User defined tet mesh reference. (Not the actual mesh)
- Any embedded mesh reference. (Not the actual mesh)
- Curve reference to drive zLineOfAction. (Not actual curve)
- Relevant zSolver for each node.
- Mesh information used for world space lookup to interpolate maps if needed.

Parameters **get_parameters** (*bool*) – To get parameters or not.

build(*name_filter=None*, *attr_filter=None*, *interp_maps='auto'*, *mirror=False*, *permissive=True*, *check_meshes=True*)
This builds the Ziva rig into the Maya scene. It does not build geometry as the expectation is that the geometry is in the scene.

Parameters

- **solver** (*bool*) – Build the solver.
- **bones** (*bool*) – Build the bones.
- **tissues** (*bool*) – Build the tissue and tets.
- **attachments** (*bool*) – Build the attachments.
- **materials** (*bool*) – Build the materials.
- **fibers** (*bool*) – Build the fibers.
- **embedder** (*bool*) – Build the embedder.
- **cloth** (*bool*) – Build the cloth.
- **lineOfActions** (*bool*) – Build the line of actions.
- **interp_maps** (*str*) – Option to interpolate maps. True: Yes interpolate False: No auto: Interpolate if it needs it (vert check)
- **mirror** (*bool*) – This mirrors the geometry in bundle.

- **permissive** (*bool*) – False raises errors if something is wrong. Defaults to True
- **attr_filter** (*dict*) – Attribute filter on what attributes to get. dictionary is key value where key is node type and value is list of attributes to use.

  tmp = {'zSolver':['substeps']}
- **association_filter** (*str*) – filter by node association. Defaults to list()

class zBuilder.builders.ziva.**Builder**

The main entry point for using zBuilder.

write(*file_path*, *type_filter=list()*, *invert_match=False*)

Writes out the scene items to a json file given a file path.

Parameters

- **file_path** (*str*) – The file path to write to disk.
- **type_filter** (*list, optional*) – Types of scene items to write.
- **invert_match** (*bool*) – Invert the sense of matching, to select non-matching items. Defaults to False

retrieve_from_file(*file_path*)

Reads scene items from a given file. The items get placed in the bundle.

Parameters **file_path** (str) – The file path to read from disk.

string_replace(*search*, *replace*)

Searches and replaces with regular expressions scene items in the builder.

Parameters

- **search** (str) – what to search for
- **replace** (str) – what to replace it with

Example

replace *r_* at front of item with *l_*:

```
>>> z.string_replace('^r_','l_')
```

replace *_r* at end of line with *_l*:

```
>>> z.string_replace('_r$','_l')
```

stats()

Prints out basic information in Maya script editor. Information is scene item types and counts.

print_(*type_filter=[]*, *name_filter=[]*)

Prints out basic information for each scene item in the Builder. Information is all information that is stored in the __dict__. Useful for trouble shooting.

Parameters

- **type_filter** (list or str) – filter by parameter type. Defaults to list
- **name_filter** (list or str) – filter by parameter name. Defaults to list

get_scene_items(*type_filter=[]*, *name_filter=[]*, *name_regex=None*, *association_filter=[]*, *association_regex=None*, *invert_match=False*)

Gets the scene items from builder for further inspection or modification.

Parameters

- **type_filter** (str or list, optional) – filter by parameter type. Defaults to list.
- **name_filter** (str or list, optional) – filter by parameter name. Defaults to list.
- **name_regex** (str) – filter by parameter name by regular expression. Defaults to None.
- **association_filter** (str or list, optional) – filter by parameter association. Defaults to list.
- **association_regex** (str) – filter by parameter association by regular expression. Defaults to None.
- **invert_match** (*bool*) – Invert the sense of matching, to select non-matching items. Defaults to False

Returns List of scene items.

Return type list

3.3 Extending zBuilder

zBuilder is designed to be easily extended by adding new customized builders that handle specialized sets of nodes and parameters. It's also possible to add the ability to handle new types of nodes and parameters to the framework.

More to come....

3.4 Whats changed in 1.0.0

zBuilder 1.0.0 is backwards compatible with previous versions but some of command names have changed. For the most part it will be unnoticeable but a few things have changed in how zBuilder is accessed. For instance:

```
# pre 1.0.0 instantiating object
import zBuilder.setup.Ziva as zva
z = zva.ZivaSetup()

# 1.0.0
import zBuilder.builders.ziva as zva
z = zva.Ziva()
```

Building has changed as well. Previously that was called 'apply'

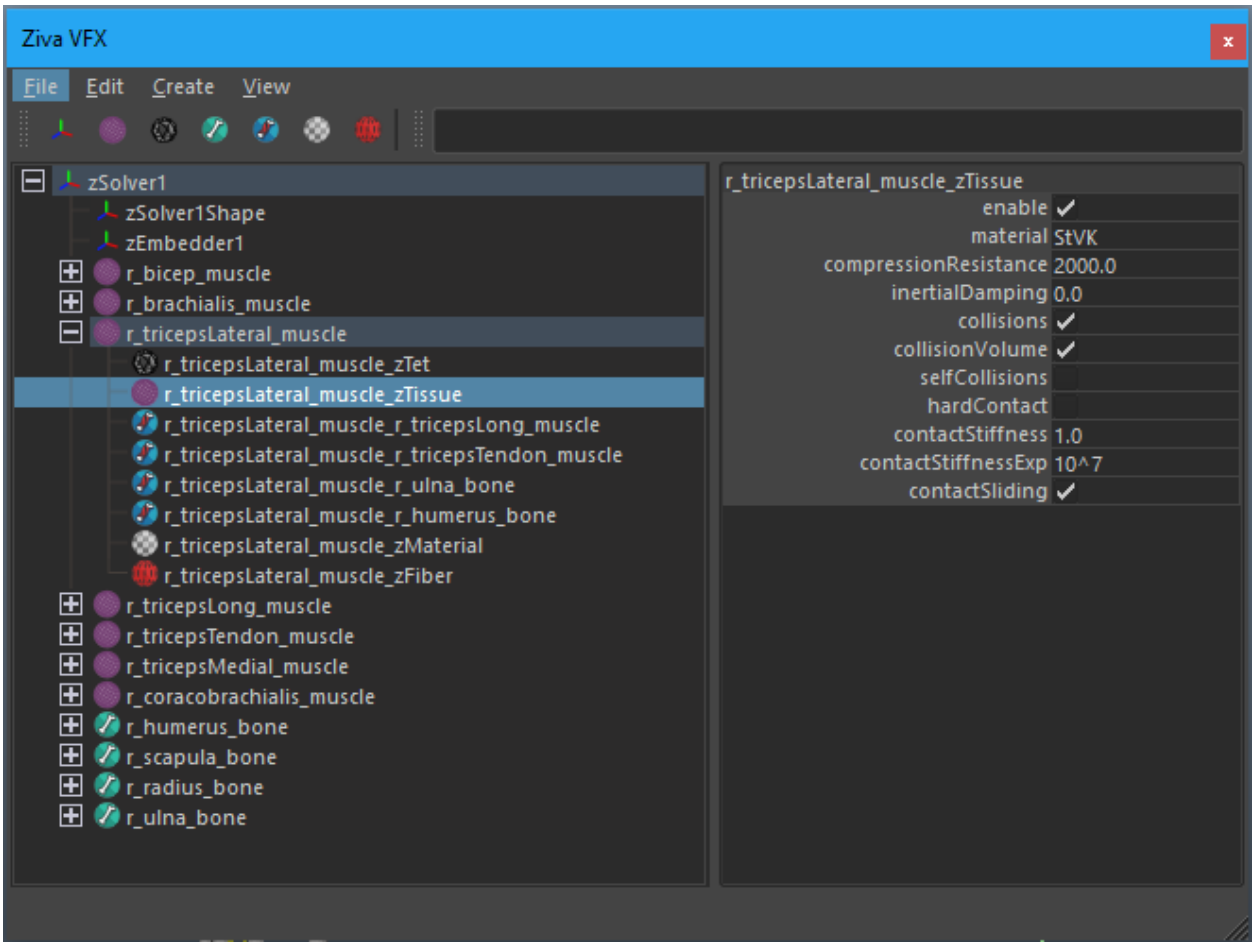
```
# pre 1.0.0 instantiating object
import zBuilder.setup.Ziva as zva
z = zva.ZivaSetup()
z.apply()

# 1.0.0
import zBuilder.builders.ziva as zva
z = zva.Ziva()
z.build()
```

At the higher level just remember "setups" are now "builders" and "apply" is now "build".

CHAPTER 4

zUI

A *beta* graphical UI to help navigate the Ziva setup in Maya.



To run in Maya:

```
import zUI.ui as ui
myWin = ui.ZivaUi()
myWin.run()
```

CHAPTER 5

Release Notes

5.1 1.0.4

- QT tree view for builder data
- bug fixes

5.2 1.0.3

- zUI support on maya 2017 and 2018
- bug fixes

5.3 1.0.0

- major refactor
- file backwards compatibility
- support for multiple solvers
- easier to extend

5.4 0.11.3

- zBuilder support for sub-tissues
- mirroring of geo before application (experimental)
- zLineOfAction functionality added to retrieve_from_scene_selecton

- general bug fixes

5.5 0.11.2

- Restructure of class hierarchy
- packages can extend themselves
- bug fixes

5.6 0.11.1

- Material, Fiber and Attachment creation now more robust. No longer name cascading problems.
- lineOfAction node added

5.7 0.11.0

- removed abstract methods from NodeCollection
- depractated set_attrs and set_weights in favor of using a MayaMixin class
- storing mObjects internally during node creation to get around maya renaming
- zMaya.rename_ziva_nodes() handles zBones and zCloth

5.8 0.10.0

- save out component data and node data seperatly
- changed map.py to maps.py
- fixed bug in cloth creation
- changed node_filter to name_filter. Better representation on what it is.

