

---

# **PyFrost**

***Release 1.0.0***

**unknown**

**Dec 16, 2020**



## GENERAL

<b>1</b>	<b>Introduction</b>	<b>3</b>
<b>2</b>	<b>Installation</b>	<b>5</b>
<b>3</b>	<b>Usage</b>	<b>7</b>
<b>4</b>	<b>pyfrost</b>	<b>9</b>
<b>5</b>	<b>Release Notes</b>	<b>15</b>
<b>6</b>	<b>Module Index</b>	<b>17</b>
<b>7</b>	<b>Index</b>	<b>19</b>
	<b>Python Module Index</b>	<b>21</b>
	<b>Index</b>	<b>23</b>



Python API for Maya Bifrost



## INTRODUCTION

PyFrost is an object oriented Python API for Maya Bifrost. It simply wraps the `cmds.vnn` commands.

It is still very early and has a lot of hardcoded stuff, but hopefully it will grow nicely over time (unless Autodesk provides a proper API!)

### 1.1 License

The current repository is under [MIT License](#).

Feel free to use, change, and share it as you please. You don't have to, but mentioning my name whenever you use source code from here would be much appreciated!

### 1.2 API Documentation

You can find a generated sphinx documentation at <https://pyfrost-maya.readthedocs.io/en/latest/>





## INSTALLATION

PyFrost requires Autodesk Maya  $\geq$  2018 and the latest version of [Bifrost](#), currently [2.2.0.1](#)

You can find a module file available in `pyfrost\src\module\modules\` which you can add to the `MAYA_MODULE_PATH` environment variable. It'll allow Maya to pick up the whole repository automatically for you on startup.

You can always run `sys.path.append()` on the python source folder `pyfrost\src`.



## USAGE

Once the module is installed, all you need to do is to run `import pyfrost` inside Maya.

Please note that importing `pyfrost.main` may cause a small freeze as it's also loading the `bifrostGraph` plugin, which can take some time.

Example multiply node:

```
import pyfrost.main

# create a new graph node
graph = pyfrost.main.Graph("multiplyNode")

# get the input node and add a "value1" float output
root = graph["/input"]
root["value1"].add("output", "float")

# you can also just stack the full path with its attribute
graph["/input.value2"].add("output", "float")

# or you can keep the ports separated if you prefer to
graph["/input"]["value3"].add("output", "float")

# create a new multiply node
# Note: you can find the nodetype in the scriptEditor by creating a node manually,
↳ first.
# Then remove the "BifrostGraph," that shows up before the node type
mult = graph.create_node("Core::Math,multiply")

# to connect you can use the bitwise operator
root["value1"] >> mult["value1"]

# if a port doesn't exist on either the target or the source,
# it will try to create a new one with the type set to "auto"
root["value2"] >> mult["new_value"]

# you can also use the default method for connection
graph["/input"]["value3"].connect(mult["another_value"])

# now lets connect that to the output of the graph
mult["output"] >> graph["/output"]["result"]
```



## 4.1 pyfrost.api

### 4.1.1 pyfrost.api.maya

Maya Node API.

**author** Benoit Gielly <benoit.gielly@gmail.com>

The intention here is to build a node API specific to each DDC (here, Maya), so in the main.py we can call that and remove the *cmds* calls.

When a DCC is started, the relevant API is injected in the Main one.

---

**Note:** This is a Work in Progress for now and not in use.

---

#### Example

If we were to rewrite the *pyfrost.main.Graph.validate\_board* method, we could like that:

```
def __init__(self):
    self.api = Api()

def validate_board(self, name=None):
    node = self.api[name]
    if node.exists and node.type == "bifrostBoard":
        return name
    name = name if name else "bifrostGraph"
    board = api.create("bifrostBoard", name)
    return board.name
```

That way, all the main code remains clean of DCC commands. Obviously, each DCCs APIs must be implemented the same way for this to work.

**class MayaAPI**

Bases: object

Create a Maya API object.

**\_\_repr\_\_**()

Return repr(self).

**\_\_getitem\_\_**(key)

**create** (*nodetype*, *name=None*)

Create new node.

**get** (*name*)

Get existing node.

**class MayaNode** (*api*, *node*)

Bases: object

Get MayaNode object.

**\_\_init\_\_** (*api*, *node*)

Initialize self. See help(type(self)) for accurate signature.

**\_\_repr\_\_** (*\*args*, *\*\*kwargs*)

Return repr(self).

**\_\_str\_\_** ()

Return str(self).

**\_\_getitem\_\_** (*key*)

**type** ()

Get node type.

**rename** (*name*)

Rename node.

**class MayaAttr** (*node*, *name*)

Bases: object

Create a Maya Attribute class.

