

---

# DefTree Documentation

*Release 1.0.2*

**mattias.hedberg**

**Feb 28, 2018**



---

## Contents

---

<b>1</b>	<b>Element</b>	<b>1</b>
<b>2</b>	<b>Attribute</b>	<b>3</b>
<b>3</b>	<b>DefTree</b>	<b>5</b>
<b>4</b>	<b>Helpers</b>	<b>7</b>
<b>5</b>	<b>Using DefTree</b>	<b>9</b>
5.1	Parsing Defold Documents . . . . .	9
5.2	Finding interesting elements . . . . .	10
5.3	Modifying existing scenes . . . . .	10
<b>6</b>	<b>Changelog</b>	<b>11</b>
6.1	1.0.2 . . . . .	11
6.2	1.0.1 . . . . .	11
6.3	0.2.0 . . . . .	12
6.4	0.1.1 . . . . .	12
6.5	0.1.0 . . . . .	13
<b>7</b>	<b>Contributing to DefTree</b>	<b>15</b>
7.1	Bug fixes, feature additions, etc. . . . .	15
7.2	Guidelines . . . . .	15
7.3	Reporting Issues . . . . .	15
<b>8</b>	<b>DefTree</b>	<b>17</b>
8.1	Installation . . . . .	17
8.2	Old Versions . . . . .	17



**class** `deftree.Element` (*name*)  
Element class. This class defines the Element interface

**add\_attribute** (*name*, *value*)  
Creates an *Attribute* instance with name and value as a child to self.

**add\_element** (*name*)  
Creates an *Element* instance with name as a child to self.

**append** (*item*)  
Inserts the item at the end of this element's internal list of children. Raises *TypeError* if item is not a *Element* or *Attribute*

**attributes** ([*name*, *value*])  
Creates a tree iterator with the current element as the root. The iterator iterates over this element. Only *Attributes*. Name and value are optional and used for filters

**clear** ()  
Resets an element. This function removes all children, clears all attributes

**copy** ()  
Returns a deep copy of the current *Element*.

**elements** ([*name*])  
Creates a tree iterator with the current element as the root. The iterator iterates over this element and all elements below it, in document (depth first) order. Only *Element* whose name equals name are returned from the iterator

**get\_attribute** (*name*[, *value*])  
Returns the first *Attribute* instance whose name matches name and if value is not None whose value equal value. If none is found it returns None.

**get\_element** (*name*)  
Returns the first *Element* whose name matches name, if none is found returns None.

**get\_parent** ()  
Returns the parent of the current *Element*

**index** (*item*)

Returns the index of the item in this element, raises *ValueError* if not found.

**insert** (*index*, *item*)

Inserts the item at the given position in this element. Raises *TypeError* if item is not a *Element* or *Attribute*

**iter** ()

Creates a tree iterator with the current element as the root. The iterator iterates over this element and all elements below it, in document (depth first) order. Both *Element* and *Attribute* are returned from the iterator.

**iter\_elements** (*name* [, *value* ])

Creates a tree iterator with the current element as the root. The iterator iterates over this element and all elements below it, in document (depth first) order. Only *Element* are returned from the iterator.

**remove** (*child*)

Removes child from the element. Compares on instance identity not name. Raises *TypeError* if child is not a *Element* or *Attribute*

**set\_attribute** (*name*, *value*)

Sets the first *Attribute* with name to value.

## CHAPTER 2

---

### Attribute

---

```
class deftree.Attribute (parent, name, value)  
    Attribute class. This class defines the Attribute interface.  
  
    get_parent ()  
        Returns the parent element of the attribute.
```





**class** `deftree.DefTree`

DefTree class. This class represents an entire element hierarchy.

**dump** ()

Write the the DefTree structure to sys.stdout. This function should be used for debugging only.

**from\_string** (*text* [, *parser* ])

Parses a Defold document section from a string constant which it returns. *parser* is an optional parser instance. If not given the standard parser is used. Returns the root of *DefTree*.

**get\_document\_path** ()

Returns the path to the parsed document.

**get\_root** ()

Returns the root *Element*

**parse** (*source* [, *parser* ])

Parses a Defold document into a *DefTree* which it returns. *source* is a *file\_path*. *parser* is an optional parser instance. If not given the standard parser is used.

**write** (*file\_path*)

Writes the element tree to a file, as plain text. *file\_path* needs to be a path.



## CHAPTER 4

---

### Helpers

---

`deftree.parse(source)`

Parses a Defold document into a DefTree which it returns. *source* is a file\_path. *parser* is an optional parser instance. If not given the standard parser is used.

`deftree.from_string(text[, parser])`

Parses a Defold document section from a string constant which it returns. *parser* is an optional parser instance. If not given the standard parser is used. Returns the root of *DefTree*.