5.9 0.9.5

- changed order of cloth application when applying

5.10 0.9.4

- retrieving from scene in ZivaSetup now works by passing nodes or not. Default behavior is unchanged.
- restoring user selection when using zMapa.py methods.
- added support for cloth

CHAPTER 6

Glossary

body (simulation body) Bone, tissue or cloth objects.

build() The act of assembling a Maya scene from the *scene items* stored in a *builder* object.

builder The main entry point into using zBuilder. A builder object manages a bundle of *scene items*. There are different types of specialized builders, each one defining what types of *nodes* and *parameters* they allow you to inspect, modify and build in the Maya scene.

map A type of *parameter* consisting of per-vertex data on a mesh, typically created in Maya through the weight painting tool. Commonly used by deformers, as well as many Ziva nodes.

node A Maya dependency graph (DG) node. These are the fundamental building blocks that define the state of a Maya scene. Therefore they are the primary *scene items* that a *builder* retrieves from and rebuilds into a scene.

parameter A specific piece of data in the Maya scene, for example a mesh or an attribute of a node. These are the secondary type of *scene item* managed by a *builder*, and are always associated in some way with the *nodes* in the scene.

retrieve() Capturing the Maya scene with a *builder* and storing it as a set of *scene items* in the *builder* object.

rig The set of all components that go into creating an animatable character in Maya. A typical Ziva rig includes geometry defining the shape of the anatomy, and a suite of dependency graph nodes that define the physical characteristics of all the simulation *bodies*. Often used interchangeably with *setup*.

scene item *Nodes* and associated *parameters* that together define the current state of a scene in Maya. These are the items that a *builder* retrieves from the scene, allowing you to inspect them, modify them, and re-apply them to a scene at a later time.

setup Used as a synonym for *rig* in the context of creating characters in Maya with ZivaVFX.

zNode A Maya DG node specific to Ziva. Examples include zTet, zTissue, zAttachment, etc.

zUi A Maya UI to help you navigate scenes containing Ziva rigs. Currently in Beta.

CHAPTER 7

Module index

7.1 Subpackages

7.1.1 zBuilder.builders package

Submodules

zBuilder.builders.attributes module

class zBuilder.builders.attributes.**Attributes**
Bases: *zBuilder.builder.Builder*

Storing maya attributes

build(**args*, ***kwargs*)

retrieve_from_scene(**args*, ***kwargs*)

zBuilder.builders.constraints module

class zBuilder.builders.constraints.**Constraints**
Bases: *zBuilder.builder.Builder*

To capture Maya constraints. Supports point, orient and parent constraints.

build(**args*, ***kwargs*)

retrieve_from_scene(**args*, ***kwargs*)

zBuilder.builders.deltaMush module

class zBuilder.builders.deltaMush.**DeltaMush**
Bases: *zBuilder.builder.Builder*

To capture a deltaMush

build(**args*, ***kwargs*)

retrieve_from_scene(**args*, ***kwargs*)

zBuilder.builders.selection module

class zBuilder.builders.selection.**Selection**
Bases: *zBuilder.builder.Builder*

Storing maya selection.

build(**args*, ***kwargs*)

retrieve_from_scene(**args*, ***kwargs*)

zBuilder.builders.skinClusters module

class zBuilder.builders.skinClusters.**SkinCluster**
Bases: *zBuilder.builder.Builder*

Capturing Maya skinClusters

build(**args*, ***kwargs*)

retrieve_from_scene(**args*, ***kwargs*)

zBuilder.builders.ziva module

class zBuilder.builders.ziva.**Ziva**
Bases: *zBuilder.builder.Builder*

To capture a Ziva rig.

build(**args*, ***kwargs*)
This builds the Ziva rig into the Maya scene. It does not build geometry as the expectation is that the geometry is in the scene.

Parameters

- **solver** (*bool*) – Build the solver.
- **bones** (*bool*) – Build the bones.
- **tissues** (*bool*) – Build the tissue and tets.
- **attachments** (*bool*) – Build the attachments.
- **materials** (*bool*) – Build the materials.
- **fibers** (*bool*) – Build the fibers.
- **embedder** (*bool*) – Build the embedder.
- **cloth** (*bool*) – Build the cloth.
- **lineOfActions** (*bool*) – Build the line of actions.
- **interp_maps** (*str*) – Option to interpolate maps. True: Yes interpolate False: No auto: Interpolate if it needs it (vert check)
- **mirror** (*bool*) – This mirrors the geometry in bundle.

- **permissive** (*bool*) – False raises errors if something is wrong. Defaults to True
- **attr_filter** (*dict*) – Attribute filter on what attributes to get. dictionary is key value where key is node type and value is list of attributes to use.

  tmp = {'zSolver':['substeps']}
- **association_filter** (*str*) – filter by node association. Defaults to list()

get_parent()

reset_solvers()
This resets the solvers stored in the zBuilder. Specifically, it removes any stored MObjects from the solvers.

retrieve_from_scene(**args*, ***kwargs*)
This gets the scene items from the scene for further manipulation or saving. It works on selection or something passed in args. If nothing is selected it looks for a zSolver in the scene. If something is selected or passed it uses that specific solver to retrieve.

Items captured in this case are:

- All the Ziva nodes. (zTissue, zTet, zAttachment, etc..)
- Order of the nodes so we can re-create material layers reliably.
- Attributes and values of the nodes. (Including weight maps)
- Sub-tissue information.
- User defined tet mesh reference. (Not the actual mesh)
- Any embedded mesh reference. (Not the actual mesh)
- Curve reference to drive zLineOfAction. (Not actual curve)
- Relevant zSolver for each node.
- Mesh information used for world space lookup to interpolate maps if needed.

Parameters **get_parameters** (*bool*) – To get parameters or not.

retrieve_from_scene_selection(**args*, ***kwargs*)
Gets scene items based on selection.

Parameters

- **attr_filter** (*dict*) – Attribute filter on what attributes to get. dictionary is key value where key is node type and value is list of attributes to use.

  af = {'zSolver':['substeps']}
- **connections** (*bool*) – Gets the ziva nodes connected to selection as well. Defaults to True
- **solver** (*bool*) – Gets solver data. Defaults to True
- **bones** (*bool*) – Gets bone data. Defaults to True
- **tissue** (*bool*) – Gets tissue data. Defaults to True
- **attachments** (*bool*) – Gets attachments data. Defaults to True
- **materials** (*bool*) – Gets materials data. Defaults to True
- **fibers** (*bool*) – Gets fibers data. Defaults to True
- **cloth** (*bool*) – Gets cloth data. Defaults to True

- **lineOfAction** (*bool*) – Gets line of action data. Defaults to True
- **embedder** (*bool*) – Gets embedder data. Defaults to True
- **get_parameters** (*bool*) – get mesh info. Defaults to True

Module contents

7.1.2 zBuilder.nodes package

Subpackages

zBuilder.nodes.deformers package

Submodules

zBuilder.nodes.deformers.blendShape module

class zBuilder.nodes.deformers.blendShape.**BlendShape**(**args*, ***kwargs*)

Bases: *zBuilder.nodes.deformer.Deformer*

EXTEND_ATTR_LIST = ['origin']

MAP_LIST = ['inputTarget[*].inputTargetGroup[*].targetWeights', 'inputTarget[*].baseWe

build(**args*, ***kwargs*)

get_map_meshes()

This is the mesh associated with each map in obj.MAP_LIST. Typically it seems to coincide with mesh store in get_association. Sometimes it deviates, so you can override this method to define your own list of meshes against the map list.

For blendShapes we don't know how many maps so we are generating this list based on length of maps.

Returns of long mesh names.

Return type list()

long_target

populate(*maya_node=None*)

target

type = 'blendShape'

zBuilder.nodes.deformers.blendShape.**get_target**(*blend_shape*)

zBuilder.nodes.deformers.deltaMush module

class zBuilder.nodes.deformers.deltaMush.**DeltaMush**(*parent=None*, *maya_node=None*, *builder=None*, *deserialize=None*)

Bases: *zBuilder.nodes.deformer.Deformer*

MAP_LIST = ['weightList[0].weights']

build(**args*, ***kwargs*)

type = 'deltaMush'

zBuilder.nodes.deformers.skinCluster module

class zBuilder.nodes.deformers.skinCluster.**SkinCluster**(**args*, ***kwargs*)

Bases: *zBuilder.nodes.dg_node.DGNode*

The base node for the node functionality of all nodes

EXTEND_ATTR_LIST = []

List of maya attributes to add to attribute list when capturing.

SEARCH_EXCLUDE = ['_class', 'attrs', '_builder_type', 'type']

List of attributes to exclude with a string_replace

TYPES = []

The type of node.

build(**args*, ***kwargs*)

populate(*maya_node=None*)

This extends ZivaBase.populate()

Adds parent and child storage.

Parameters **maya_node** – Maya node to populate with.

type = 'skinCluster'

zBuilder.nodes.deformers.skinCluster.**apply_weights**(*skin_cluster*, *mesh*, *influences*, *weights*)

zBuilder.nodes.deformers.skinCluster.**get_associations**(*skin_cluster*)

zBuilder.nodes.deformers.skinCluster.**get_influences**(*skin_cluster*)

zBuilder.nodes.deformers.skinCluster.**get_weights**(*skin_cluster*)

zBuilder.nodes.deformers.wrap module

class zBuilder.nodes.deformers.wrap.**Wrap**(*parent=None*, *maya_node=None*, *builder=None*, *deserialize=None*)

Bases: *zBuilder.nodes.deformer.Deformer*

build(**args*, ***kwargs*)

static get_meshes(*node*)

Queries the deformer and returns the meshes associated with it.

Parameters **node** – Maya node to query.

Returns list of strings of mesh names.

Return type list od strings

type = 'wrap'

Module contents

zBuilder.nodes.utils package

Submodules

zBuilder.nodes.utils.constraint module

class zBuilder.nodes.utils.constraint.**Constraint**(**args*, ***kwargs*)
Bases: *zBuilder.nodes.dg_node.DGNode*

The base node for the node functionality of all nodes

EXTEND_ATTR_LIST = []
List of maya attributes to add to attribute list when capturing.

SEARCH_EXCLUDE = ['_class', '_attrs']
List of attributes to exclude with a string_replace

TYPES = ['pointConstraint', 'orientConstraint', 'parentConstraint']
The type of node.

build(**args*, ***kwargs*)
Builds the zCloth in maya scene.