**\_\_init\_\_** (*node*, *name*)

Initialize self. See help(type(self)) for accurate signature.

**\_\_repr\_\_** ()

Return repr(self).

**\_\_str\_\_** ()

Return str(self).

**property plug**

Get plug.

**exists** ()

Check if node exists.

**property value**

Get & set attribute's value.

**property type**

Get & set attribute's type.

**add** (*type\_*, *\*\*kwargs*)

Add attribute on node.

**get** ()

Set node's attribute value.

**set** (*value*)

Set node's attribute value.

**connect** (*target*)

Connect current node to target.

**disconnect** (*target*)  
Disconnect current node from target.

## 4.2 pyfrost.compounds

### 4.2.1 pyfrost.compounds.paint\_delta

Create a paintDelta bifrost compound.

**author** Benoit Gielly <benoit.gielly@gmail.com>

Based on the compound created by Iker J. de los Mozos: <https://forums.autodesk.com/t5/bifrost-forum/paintdeltamap-compound/td-p/8972674>

```
class PainDeltaGraph (*args, **kwargs)
    Bases: pyfrost.main.Graph

    Custom Graph to paint deltas between 2 meshes.

    board_name = 'paintDelta'

    __init__ (*args, **kwargs)
        Initialize self. See help(type(self)) for accurate signature.

    create_graph (as_compound=False)
        Create paintDelta node graph.
```

## 4.3 pyfrost.main

Convenient class used to create bifrost node graphs in python.

**author** Benoit Gielly <benoit.gielly@gmail.com>

Bifrost VNN command documentation [https://help.autodesk.com/view/BIFROST/ENU/?guid=\\_\\_CommandsPython\\_index\\_html](https://help.autodesk.com/view/BIFROST/ENU/?guid=__CommandsPython_index_html)

```
class Graph (board=None)
    Bases: object

    Create a new bifrost graph object.

    board_name = 'default'

    __init__ (board=None)
        Initialize self. See help(type(self)) for accurate signature.

    __repr__ ()
        Return repr(self).

    __str__ ()
        Return str(self).

    __getitem__ (key)

    get (name)
        Get given string as node or attr.

    property name
        Get the name of the board.
```

**property nodes**

Get nodes at the board/root level.

**create\_node** (*type\_*, *parent*='/', *name*=None)

Create a new bifrost node in the graph.

**from\_json** (*path*)

Create a compound from JSON file.

**class Node** (*graph*, *parent*, *nodetype*=None, *name*=None)

Bases: object

Create Node object.

**\_\_init\_\_** (*graph*, *parent*, *nodetype*=None, *name*=None)

Initialize self. See help(type(self)) for accurate signature.

**\_\_repr\_\_** (*\*args*, *\*\*kwargs*)

Return repr(self).

**\_\_str\_\_** ()

Return str(self).

**\_\_getitem\_\_** (*key*)

**attr** (*value*)

Return the attribute class.

**node** (*value*)

Get a child of this node.

**get\_children** ()

Get children nodes.

**create\_node** (*type\_*, *name*=None)

Create a new node in the current compound.

**rename** (*name*)

Rename node.

---

**Note:** the *renameNode* option doesn't return the new name, so the only way to figure out the unique name is to query all nodes, rename, query again and diff... (cool stuff, right?!)

---

**property path**

Get node's path.

**property name**

Get node's name.

**property parent**

Get node's parent.

**property type**

Get node's type.

**property uuid**

Get node's UUID.

**set\_metadata** (*metadata*)

Set node metadata.



```
class Attribute (node_object, attribute=None)  
    Bases: object  
  
    Create Attribute object.  
  
    __init__ (node_object, attribute=None)  
        Initialize self. See help(type(self)) for accurate signature.  
  
    __str__ ()  
        Return str(self).  
  
    __repr__ ()  
        Return repr(self).  
  
    __rshift__ (plug)  
  
    __floordiv__ (plug)  
  
    property exists  
        Check if attribute exists.  
  
    property type  
        Get attribute type.  
  
    property value  
        Get and set attribute value.  
  
    add (direction, datatype='auto', value=None)  
        Add input plug on given node.  
  
    connect (target)  
        Connect plugs.  
  
    disconnect (target)  
        Disconnect plugs.
```



## RELEASE NOTES

### 5.1 1.1.1 (2020-12-16)

- Hotfix to the `from_json` method that still used the previous API version

### 5.2 1.1.0 (2020-12-16)

- Changed attribute access across all API

Removed some logic in the API that was too complicated. It was using the `__getattr__` method first to get nodes and attributes, but it ended requiring extra methods for when you wanted to pass nodes as strings like `graph.node("node")` or `node.attr("attribute")`. I have removed all that and decided to use `__getitem__` instead, which also prevents attribute clashing with builtin methods.