Parameters

- **attr_filter** (*dict*) – Attribute filter on what attributes to get. dictionary is key value where key is node type and value is list of attributes to use.

  tmp = {'zSolver':['substeps']}

- **permissive** (*bool*) – Pass on errors. Defaults to `True`

constrained

populate(*maya_node=None*)

targets

type = None

zBuilder.nodes.utils.constraint.**get_constrained**(*constraint_name*)

zBuilder.nodes.utils.constraint.**get_targets**(*constraint_name*)

Module contents

zBuilder.nodes.ziva package

Submodules

zBuilder.nodes.ziva.zAttachment module

class zBuilder.nodes.ziva.zAttachment.**AttachmentNode**(**args*, ***kwargs*)
Bases: *zBuilder.nodes.ziva.zivaBase.Ziva*

This node for storing information related to zAttachments.

MAP_LIST = ['weightList[0].weights', 'weightList[1].weights']
List of maps to store.

build(**args*, ***kwargs*)
Builds the zAttachment in maya scene.

Parameters

- **attr_filter** (*dict*) – Attribute filter on what attributes to get. dictionary is key value where key is node type and value is list of attributes to use.

  tmp = {'zSolver':['substeps']}
- **interp_maps** (*str*) – Interpolating maps. Defaults to `auto`
- **permissive** (*bool*) – Pass on errors. Defaults to `True`

type = 'zAttachment'
The type of node.

zBuilder.nodes.ziva.zBone module

class zBuilder.nodes.ziva.zBone.**BoneNode**(**args*, ***kwargs*)
Bases: *zBuilder.nodes.ziva.zivaBase.Ziva*

This node for storing information related to zBones.

build(**args*, ***kwargs*)
Builds the zBones in maya scene.

Parameters

- **attr_filter** (*dict*) – Attribute filter on what attributes to get. dictionary is key value where key is node type and value is list of attributes to use.

  tmp = {'zSolver':['substeps']}
- **permissive** (*bool*) – Pass on errors. Defaults to `True`

type = 'zBone'
The type of node.

zBuilder.nodes.ziva.zBone.**apply_multiple**(*parameters*, *attr_filter=None*, *permissive=False*)
Each node can deal with it's own building. Though, with zBones it is much faster to build them all at once with one command instead of looping through them. This function builds all the zBones at once.

Parameters

- **permissive** (*bool*) –
- **parameters** –
- **attr_filter** (*obj*) –

Returns:

zBuilder.nodes.ziva.zCloth module

class zBuilder.nodes.ziva.zCloth.**ClothNode**(**args*, ***kwargs*)
Bases: *zBuilder.nodes.ziva.zivaBase.Ziva*

This node for storing information related to zCloth.

build(**args*, ***kwargs*)
Builds the zCloth in maya scene.

Parameters

- **attr_filter** (*dict*) – Attribute filter on what attributes to get. dictionary is key value where key is node type and value is list of attributes to use.

  tmp = {'zSolver':['substeps']}

- **permissive** (*bool*) – Pass on errors. Defaults to `True`

type = 'zCloth'
The type of node.

zBuilder.nodes.ziva.zEmbedder module

class zBuilder.nodes.ziva.zEmbedder.**EmbedderNode**(**args*, ***kwargs*)
Bases: *zBuilder.nodes.ziva.zivaBase.Ziva*

This node for storing information related to zEmebedder.

build(**args*, ***kwargs*)
Builds the zEmbedder in maya scene.

Parameters

- **attr_filter** (*dict*) – Attribute filter on what attributes to get. dictionary is key value where key is node type and value is list of attributes to use.

  tmp = {'zSolver':['substeps']}

- **permissive** (*bool*) – Pass on errors. Defaults to `True`

get_collision_meshes(*long_name=False*)
Gets the collision meshes stored. :param long_name: Returns long name or short. Default `False` :type long_name: bool

Returns String of collision mesh name.

Return type str

get_embedded_meshes(*long_name=False*)
Gets the embedded meshes stored. :param long_name: Returns long name or short. Default `False` :type long_name: bool

Returns String of embedded mesh name.

Return type str

populate(*maya_node=None*)
This populates the node given a selection.

Parameters **maya_node** – Maya node to populate with.

set_collision_meshes(*meshes*)
Sets the collision meshes.

Parameters **meshes** (*list*) – The meshes to set.

set_embedded_meshes(*meshes*)
Sets the embedded meshes.

Parameters **meshes** (*list*) – The meshes to set.

type = 'zEmbedder'
The type of node.

zBuilder.nodes.ziva.zEmbedder.**get_embedded_meshes**(*bodies*)
Returns embedded meshes for given body. :param bodies: Maya mesh to find embedded meshes with.

Returns of embedded meshes and collision meshes.

Return type 2 dict

zBuilder.nodes.ziva.zFiber module

class zBuilder.nodes.ziva.zFiber.**FiberNode**(**args*, ***kwargs*)
Bases: *zBuilder.nodes.ziva.zivaBase.Ziva*

This node for storing information related to zFibers.

MAP_LIST = ['weightList[0].weights', 'endPoints']
List of maps to store.

build(**args*, ***kwargs*)
Builds the zFiber in maya scene.

Parameters

- **attr_filter** (*dict*) – Attribute filter on what attributes to get. dictionary is key value where key is node type and value is list of attributes to use.

  tmp = {'zSolver':['substeps']}
- **interp_maps** (*str*) – Interpolating maps. Defaults to `auto`
- **permissive** (*bool*) – Pass on errors. Defaults to `True`

get_map_meshes()
This is the mesh associated with each map in obj.MAP_LIST. Typically it seems to coincide with mesh store in get_association. Sometimes it deviates, so you can override this method to define your own list of meshes against the map list.

Returns of long mesh names.

Return type list()

type = 'zFiber'
The type of node.

zBuilder.nodes.ziva.zLineOfAction module

class zBuilder.nodes.ziva.zLineOfAction.**LineOfActionNode**(**args*, ***kwargs*)
Bases: *zBuilder.nodes.ziva.zivaBase.Ziva*

This node for storing information related to zLineOfAction.

build(**args*, ***kwargs*)
Builds the Line of Actions in maya scene.

Parameters

- **attr_filter** (*dict*) – Attribute filter on what attributes to get. dictionary is key value where key is node type and value is list of attributes to use.

  tmp = {'zSolver':['substeps']}

- **permissive** (*bool*) – Pass on errors. Defaults to `True`

populate(*maya_node=None*)
This populates the node given a selection.

Parameters **maya_node** – Maya node to populate with.

spawn_parameters()

type = 'zLineOfAction'
The type of node.

zBuilder.nodes.ziva.zMaterial module

class zBuilder.nodes.ziva.zMaterial.**MaterialNode**(**args*, ***kwargs*)
Bases: *zBuilder.nodes.ziva.zivaBase.Ziva*

This node for storing information related to zMaterials.

MAP_LIST = ['weightList[0].weights']
List of maps to store.

build(**args*, ***kwargs*)
Builds the zMaterial in maya scene.

Parameters

- **attr_filter** (*dict*) – Attribute filter on what attributes to get. dictionary is key value where key is node type and value is list of attributes to use.

  tmp = {'zSolver':['substeps']}

- **interp_maps** (*str*) – Interpolating maps. Defaults to `auto`
- **permissive** (*bool*) – Pass on errors. Defaults to `True`

type = 'zMaterial'
The type of node.

zBuilder.nodes.ziva.zSolver module

class zBuilder.nodes.ziva.zSolver.**SolverNode**(**args*, ***kwargs*)
Bases: *zBuilder.nodes.ziva.zivaBase.Ziva*

This node is for storing information related to zSolver.

build(**args*, ***kwargs*)
Builds the zSolver in maya scene.

Parameters

- **attr_filter** (*dict*) – Attribute filter on what attributes to get. dictionary is key value where key is node type and value is list of attributes to use.

  tmp = {'zSolver':['substeps']}

- **permissive** (*bool*) – Pass on errors. Defaults to `True`

type = 'zSolver'
The type of node.

zBuilder.nodes.ziva.zSolverTransform module

class zBuilder.nodes.ziva.zSolverTransform.**SolverTransformNode**(**args*, ***kwargs*)

Bases: *zBuilder.nodes.ziva.zivaBase.Ziva*

This node for storing information related to zSolverTransform.

build(**args*, ***kwargs*)

Builds the zSolverTransform in maya scene.

Parameters

- **attr_filter** (*dict*) – Attribute filter on what attributes to get. dictionary is key value where key is node type and value is list of attributes to use.

  tmp = {'zSolver':['substeps']}

- **permissive** (*bool*) – Pass on errors. Defaults to `True`

type = 'zSolverTransform'

The type of node.

zBuilder.nodes.ziva.zTet module

class zBuilder.nodes.ziva.zTet.**TetNode**(**args*, ***kwargs*)

Bases: *zBuilder.nodes.ziva.zivaBase.Ziva*

This node for storing information related to zTets.

MAP_LIST = ['weightList[0].weights']

List of maps to store.

apply_user_tet_mesh()

Applies the user tet mesh if any.

build(**args*, ***kwargs*)

Builds the zTets in maya scene.

These get built after the tissues so it is assumed they are in scene. This just checks what tet is associated with mesh and uses that one, renames it and stores mObject then changes attributes. There is only ever 1 per mesh so no need to worry about multiple tets

Parameters

- **attr_filter** (*dict*) – Attribute filter on what attributes to get. dictionary is key value where key is node type and value is list of attributes to use.

  tmp = {'zSolver':['substeps']}

- **interp_maps** (*str*) – Interpolating maps. Defaults to `auto`
- **permissive** (*bool*) – Pass on errors. Defaults to `True`

get_user_tet_mesh(*long_name=False*)

Get user tet mesh. :param long_name: return long name or not. Defaults to `False`

Returns String of user tet mesh name.

Return type str

set_user_tet_mesh(*mesh*)

Setting of the user tet mesh. :param mesh: A maya mesh. :type mesh: str

type = 'zTet'
The type of node.

zBuilder.nodes.ziva.zTissue module

class zBuilder.nodes.ziva.zTissue.**TissueNode**(**args*, ***kwargs*)
Bases: *zBuilder.nodes.ziva.zivaBase.Ziva*

This node for storing information related to zTissues.

build(**args*, ***kwargs*)
Builds the zTissue in maya scene.

Kwargs:

attr_filter (dict): Attribute filter on what attributes to get. dictionary is key value where key is node type and value is list of attributes to use.

tmp = {'zSolver':['substeps']}

interp_maps (str): Interpolating maps. Defaults to `auto` permissive (bool): Pass on errors. Defaults to `True`

populate(*maya_node=None*)
This populates the node given a selection.

Parameters **maya_node** – Maya node to populate with.

type = 'zTissue'
The type of node.

zBuilder.nodes.ziva.zTissue.**build_multiple**(*tissue_items*, *tet_items*, *interp_maps='auto'*, *attr_filter=None*, *permissive=True*, *solver=None*)
Each node can deal with it's own building. Though, with zBones it is much faster to build them all at once with one command instead of looping through them. This function builds all the zBones at once.

Parameters

- **permissive** (*bool*) –
- **tissue_items** –
- **tet_items** –
- **attr_filter** (*obj*) –
- **solver** –
- **interp_maps** –

Returns:

zBuilder.nodes.ziva.zTissue.**get_tissue_children**(*ztissue*)
This checks a zTissue if it has children. Useful for sub-tissues. :param ztissue: The zTissue object in the maya scene. :type ztissue: str

Returns (str) Children mesh of zTissue, or None if none found.

zBuilder.nodes.ziva.zTissue.**get_tissue_parent**(*ztissue*)
This checks a zTissue if it has a parent. Useful for sub-tissues. :param ztissue: The zTissue object in the maya scene. :type ztissue: str

Returns (str) Parent mesh of zTissue, or None if none found

zBuilder.nodes.ziva.zivaBase module

class zBuilder.nodes.ziva.zivaBase.**Ziva**(*parent=None*, *maya_node=None*, *builder=None*, *deserialize=None*)

Bases: *zBuilder.nodes.deformer.Deformer*

Base node for Ziva type nodes.

extended from base to deal with maps and meshes and storing the solver.

EXTEND_ATTR_LIST = []

populate(*maya_node=None*)

This populates the node given a selection.

Parameters **maya_node** – Maya node to populate with.

Module contents

Submodules

zBuilder.nodes.base module

class zBuilder.nodes.base.**Base**(**args*, ***kwargs*)

Bases: object

SEARCH_EXCLUDE = ['_class', 'attrs', '_builder_type', 'type']

A list of attribute names in __dict__ to exclude from the string_replace method.

TYPES = None

add_child(*child*)

child(*row*)

child_count()

deserialize(*dictionary*)

Deserializes a node with given dict.

Takes a dictionary and goes through keys and fills up __dict__.

Args (dict): The given dict.

log(*tab_level=-1*)

long_name

Long name of parameter corresponding to long name of maya node. This property is not settable. To set it use self.name.

name

Name of parameter corresponding to maya node name. Setting this property will check for long name and store that. self.name still returns short name, self.long_name returns the stored long name.

parent()

row()

serialize()

Makes node serializable.

This replaces an mObject with the name of the object in scene to make it serializable for writing out to json. Then it loops through keys in dict and saves out a temp dict of items that can be serializable and returns that temp dict for json writing purposes.

Returns of serializable items

Return type dict

string_replace(*search*, *replace*)
Search and replaces items in the node. Uses regular expressions. Uses SEARCH_EXCLUDE to define attributes to exclude from this process.

Goes through the __dict__ and search and replace items.

Works with strings, lists of strings and dictionaries where the values are either strings or list of strings. More specific searches should be overridden here.

Parameters

- **search** (*str*) – string to search for.
- **replace** (*str*) – string to replace it with.

zBuilder.nodes.deformer module

class zBuilder.nodes.deformer.**Deformer**(*parent=None*, *maya_node=None*, *builder=None*, *deserialize=None*)
Bases: *zBuilder.nodes.dg_node.DGNode*

build(**args*, ***kwargs*)
Builds the node in maya. mean to be overwritten.

check_map_interpolation(*interp_maps*)
For each map it checks if it is topologically corresponding and if it isn't it will interpolate the map if the flag is True. Once the map has been interpolated it updates the stored value before it gets applied in Maya.

Parameters **interp_maps** (*bool*) – Do you want to do it?

get_map_meshes()
This is the mesh associated with each map in obj.MAP_LIST. Typically it seems to coincide with mesh store in get_association. Sometimes it deviates, so you can override this method to define your own list of meshes against the map list.

Returns List of long mesh names.

Return type list

get_map_names()
This builds the map names. maps from MAP_LIST with the object name in front

For this we want to get the .name and not scene name.

get_map_objects()
Returns:

get_mesh_objects()
Returns:

static get_meshes(*node*)
Queries the deformer and returns the meshes associated with it.

Parameters **node** – Maya node to query.

Returns list of strings of mesh names.

Return type list od strings

populate(*maya_node=None*)
Populates the node with the info from the passed maya node in args.

This is deals with basic stuff including attributes. For other things it is meant to be overridden in inherited node.

This is inherited from Base and extended to deal with maps and meshes. :param maya_node: The maya node to populate parameter with. :type maya_node: str

set_maya_weights(*interp_maps=False*)
Given a Builder node this set the map values of the object in the maya scene. It first does a mObject check to see if it has been tracked, if it has it uses that instead of stored scene_name.

Parameters **interp_maps** (*str*) – Do you want maps interpolated? True forces interpolation. False cancels it. auto checks if it needs to. Default = "auto"

Returns nothing.

spawn_parameters()
Returns:

zBuilder.nodes.dg_node module

class zBuilder.nodes.dg_node.**DGNode**(*parent=None*, *maya_node=None*, *builder=None*, *deserialize=None*)
Bases: *zBuilder.nodes.base.Base*

The base node for the node functionality of all nodes

Parameters

- **maya_node** (*str, optional*) – maya node to populate parameter with.
- **builder** (*object, optional*) – The builder object for reference.
- **deserialize** (*dict, optional*) – if given a dictionary to deserialize it fills hte parameter with contents of dictionary using the deserialize method.

Variables

- ***type*** (*str*) – type of parameter. Tied with maya node type.
- **attrs** (*dict*) – A place for the maya attributes dictionary.

EXTEND_ATTR_LIST = []
List of maya node attribute names to add to the auto generated attribute list to include.

MAP_LIST = []
List of maya node attribute names that represent the paintable map.

SEARCH_EXCLUDE = ['_class', 'attrs', '_builder_type', 'type']
A list of attribute names in __dict__ to exclude from the string_replace method.

TYPES = None
Types of maya nodes this parameter is aware of. Only needed if parameter can deal with multiple types. Else leave at None

association
associations of node.

build(**args*, ***kwargs*)
Builds the node in maya. meant to be overwritten.

compare()
Compares populated parameter with that which is in maya scene.

Returns prints out items that are different.

get_scene_name(*long_name=False*)
This checks stored mObject and gets name of maya object in scene. If no mObject it returns parameter name.

Parameters **long_name** (*bool*) – Return the fullpath or not. Defaults to False.

Returns (str) Name of maya object.

long_association
Long names of associations.

mobject
Gets mObject stored with parameter. :returns: mObject

mobject_reset()

populate(*maya_node=None*)
Populates the node with the info from the passed maya node in args.

This is deals with basic stuff including attributes. For other things it is meant to be overridden in inherited node.

Parameters **maya_node** (*str*) – The maya node to populate parameter with.

set_maya_attrs(*attr_filter=None*)
Given a Builder node this set the attributes of the object in the maya scene. It first does a mObject check to see if it has been tracked, if it has it uses that instead of stored name.

Parameters **attr_filter** (*dict*) – Attribute filter on what attributes to set. dictionary is key value where key is node type and value is list of attributes to use.

af = {'zSolver':['substeps']}

Returns nothing.

type = None

zBuilder.nodes.transform module

class zBuilder.nodes.transform.**TransformNode**(**args*, ***kwargs*)
Bases: *zBuilder.nodes.dg_node.DGNode*

EXTEND_ATTR_LIST = ['rotatePivotX', 'rotatePivotY', 'rotatePivotZ']

build()

populate(*transformName*)

type = 'transform'

zBuilder.nodes.transform.**build_transform**(*transformName*, *transformType*, *parentName*, *jointOrient*)

zBuilder.nodes.transform.**get_transformData_data**(*transformName*)

Module contents

7.1.3 zBuilder.parameters package

Submodules

zBuilder.parameters.maps module

class zBuilder.parameters.maps.**Map**(**args*, ***kwargs*)
Bases: *zBuilder.nodes.base.Base*

get_mesh(*long_name=False*)
Gets the stores name of the mesh associated with map.

Parameters **long_name** – Returns long name. Default to False

Returns Name of mesh.

get_mesh_component()
Gets the mesh data object.

Returns zBuilder data object of mesh.

interpolate()
Interpolates map against mesh in scene. Re-sets value.

is_topologically_corresponding()
Checks if mesh ih data object is corresponding with mesh in scene.

Returns True if they are, else False.

populate(*map_name=None*, *mesh_name=None*)
Populate node with that from the maya scene.

Parameters

- **map_name** – Name of map to populate it with.
- **mesh_name** – Name of mesh to populate it with.

set_mesh(*mesh*)
Stores the mesh name.

Parameters **mesh** – The mesh name to store.

type = 'map'

values = None
str – Docstring *after* attribute, with type specified.

zBuilder.parameters.maps.**get_weights**(*map_name*, *mesh_name*)
Gets the weights for the map. :param map_name: Map to get weights from. :param mesh_name: Mesh to check vert count.

Returns value of map

Raises ValueError – if there is a problem getting map.

zBuilder.parameters.maps.**interpolate_values**(*source_mesh*, *destination_mesh*, *weight_list*)

Description: Will transfer values between similar meshes with differing topology. Lerps values from triangleIndex of closest point on mesh.