### 5.3 1.0.0 (2020-12-10)

- First public release



**MODULE INDEX**



---

**CHAPTER  
SEVEN**

---

**INDEX**





## PYTHON MODULE INDEX

### p

- `pyfrost`, [9](#)
- `pyfrost.api`, [9](#)
- `pyfrost.api.maya`, [9](#)
- `pyfrost.compounds`, [11](#)
- `pyfrost.compounds.paint_delta`, [11](#)
- `pyfrost.main`, [11](#)



## Symbols

\_\_floordiv\_\_ () (*Attribute method*), 13  
 \_\_getitem\_\_ () (*Graph method*), 11  
 \_\_getitem\_\_ () (*MayaAPI method*), 9  
 \_\_getitem\_\_ () (*MayaNode method*), 10  
 \_\_getitem\_\_ () (*Node method*), 12  
 \_\_init\_\_ () (*Attribute method*), 13  
 \_\_init\_\_ () (*Graph method*), 11  
 \_\_init\_\_ () (*MayaAttr method*), 10  
 \_\_init\_\_ () (*MayaNode method*), 10  
 \_\_init\_\_ () (*Node method*), 12  
 \_\_init\_\_ () (*PainDeltaGraph method*), 11  
 \_\_repr\_\_ () (*Attribute method*), 13  
 \_\_repr\_\_ () (*Graph method*), 11  
 \_\_repr\_\_ () (*MayaAPI method*), 9  
 \_\_repr\_\_ () (*MayaAttr method*), 10  
 \_\_repr\_\_ () (*MayaNode method*), 10  
 \_\_repr\_\_ () (*Node method*), 12  
 \_\_rshift\_\_ () (*Attribute method*), 13  
 \_\_str\_\_ () (*Attribute method*), 13  
 \_\_str\_\_ () (*Graph method*), 11  
 \_\_str\_\_ () (*MayaAttr method*), 10  
 \_\_str\_\_ () (*MayaNode method*), 10  
 \_\_str\_\_ () (*Node method*), 12

## A

add () (*Attribute method*), 13  
 add () (*MayaAttr method*), 10  
 attr () (*Node method*), 12  
 Attribute (*class in pyfrost.main*), 12

## B

board\_name (*Graph attribute*), 11  
 board\_name (*PainDeltaGraph attribute*), 11

## C

connect () (*Attribute method*), 13  
 connect () (*MayaAttr method*), 10  
 create () (*MayaAPI method*), 9  
 create\_graph () (*PainDeltaGraph method*), 11  
 create\_node () (*Graph method*), 12  
 create\_node () (*Node method*), 12

## D

disconnect () (*Attribute method*), 13  
 disconnect () (*MayaAttr method*), 10

## E

exists () (*Attribute property*), 13  
 exists () (*MayaAttr method*), 10

## F

from\_json () (*Graph method*), 12

## G

get () (*Graph method*), 11  
 get () (*MayaAPI method*), 10  
 get () (*MayaAttr method*), 10  
 get\_children () (*Node method*), 12  
 Graph (*class in pyfrost.main*), 11

## M

MayaAPI (*class in pyfrost.api.maya*), 9  
 MayaAttr (*class in pyfrost.api.maya*), 10  
 MayaNode (*class in pyfrost.api.maya*), 10  
 module  
     pyfrost, 9  
     pyfrost.api, 9  
     pyfrost.api.maya, 9  
     pyfrost.compounds, 11  
     pyfrost.compounds.paint\_delta, 11  
     pyfrost.main, 11

## N

name () (*Graph property*), 11  
 name () (*Node property*), 12  
 Node (*class in pyfrost.main*), 12  
 node () (*Node method*), 12  
 nodes () (*Graph property*), 11

## P

PainDeltaGraph (*class in pyfrost.compounds.paint\_delta*), 11  
 parent () (*Node property*), 12

`path()` (*Node property*), 12  
`plug()` (*MayaAttr property*), 10  
`pyfrost`  
    *module*, 9  
`pyfrost.api`  
    *module*, 9  
`pyfrost.api.maya`  
    *module*, 9  
`pyfrost.compounds`  
    *module*, 11  
`pyfrost.compounds.paint_delta`  
    *module*, 11  
`pyfrost.main`  
    *module*, 11

## R

`rename()` (*MayaNode method*), 10  
`rename()` (*Node method*), 12

## S

`set()` (*MayaAttr method*), 10  
`set_metadata()` (*Node method*), 12

## T

`type()` (*Attribute property*), 13  
`type()` (*MayaAttr property*), 10  
`type()` (*MayaNode method*), 10  
`type()` (*Node property*), 12

## U

`uuid()` (*Node property*), 12

## V

`value()` (*Attribute property*), 13  
`value()` (*MayaAttr property*), 10