Accepts: sourceMeshName, destinationMeshName - strings for each mesh transform

Returns:

zBuilder.parameters.mesh module

class zBuilder.parameters.mesh.**Mesh**(**args*, ***kwargs*)
Bases: *zBuilder.nodes.base.Base*

build_mesh()
Builds mesh in maya scene.

Returns mesh name.

get_point_list()
Get point List

get_polygon_connects()
Get polygon Connects

get_polygon_counts()
Get polygon counts.

is_topologically_corresponding()
Compare a mesh in scene with one saved in this node. Currently just checking if vert count is same. Need to update this to a better method.

Returns True if topologically corresponding, else False

mirror()
Mirrors internal mesh by negating translate X on all points.

populate(*mesh_name*)
Populate node with that from the maya scene.

Parameters **mesh_name** – Name of mesh to populate it with.

set_point_list(*point_list*)
Stores the point list.

Parameters **point_list** (*list*) – List of points.

set_polygon_connects(*pConnectList*)
Stores the polygon connects.

Parameters **pConnectList** (*list*) – The connect list.

set_polygon_counts(*pCountList*)
Stores the polygon counts.

Parameters **pCountList** (*list*) – The count list.

type = 'mesh'
Type of node.

zBuilder.parameters.mesh.**build_mesh**(*name*, *polygonCounts*, *polygonConnects*, *vertexArray*)
Builds mesh in maya scene. :param name: Name of mesh. :param polygonCounts: The polygon counts. :param polygonConnects: The polygon connects. :param vertexArray: The point list.

Returns Name of newly built mesh.

Module contents

7.2 Submodules

7.3 zBuilder.IO module

class zBuilder.IO.**BaseNodeEncoder**(*skipkeys=False*, *ensure_ascii=True*, *check_circular=True*, *allow_nan=True*, *sort_keys=False*, *indent=None*, *separators=None*, *encoding='utf-8'*, *default=None*)

Bases: json.encoder.JSONEncoder

default(*obj*)

zBuilder.IO.**check_data**(*data*)

Utility to check data format after loaded from josn. Used to check if data is wrapped in dictionary. If it isn't it wraps it. Used to deal with older zBuilder files.

Parameters **data** – Data to check.

Returns Result of operation.

zBuilder.IO.**dump_json**(*file_path*, *json_data*)

Saves a json file to disk given a file path and data.

Parameters

- **file_path** – The location to save the json file.
- **json_data** – The data to save in the json.

Returns file path if successful.

Raises IOError – If not able to write file.

zBuilder.IO.**find_class**(*module_*, *type_*)

Given a module and a type returns class object.

Parameters

- **module** (str) – The module to look for.
- **type** (str) – The type to look for.

Returns class object.

Return type obj

zBuilder.IO.**load_base_node**(*json_object*)

Loads json objects into proper classes. Serves as object hook for loading json.

Parameters **json_object** (*obj*) – json obj to perform action on

Returns Result of operation

Return type obj

zBuilder.IO.**load_json**(*file_path*)

loads a json file from disk given a file path.

Parameters **file_path** – The location to save the json file.

Returns json data

Raises IOError – If not able to read file.

zBuilder.IO.**update_json**(*json_object*)
This takes the json_object and updates it to work with zBuilder 1.0.0

Returns modified json_object

zBuilder.IO.**wrap_data**(*data*, *type_*)
Utility wrapper to identify data.

Parameters

- **data** –
- **type** (str) – The type of data it is.

7.4 zBuilder.builder module

class zBuilder.builder.**Builder**
Bases: object

The main entry point for using zBuilder.

build(**args*, ***kwargs*)

get_scene_items(*type_filter=[]*, *name_filter=[]*, *name_regex=None*, *association_filter=[]*, *association_regex=None*, *invert_match=False*)
Gets the scene items from builder for further inspection or modification.

Parameters

- **type_filter** (str or list, optional) – filter by parameter type. Defaults to list.
- **name_filter** (str or list, optional) – filter by parameter name. Defaults to list.
- **name_regex** (str) – filter by parameter name by regular expression. Defaults to None.
- **association_filter** (str or list, optional) – filter by parameter association. Defaults to list.
- **association_regex** (str) – filter by parameter association by regular expression. Defaults to None.
- **invert_match** (*bool*) – Invert the sense of matching, to select non-matching items. Defaults to False

Returns List of scene items.

Return type list

log()

node_factory(*node*, *parent=None*, *get_parameters=True*)
Given a maya node, this checks objType and instantiates the proper zBuilder.node and populates it and returns it.

Parameters

- **node** (str) – Name of maya node.
- **get_parameters** (*bool*) –

Returns zBuilder node populated.

Return type obj

parameter_factory(*type_*, *stuff*)

print_(*type_filter=[]*, *name_filter=[]*)
Prints out basic information for each scene item in the Builder. Information is all information that is stored in the __dict__. Useful for trouble shooting.

Parameters

- **type_filter** (`list` or `str`) – filter by parameter type. Defaults to `list`
- **name_filter** (`list` or `str`) – filter by parameter name. Defaults to `list`

retrieve_from_file(*file_path*)
Reads scene items from a given file. The items get placed in the bundle.

Parameters **file_path** (`str`) – The file path to read from disk.

retrieve_from_scene(**args*, ***kwargs*)
must create a method to inherit this class

stats()
Prints out basic information in Maya script editor. Information is scene item types and counts.

string_replace(*search*, *replace*)
Searches and replaces with regular expressions scene items in the builder.

Parameters

- **search** (`str`) – what to search for
- **replace** (`str`) – what to replace it with

Example

replace *r_* at front of item with *l_*:

```
>>> z.string_replace('^r_','l_')
```

replace *_r* at end of line with *_l*:

```
>>> z.string_replace('_r$','_l')
```

static time_this(*original_function*)
A decorator to time functions.

view()

write(*file_path*, *type_filter=[]*, *invert_match=False*)
Writes out the scene items to a json file given a file path.

Parameters

- **file_path** (*str*) – The file path to write to disk.
- **type_filter** (*list, optional*) – Types of scene items to write.
- **invert_match** (*bool*) – Invert the sense of matching, to select non-matching items. Defaults to `False`

7.5 zBuilder.bundle module

class zBuilder.bundle.**Bundle**
Bases: `object`

Mixin class to deal with storing node data and component data. meant to be inherited by main.

append_scene_item(*scene_item*)
appends a parameter to the parameter list. Checks if parameter is already in list, if it is it overrides the previous one.

Parameters **scene_item** (obj) – the parameter to append to collection list.

compare(*type_filter=[]*, *name_filter=[]*)
Compares info in memory with that which is in scene.

Parameters

- **type_filter** (list or str) – filter by parameter type. Defaults to list
- **name_filter** (list or str) – filter by parameter name. Defaults to list

extend_scene_items(*scene_items*)

Parameters **scene_items** –

Returns:

get_scene_items(*type_filter=[]*, *name_filter=[]*, *name_regex=None*, *association_filter=[]*, *association_regex=None*, *invert_match=False*)
Gets the scene items from builder for further inspection or modification.

Parameters

- **type_filter** (str or list, optional) – filter by parameter type. Defaults to list.
- **name_filter** (str or list, optional) – filter by parameter name. Defaults to list.
- **name_regex** (str) – filter by parameter name by regular expression. Defaults to None.
- **association_filter** (str or list, optional) – filter by parameter association. Defaults to list.
- **association_regex** (str) – filter by parameter association by regular expression. Defaults to None.
- **invert_match** (*bool*) – Invert the sense of matching, to select non-matching items. Defaults to False

Returns List of scene items.

Return type list

print_(*type_filter=[]*, *name_filter=[]*)
Prints out basic information for each scene item in the Builder. Information is all information that is stored in the __dict__. Useful for trouble shooting.

Parameters

- **type_filter** (list or str) – filter by parameter type. Defaults to list
- **name_filter** (list or str) – filter by parameter name. Defaults to list

remove_scene_item(*scene_item*)
Removes a scene_item from the bundle list while keeping order. :param scene_item: The scene_item object to remove. :type scene_item: obj

stats(*type_filter=''*)
Prints out basic information in Maya script editor. Information is scene item types and counts.

Parameters **type_filter** (str) – filter by parameter type. Defaults to str

string_replace(*search*, *replace*)
Searches and replaces with regular expressions the scene items in the builder.

Parameters

- **search** (str) – what to search for
- **replace** (str) – what to replace it with

Example

replace *r_* at front of item with *l_*:

```
>>> z.string_replace('^r_','l_')
```

replace *_r* at end of line with *_l*:

```
>>> z.string_replace('_r$','_l')
```

7.6 zBuilder.util module

zBuilder.util.**check_map_validity**()
This checks the map validity for zAttachments and zFibers. For zAttachments it checks if all the values are zero. If so it failed and turns off the associated zTissue node. For zFibers it checks to make sure there are at least 1 value of 0 and 1 value of .5 within a .1 threshold. If not that fails and turns off the zTissue

Returns list of offending maps

zBuilder.util.**copy_paste**(**args*, ***kwargs*)
A utility wrapper for copying and pasting a tissue

7.7 zBuilder.zMaya module

zBuilder.zMaya.**ZNODES = ['zGeo', 'zSolver', 'zSolverTransform', 'zIsoMesh', 'zDelaunayTetMe**
All available ziva nodes to be able to cleanup.

zBuilder.zMaya.**build_attr_key_values**(*selection*, *attr_list*)
Builds a dictionary of attribute key/values. Stores the value, type, and locked status. :param selection: Items to save attrbutes for. :param attr_list: List of attributes to save.

Returns of attribute values.

Return type dict

zBuilder.zMaya.**build_attr_list**(*selection*, *attr_filter=None*)
Builds a list of attributes to store values for. It is looking at keyable attributes and if they are in channelBox.

Parameters

- **attr_filter** –
- **selection** (*str*) – maya object to find attributes

Returns list of attributes names

Return type list

zBuilder.zMaya.**check_body_type**(*bodies*)
Checks if given bodies are either zTissue, zCloth and or zBone. Mostly used to see if we can create a zAttachment before we try. Additionally does a check if all objects exist in scene.

Parameters **bodies** (*list*) – List of bodies we want to check type of.

Returns True if all bodies pass test, else False.

Return type (bool)

zBuilder.zMaya.**check_map_validity**(*map_parameters*)
This checks the map validity for zAttachments and zFibers. For zAttachments it checks if all the values are zero. If so it failed and turns off the associated zTissue node. For zFibers it checks to make sure there are at least 1 value of 0 and 1 value of .5 within a .1 threshold. If not that fails and turns off the zTissue

Parameters **map_parameters** – map parameters to check.

Returns list of offending maps

zBuilder.zMaya.**check_maya_node**(*maya_node*)

Parameters **maya_node** –

Returns:

zBuilder.zMaya.**check_mesh_quality**(*meshes*)
Light wrapper around checking mesh quality.

Parameters **meshes** (*list*) – A list of meshes you want to check

Raises StandardError – If any mesh does not pass mesh check

zBuilder.zMaya.**clean_scene**()
Deletes all ziva nodes in scene. Effectively cleaning it up.

zBuilder.zMaya.**cull_creation_nodes**(*parameters*, *permissive=True*)
To help speed up the build of a Ziva setup we are creating the bones and the tissues with one command. Given a list of zBuilder nodes this checks if a given node needs to be created in scene. Checks to see if it already exists or if associated mesh is missing. Either case it culls it from list.

Parameters

- **permissive** (*bool*) –
- **parameters** (*object*) – the zBuilder nodes to check.

Returns Dictionary of non culled

Return type dict

zBuilder.zMaya.**get_association**(*zNode*)
Gets an association of given zNode

Parameters **zNode** (*string*) – the zNode to find association of.

zBuilder.zMaya.**get_fiber_lineofaction**(*zFiber*)
Gets the zLineOfAction node hooked up to a given zFiber in any.

Parameters **zFiber** (*string*) – the zFiber to query.

Returns zLineOfAction

Return type str

zBuilder.zMaya.**get_lineOfAction_fiber**(*zlineofaction*)
Gets the zFiber node hooked up to a given zLineOfAction in any.

Parameters **zlineofaction** (*string*) – the zLineOfAction to query.

Returns Name of zFiber hooked up to lineOfAction

Return type str

zBuilder.zMaya.**get_mdagpath_from_mesh**(*mesh_name*)
Maya stuff, getting the dagpath from a mesh name

Parameters **mesh_name** – The mesh to get dagpath from.

zBuilder.zMaya.**get_mesh_connectivity**(*mesh_name*)
Gets mesh connectivity for given mesh.

Parameters **mesh_name** – Name of mesh to process.

Returns Dictionary of polygonCounts, polygonConnects, and points.

Return type dict

zBuilder.zMaya.**get_name_from_m_object**(*m_object*, *long_name=True*)
Gets maya scene name from given mObject. :param m_object: The m_object to get name from. :param long_name: Returns long name. Default = `True`

Returns Maya object name.

Return type str

zBuilder.zMaya.**get_type**(*body*)
Really light wrapper for getting type of maya node. Ya, I know.

Parameters **body** (*str*) – Maya node to get type of

Returns String of node type.

Return type str

zBuilder.zMaya.**get_zAttachments**(*bodies*)
Gets zAttachments in scene. :param body: Maya node to find associated zAttachments.

Returns string of name of zAttachments.

zBuilder.zMaya.**get_zBones**(*bodies*)
Gets zBones in scene. :param body: Maya node to find associated zBones.

Returns string of name of zBones.

zBuilder.zMaya.**get_zCloth**(*bodies*)
Gets zCloth in scene. :param body: Maya node to find associated zCloth.

Returns string of name of zCloth.

zBuilder.zMaya.**get_zFibers**(*bodies*)
Gets zFibers in scene. :param body: Maya node to find associated zFibers.

Returns string of name of zFibers.

zBuilder.zMaya.**get_zMaterials**(*bodies*)
Gets zmaterials in scene. :param body: Maya node to find associated zMaterials.

Returns string of name of zmaterials.

zBuilder.zMaya.**get_zSolver**(*body*)
Gets zSolver in scene. :param body: Maya node to find associated solver.

Returns returns long name of zSolver.

zBuilder.zMaya.**get_zSolverTransform**(*body*)
Gets zSolverTransform in scene. :param body: Maya node to find associated solverTransform.

Returns returns long name of zSolverTransform.

zBuilder.zMaya.**get_zTet_user_mesh**(*zTet*)
Gets the user tet mesh hooked up to a given zTet in any.

Parameters **zTet** (*string*) – the zTet to query.

Returns User tet mesh.

Return type str

zBuilder.zMaya.**get_zTets**(*bodies*)
Gets zTets in scene. :param body: Maya node to find associated zTets.

Returns string of name of zTets.

zBuilder.zMaya.**get_zTissues**(*bodies*)
Gets zTissues in scene. :param body: Maya node to find associated zTissues.

Returns string of name of zTissues.

zBuilder.zMaya.**isSolver**(*selection*)
Checks if passed is zSolver or zSolverTransform. :param selection: Item of interest.

Returns True if it is solver, else false.

zBuilder.zMaya.**parse_maya_node_for_selection**(*args*)
This is used to check passed args in a function to see if they are valid maya objects in the current scene. If any of the passed names are not in the it raises a StandardError. If nothing is passed it looks at what is actively selected in scene to get selection. This way it functions like a lot of the maya tools, uses what is passed OR it uses what is selected.

Parameters **args** – The args to test

Returns (list) maya selection

zBuilder.zMaya.**rename_ziva_nodes**(*replace=['_muscle', '_bone']*)
Renames zNodes based on mesh it's connected to.

Parameters **replace** (*list*) – subset of mesh name to replace with zNode name

- zFiber: <meshName>_zFiber
- zMaterial: <meshName>_zMaterial
- zTet: <meshName>_zTet
- zTissue: <meshName>_zTissue
- zBone: <meshName>_zBone
- zCloth: <meshName>_zCloth
- zAttachment: <sourceMesh>__<destinationMesh>_zAttachment

zBuilder.zMaya.**replace_dict_keys**(*search*, *replace*, *dictionary*)
Does a search and replace on dictionary keys

Parameters

- **search** (str) – search term
- **replace** (str) – replace term

- **dictionary** (dict) – the dictionary to do search on

Returns result of search and replace

Return type dict

zBuilder.zMaya.**replace_long_name**(*search*, *replace*, *long_name*)
does a search and replace on a long name. It splits it up by ('|') then performs it on each piece

Parameters

- **search** (*str*) – search term
- **replace** (*str*) – replace term
- **long_name** (*str*) – the long name to perform action on

Returns result of search and replace

Return type str

zBuilder.zMaya.**select_tissue_meshes**()
Selects all zTissues in scene

7.8 Module contents

Python Module Index

Z

zBuilder, 51
zBuilder.builder, 44
zBuilder.builders, 28
zBuilder.builders.attributes, 25
zBuilder.builders.constraints, 25
zBuilder.builders.deltaMush, 25
zBuilder.builders.selection, 26
zBuilder.builders.skinClusters, 26
zBuilder.builders.ziva, 26
zBuilder.bundle, 45
zBuilder.IO, 43
zBuilder.nodes, 41
zBuilder.nodes.base, 37
zBuilder.nodes.deformer, 38
zBuilder.nodes.deformers, 30
zBuilder.nodes.deformers.blendShape, 28
zBuilder.nodes.deformers.deltaMush, 28
zBuilder.nodes.deformers.skinCluster, 29
zBuilder.nodes.deformers.wrap, 29
zBuilder.nodes.dg_node, 39
zBuilder.nodes.transform, 40
zBuilder.nodes.utils, 30
zBuilder.nodes.utils.constraint, 30
zBuilder.nodes.ziva, 37
zBuilder.nodes.ziva.zAttachment, 30
zBuilder.nodes.ziva.zBone, 31
zBuilder.nodes.ziva.zCloth, 31
zBuilder.nodes.ziva.zEmbedder, 32
zBuilder.nodes.ziva.zFiber, 33
zBuilder.nodes.ziva.zivaBase, 37
zBuilder.nodes.ziva.zLineOfAction, 33
zBuilder.nodes.ziva.zMaterial, 34
zBuilder.nodes.ziva.zSolver, 34
zBuilder.nodes.ziva.zSolverTransform, 35
zBuilder.nodes.ziva.zTet, 35
zBuilder.nodes.ziva.zTissue, 36
zBuilder.parameters, 43
zBuilder.parameters.maps, 41
zBuilder.parameters.mesh, 42
zBuilder.util, 47
zBuilder.zMaya, 47

Index

A

add_child() (zBuilder.nodes.base.Base method), 37
append_scene_item() (zBuilder.bundle.Bundle method), 46
apply_multiple() (in module zBuilder.nodes.ziva.zBone), 31
apply_user_tet_mesh() (zBuilder.nodes.ziva.zTet.TetNode method), 35
apply_weights() (in module zBuilder.nodes.deformers.skinCluster), 29
association (zBuilder.nodes.dg_node.DGNode attribute), 39
AttachmentNode (class in zBuilder.nodes.ziva.zAttachment), 30
Attributes (class in zBuilder.builders.attributes), 25

B

Base (class in zBuilder.nodes.base), 37
BaseNodeEncoder (class in zBuilder.IO), 43
BlendShape (class in zBuilder.nodes.deformers.blendShape), 28
body, **23**
BoneNode (class in zBuilder.nodes.ziva.zBone), 31
build(), **23**
build() (zBuilder.builder.Builder method), 44
build() (zBuilder.builders.attributes.Attributes method), 25
build() (zBuilder.builders.constraints.Constraints method), 25
build() (zBuilder.builders.deltaMush.DeltaMush method), 26
build() (zBuilder.builders.selection.Selection method), 26
build() (zBuilder.builders.skinClusters.SkinCluster method), 26
build() (zBuilder.builders.ziva.Ziva method), 15, 26
build() (zBuilder.nodes.deformer.Deformer method), 38
build() (zBuilder.nodes.deformers.blendShape.BlendShape method), 28
build() (zBuilder.nodes.deformers.deltaMush.DeltaMush method), 28
build() (zBuilder.nodes.deformers.skinCluster.SkinCluster method), 29
build() (zBuilder.nodes.deformers.wrap.Wrap method), 29
build() (zBuilder.nodes.dg_node.DGNode method), 39
build() (zBuilder.nodes.transform.TransformNode method), 40
build() (zBuilder.nodes.utils.constraint.Constraint method), 30
build() (zBuilder.nodes.ziva.zAttachment.AttachmentNode method), 31
build() (zBuilder.nodes.ziva.zBone.BoneNode method), 31
build() (zBuilder.nodes.ziva.zCloth.ClothNode method), 31
build() (zBuilder.nodes.ziva.zEmbedder.EmbedderNode method), 32
build() (zBuilder.nodes.ziva.zFiber.FiberNode method), 33
build() (zBuilder.nodes.ziva.zLineOfAction.LineOfActionNode method), 33
build() (zBuilder.nodes.ziva.zMaterial.MaterialNode method), 34
build() (zBuilder.nodes.ziva.zSolver.SolverNode method), 34
build() (zBuilder.nodes.ziva.zSolverTransform.SolverTransformNode method), 35
build() (zBuilder.nodes.ziva.zTet.TetNode method), 35
build() (zBuilder.nodes.ziva.zTissue.TissueNode method), 36
build_attr_key_values() (in module zBuilder.zMaya), 47
build_attr_list() (in module zBuilder.zMaya), 47
build_mesh() (in module zBuilder.parameters.mesh), 42
build_mesh() (zBuilder.parameters.mesh.Mesh method), 42
build_multiple() (in module zBuilder.nodes.ziva.zTissue), 36
build_transform() (in module zBuilder.nodes.transform), 40

builder, 23
Builder (class in zBuilder.builder), 44
Builder (class in zBuilder.builders.ziva), 16
Bundle (class in zBuilder.bundle), 45

C

check_body_type() (in module zBuilder.zMaya), 47
check_data() (in module zBuilder.IO), 43
check_map_interpolation() (zBuilder.nodes.deformer.Deformer method), 38
check_map_validity() (in module zBuilder.util), 47
check_map_validity() (in module zBuilder.zMaya), 48
check_maya_node() (in module zBuilder.zMaya), 48
check_mesh_quality() (in module zBuilder.zMaya), 48
child() (zBuilder.nodes.base.Base method), 37
child_count() (zBuilder.nodes.base.Base method), 37
clean_scene() (in module zBuilder.zMaya), 48
ClothNode (class in zBuilder.nodes.ziva.zCloth), 31
compare() (zBuilder.bundle.Bundle method), 46
compare() (zBuilder.nodes.dg_node.DGNode method), 39
constrained (zBuilder.nodes.utils.constraint.Constraint attribute), 30
Constraint (class in zBuilder.nodes.utils.constraint), 30
Constraints (class in zBuilder.builders.constraints), 25
copy_paste() (in module zBuilder.util), 47
cull_creation_nodes() (in module zBuilder.zMaya), 48

D

default() (zBuilder.IO.BaseNodeEncoder method), 43
Deformer (class in zBuilder.nodes.deformer), 38
DeltaMush (class in zBuilder.builders.deltaMush), 25
DeltaMush (class in zBuilder.nodes.deformers.deltaMush), 28
deserialize() (zBuilder.nodes.base.Base method), 37
DGNode (class in zBuilder.nodes.dg_node), 39
dump_json() (in module zBuilder.IO), 43

E

EmbedderNode (class in zBuilder.nodes.ziva.zEmbedder), 32
EXTEND_ATTR_LIST (zBuilder.nodes.deformers.blendShape.BlendShape attribute), 28
EXTEND_ATTR_LIST (zBuilder.nodes.deformers.skinCluster.SkinCluster attribute), 29
EXTEND_ATTR_LIST (zBuilder.nodes.dg_node.DGNode attribute), 39
EXTEND_ATTR_LIST (zBuilder.nodes.transform.TransformNode attribute), 40
EXTEND_ATTR_LIST (zBuilder.nodes.utils.constraint.Constraint attribute), 30
EXTEND_ATTR_LIST (zBuilder.nodes.ziva.zivaBase.Ziva attribute), 37
extend_scene_items() (zBuilder.bundle.Bundle method), 46

F

FiberNode (class in zBuilder.nodes.ziva.zFiber), 33
find_class() (in module zBuilder.IO), 43

G

get_association() (in module zBuilder.zMaya), 48
get_associations() (in module zBuilder.nodes.deformers.skinCluster), 29
get_collision_meshes() (zBuilder.nodes.ziva.zEmbedder.EmbedderNode method), 32
get_constrained() (in module zBuilder.nodes.utils.constraint), 30
get_embedded_meshes() (in module zBuilder.nodes.ziva.zEmbedder), 33
get_embedded_meshes() (zBuilder.nodes.ziva.zEmbedder.EmbedderNode method), 32
get_fiber_lineofaction() (in module zBuilder.zMaya), 48
get_influences() (in module zBuilder.nodes.deformers.skinCluster), 29
get_lineOfAction_fiber() (in module zBuilder.zMaya), 48
get_map_meshes() (zBuilder.nodes.deformer.Deformer method), 38
get_map_meshes() (zBuilder.nodes.deformers.blendShape.BlendShape method), 28
get_map_meshes() (zBuilder.nodes.ziva.zFiber.FiberNode method), 33
get_map_names() (zBuilder.nodes.deformer.Deformer method), 38
get_map_objects() (zBuilder.nodes.deformer.Deformer method), 38
get_mdagpath_from_mesh() (in module zBuilder.zMaya), 49
get_mesh() (zBuilder.parameters.maps.Map method), 41
get_mesh_component() (zBuilder.parameters.maps.Map method), 41
get_mesh_connectivity() (in module zBuilder.zMaya), 49
get_mesh_objects() (zBuilder.nodes.deformer.Deformer method), 38
get_meshes() (zBuilder.nodes.deformer.Deformer static method), 38
get_meshes() (zBuilder.nodes.deformers.wrap.Wrap static method), 29
get_name_from_m_object() (in module zBuilder.zMaya), 49
get_parent() (zBuilder.builders.ziva.Ziva method), 27
get_point_list() (zBuilder.parameters.mesh.Mesh method), 42
get_polygon_connects() (zBuilder.parameters.mesh.Mesh method), 42
get_polygon_counts() (zBuilder.parameters.mesh.Mesh method), 42

get_scene_items() (zBuilder.builder.Builder method), 44
get_scene_items() (zBuilder.builders.ziva.Builder method), 16
get_scene_items() (zBuilder.bundle.Bundle method), 46
get_scene_name() (zBuilder.nodes.dg_node.DGNode method), 40
get_target() (in module zBuilder.nodes.deformers.blendShape), 28
get_targets() (in module zBuilder.nodes.utils.constraint), 30
get_tissue_children() (in module zBuilder.nodes.ziva.zTissue), 36
get_tissue_parent() (in module zBuilder.nodes.ziva.zTissue), 36
get_transformData_data() (in module zBuilder.nodes.transform), 40
get_type() (in module zBuilder.zMaya), 49
get_user_tet_mesh() (zBuilder.nodes.ziva.zTet.TetNode method), 35
get_weights() (in module zBuilder.nodes.deformers.skinCluster), 29
get_weights() (in module zBuilder.parameters.maps), 41
get_zAttachments() (in module zBuilder.zMaya), 49
get_zBones() (in module zBuilder.zMaya), 49
get_zCloth() (in module zBuilder.zMaya), 49
get_zFibers() (in module zBuilder.zMaya), 49
get_zMaterials() (in module zBuilder.zMaya), 49
get_zSolver() (in module zBuilder.zMaya), 49
get_zSolverTransform() (in module zBuilder.zMaya), 49
get_zTet_user_mesh() (in module zBuilder.zMaya), 50
get_zTets() (in module zBuilder.zMaya), 50
get_zTissues() (in module zBuilder.zMaya), 50

I

interpolate() (zBuilder.parameters.maps.Map method), 41
interpolate_values() (in module zBuilder.parameters.maps), 41
is_topologically_corresponding() (zBuilder.parameters.maps.Map method), 41
is_topologically_corresponding() (zBuilder.parameters.mesh.Mesh method), 42
isSolver() (in module zBuilder.zMaya), 50

L

LineOfActionNode (class in zBuilder.nodes.ziva.zLineOfAction), 33
load_base_node() (in module zBuilder.IO), 43
load_json() (in module zBuilder.IO), 43
log() (zBuilder.builder.Builder method), 44
log() (zBuilder.nodes.base.Base method), 37
long_association (zBuilder.nodes.dg_node.DGNode attribute), 40
long_name (zBuilder.nodes.base.Base attribute), 37
long_target (zBuilder.nodes.deformers.blendShape.BlendShape attribute), 28

M

map, **23**
Map (class in zBuilder.parameters.maps), 41
MAP_LIST (zBuilder.nodes.deformers.blendShape.BlendShape attribute), 28
MAP_LIST (zBuilder.nodes.deformers.deltaMush.DeltaMush attribute), 28
MAP_LIST (zBuilder.nodes.dg_node.DGNode attribute), 39
MAP_LIST (zBuilder.nodes.ziva.zAttachment.AttachmentNode attribute), 30
MAP_LIST (zBuilder.nodes.ziva.zFiber.FiberNode attribute), 33
MAP_LIST (zBuilder.nodes.ziva.zMaterial.MaterialNode attribute), 34
MAP_LIST (zBuilder.nodes.ziva.zTet.TetNode attribute), 35
MaterialNode (class in zBuilder.nodes.ziva.zMaterial), 34
Mesh (class in zBuilder.parameters.mesh), 42
mirror() (zBuilder.parameters.mesh.Mesh method), 42
mobject (zBuilder.nodes.dg_node.DGNode attribute), 40
mobject_reset() (zBuilder.nodes.dg_node.DGNode method), 40

N

name (zBuilder.nodes.base.Base attribute), 37
node, **23**
node_factory() (zBuilder.builder.Builder method), 44

P

parameter, **23**
parameter_factory() (zBuilder.builder.Builder method), 44
parent() (zBuilder.nodes.base.Base method), 37
parse_maya_node_for_selection() (in module zBuilder.zMaya), 50
populate() (zBuilder.nodes.deformer.Deformer method), 39
populate() (zBuilder.nodes.deformers.blendShape.BlendShape method), 28
populate() (zBuilder.nodes.deformers.skinCluster.SkinCluster method), 29
populate() (zBuilder.nodes.dg_node.DGNode method), 40
populate() (zBuilder.nodes.transform.TransformNode method), 40
populate() (zBuilder.nodes.utils.constraint.Constraint method), 30
populate() (zBuilder.nodes.ziva.zEmbedder.EmbedderNode method), 32

populate() (zBuilder.nodes.ziva.zivaBase.Ziva method), 37
populate() (zBuilder.nodes.ziva.zLineOfAction.LineOfActionNode method), 34
populate() (zBuilder.nodes.ziva.zTissue.TissueNode method), 36
populate() (zBuilder.parameters.maps.Map method), 41
populate() (zBuilder.parameters.mesh.Mesh method), 42
print_() (zBuilder.builder.Builder method), 44
print_() (zBuilder.builders.ziva.Builder method), 16
print_() (zBuilder.bundle.Bundle method), 46

R

remove_scene_item() (zBuilder.bundle.Bundle method), 46
rename_ziva_nodes() (in module zBuilder.zMaya), 50
replace_dict_keys() (in module zBuilder.zMaya), 50
replace_long_name() (in module zBuilder.zMaya), 51
reset_solvers() (zBuilder.builders.ziva.Ziva method), 27
retrieve(), **23**
retrieve_from_file() (zBuilder.builder.Builder method), 45
retrieve_from_file() (zBuilder.builders.ziva.Builder method), 16
retrieve_from_scene() (zBuilder.builder.Builder method), 45
retrieve_from_scene() (zBuilder.builders.attributes.Attributes method), 25
retrieve_from_scene() (zBuilder.builders.constraints.Constraints method), 25
retrieve_from_scene() (zBuilder.builders.deltaMush.DeltaMush method), 26
retrieve_from_scene() (zBuilder.builders.selection.Selection method), 26
retrieve_from_scene() (zBuilder.builders.skinClusters.SkinCluster method), 26
retrieve_from_scene() (zBuilder.builders.ziva.Ziva method), 15, 27
retrieve_from_scene_selection() (zBuilder.builders.ziva.Ziva method), 27
rig, **23**
row() (zBuilder.nodes.base.Base method), 37

S

scene item, **23**
SEARCH_EXCLUDE (zBuilder.nodes.base.Base attribute), 37
SEARCH_EXCLUDE (zBuilder.nodes.deformers.skinCluster.SkinCluster attribute), 29
SEARCH_EXCLUDE (zBuilder.nodes.dg_node.DGNode attribute), 39
SEARCH_EXCLUDE (zBuilder.nodes.utils.constraint.Constraint attribute), 30
select_tissue_meshes() (in module zBuilder.zMaya), 51
Selection (class in zBuilder.builders.selection), 26
serialize() (zBuilder.nodes.base.Base method), 37
set_collision_meshes() (zBuilder.nodes.ziva.zEmbedder.EmbedderNode method), 32
set_embedded_meshes() (zBuilder.nodes.ziva.zEmbedder.EmbedderNode method), 32
set_maya_attrs() (zBuilder.nodes.dg_node.DGNode method), 40
set_maya_weights() (zBuilder.nodes.deformer.Deformer method), 39
set_mesh() (zBuilder.parameters.maps.Map method), 41
set_point_list() (zBuilder.parameters.mesh.Mesh method), 42
set_polygon_connects() (zBuilder.parameters.mesh.Mesh method), 42
set_polygon_counts() (zBuilder.parameters.mesh.Mesh method), 42
set_user_tet_mesh() (zBuilder.nodes.ziva.zTet.TetNode method), 35
setup, **23**
SkinCluster (class in zBuilder.builders.skinClusters), 26
SkinCluster (class in zBuilder.nodes.deformers.skinCluster), 29
SolverNode (class in zBuilder.nodes.ziva.zSolver), 34
SolverTransformNode (class in zBuilder.nodes.ziva.zSolverTransform), 35
spawn_parameters() (zBuilder.nodes.deformer.Deformer method), 39
spawn_parameters() (zBuilder.nodes.ziva.zLineOfAction.LineOfActionNode method), 34
stats() (zBuilder.builder.Builder method), 45
stats() (zBuilder.builders.ziva.Builder method), 16
stats() (zBuilder.bundle.Bundle method), 46
string_replace() (zBuilder.builder.Builder method), 45
string_replace() (zBuilder.builders.ziva.Builder method), 16
string_replace() (zBuilder.bundle.Bundle method), 46
string_replace() (zBuilder.nodes.base.Base method), 38

T

target (zBuilder.nodes.deformers.blendShape.BlendShape attribute), 28
targets (zBuilder.nodes.utils.constraint.Constraint attribute), 30
TetNode (class in zBuilder.nodes.ziva.zTet), 35
time_this() (zBuilder.builder.Builder static method), 45
TissueNode (class in zBuilder.nodes.ziva.zTissue), 36
TransformNode (class in zBuilder.nodes.transform), 40
type (zBuilder.nodes.deformers.blendShape.BlendShape attribute), 28
type (zBuilder.nodes.deformers.deltaMush.DeltaMush attribute), 28
type (zBuilder.nodes.deformers.skinCluster.SkinCluster attribute), 29

type (zBuilder.nodes.deformers.wrap.Wrap attribute), 29
type (zBuilder.nodes.dg_node.DGNode attribute), 40
type (zBuilder.nodes.transform.TransformNode attribute), 40
type (zBuilder.nodes.utils.constraint.Constraint attribute), 30
type (zBuilder.nodes.ziva.zAttachment.AttachmentNode attribute), 31
type (zBuilder.nodes.ziva.zBone.BoneNode attribute), 31
type (zBuilder.nodes.ziva.zCloth.ClothNode attribute), 32
type (zBuilder.nodes.ziva.zEmbedder.EmbedderNode attribute), 32
type (zBuilder.nodes.ziva.zFiber.FiberNode attribute), 33
type (zBuilder.nodes.ziva.zLineOfAction.LineOfActionNode attribute), 34
type (zBuilder.nodes.ziva.zMaterial.MaterialNode attribute), 34
type (zBuilder.nodes.ziva.zSolver.SolverNode attribute), 34
type (zBuilder.nodes.ziva.zSolverTransform.SolverTransformNode attribute), 35
type (zBuilder.nodes.ziva.zTet.TetNode attribute), 35
type (zBuilder.nodes.ziva.zTissue.TissueNode attribute), 36
type (zBuilder.parameters.maps.Map attribute), 41
type (zBuilder.parameters.mesh.Mesh attribute), 42
TYPES (zBuilder.nodes.base.Base attribute), 37
TYPES (zBuilder.nodes.deformers.skinCluster.SkinCluster attribute), 29
TYPES (zBuilder.nodes.dg_node.DGNode attribute), 39
TYPES (zBuilder.nodes.utils.constraint.Constraint attribute), 30

U

update_json() (in module zBuilder.IO), 43

V

values (zBuilder.parameters.maps.Map attribute), 41
view() (zBuilder.builder.Builder method), 45

W

Wrap (class in zBuilder.nodes.deformers.wrap), 29
wrap_data() (in module zBuilder.IO), 44
write() (zBuilder.builder.Builder method), 45
write() (zBuilder.builders.ziva.Builder method), 16

Z

zBuilder (module), 51
zBuilder.builder (module), 44
zBuilder.builders (module), 28
zBuilder.builders.attributes (module), 25
zBuilder.builders.constraints (module), 25
zBuilder.builders.deltaMush (module), 25
zBuilder.builders.selection (module), 26
zBuilder.builders.skinClusters (module), 26
zBuilder.builders.ziva (module), 26
zBuilder.bundle (module), 45
zBuilder.IO (module), 43
zBuilder.nodes (module), 41
zBuilder.nodes.base (module), 37
zBuilder.nodes.deformer (module), 38
zBuilder.nodes.deformers (module), 30
zBuilder.nodes.deformers.blendShape (module), 28
zBuilder.nodes.deformers.deltaMush (module), 28
zBuilder.nodes.deformers.skinCluster (module), 29
zBuilder.nodes.deformers.wrap (module), 29
zBuilder.nodes.dg_node (module), 39
zBuilder.nodes.transform (module), 40
zBuilder.nodes.utils (module), 30
zBuilder.nodes.utils.constraint (module), 30
zBuilder.nodes.ziva (module), 37
zBuilder.nodes.ziva.zAttachment (module), 30
zBuilder.nodes.ziva.zBone (module), 31
zBuilder.nodes.ziva.zCloth (module), 31
zBuilder.nodes.ziva.zEmbedder (module), 32
zBuilder.nodes.ziva.zFiber (module), 33
zBuilder.nodes.ziva.zivaBase (module), 37
zBuilder.nodes.ziva.zLineOfAction (module), 33
zBuilder.nodes.ziva.zMaterial (module), 34
zBuilder.nodes.ziva.zSolver (module), 34
zBuilder.nodes.ziva.zSolverTransform (module), 35
zBuilder.nodes.ziva.zTet (module), 35
zBuilder.nodes.ziva.zTissue (module), 36
zBuilder.parameters (module), 43
zBuilder.parameters.maps (module), 41
zBuilder.parameters.mesh (module), 42
zBuilder.util (module), 47
zBuilder.zMaya (module), 47
Ziva (class in zBuilder.builders.ziva), 15, 26
Ziva (class in zBuilder.nodes.ziva.zivaBase), 37
zNode, **23**
ZNODES (in module zBuilder.zMaya), 47
zUi, **23**