`deftree.to_string(element[, parser])`

Generates a string representation of the Element, including all children. *element* is a *Element* instance.

`deftree.dump(element[, parser])`

Write element tree or element structure to sys.stdout. This function should be used for debugging only. *element* is either an *DefTree*, or *Element*.

`deftree.validate(string, path_or_string[, verbose])`

Verifies that a document in string format equals a document in path\_or\_string. If Verbose is True it echoes the result. This function should be used for debugging only. Returns a bool representing the result



# CHAPTER 5

## Using DefTree

If you are not familiar with Defold files this is how the syntax looks, it is in a [Google Protobuf](#) format.

```
elementname {
  attributename: attributevalue
  position {
    x: 0.0
    y: 0.0
    z: 0.0
  }
  type: TYPE_BOX
  blend_mode: BLEND_MODE_ALPHA
  texture: "atlas/logo"
  id: "logo"
}
```

## 5.1 Parsing Defold Documents

Parsing from a file is done by calling the parse method

```
import deftree
tree = deftree.parse(path)  # parse the document into a DefTree
root = tree.get_root()     # returns the root from the tree
```

Or alternatively we can first create a DefTree instance

```
tree = deftree.Deftree()
root = tree.parse(path)
```

We can also parse a document from a string

```
tree = deftree.from_string(document_as_string)  # parse the document into a DefTree
root = tree.get_root()                         # returns the root from the tree
```

## 5.2 Finding interesting elements

Element has some useful methods that help iterate recursively over all the sub-tree below it (its children, their children, and so on). For example, `Element.iter()`:

```
for child in root.iter():
    print(child.name)
```

We can also iterate only elements by calling `Element.iter_elements()`

```
for child in root.iter_elements():
    print(child.name)

for child in root.iter_elements("nodes"): # iter_elements also supports filtering on_
↪name
    print(child.name)
```

`Element.get_attribute()` finds the first attributes with the given name in that element.

```
attribute = element.get_attribute("id")
```

## 5.3 Modifying existing scenes

DefTree provides a simple way to edit Defold documents and write them to files. The `DefTree.write()` method serves this purpose. Once created, an `Element` object may be manipulated by directly changing its fields, as well as adding new children (for example with `Element.insert()`).

Let's say we want to find all box nodes in a gui and change its layers.

```
for element in root.iter_elements("nodes")
    if element.get_attribute("type") == "TYPE_BOX":
        element.set_attribute("layer", 'new_layer')
```

We can also add new attributes and elements all together.

```
new_element = root.add_element("layers")
new_element.add_attribute("name", 'new_layer')
```

DefTree Attribute of number types supports basic math functions directly

```
new_element = root.get_element("position")
attribute = new_element.get_attribute("x")
attribute += 10
```

We will probably then overwrite the file

```
tree.write(tree.get_document_path())
```

### 6.1 1.0.2

#### 6.1.1 Changed

- How DefTree determines if a string is a string, int or float, fix for bigger numbers with science annotation
- 

### 6.2 1.0.1

#### 6.2.1 Added

- Added `Element.add_element(name)`
- Added `Element.add_attribute(name, value)`
- Added `Element.set_attribute(name, value)`
- Added `Element.elements()` - for getting top level elements of `Element`
- Added `Element.attribute()` - for getting top level attribute of `Element`
- Exposed `deftree.dump` and `deftree.validate` in the documentation
- Added `DefTree.get_document_path()` to get the path of the document that was parsed
- Attribute are now sub classed into different types this to make it easier when editing values as Defold is picky

#### 6.2.2 Changed

- `Element.iter_all()` is now `Element.iter()`
  - `Element.iter_find_elements(name)` is now `Element.iter_elements(name)`
-

- Changed how attributes reports their value. They should now be easier to work with, without any need add quotationmarks and such.

### 6.2.3 Removed

- Removed SubElement() factory, now use element.add\_element()
  - Removed Element.iter\_attributes()
  - Removed Element.iter\_find\_attributes()
  - Removed NaiveDefParser as it was obsolete and inferior
  - Removed Example folder
- 

## 6.3 0.2.0

### 6.3.1 Added

- Raises ParseError when reading invalid documents

### 6.3.2 Changed

- Updated docstrings to be easier to read.
- Refactored internal usage of a level variable to track how deep the item were in the tree

### 6.3.3 Removed

- Removed Element.add(), use Element.append() Element.insert()
  - Removed Element.items(), use Element.iter\_all()
- 

## 6.4 0.1.1

### 6.4.1 Added

- Licence to github repository
- Setup files for PyPi to github repository
- Example usage
- Unittesting with [unittest](#)
- Coverage exclusion for usage with [Coverage.py](#)
- Using `__all__` to define public api, in case of wild import



### 6.4.2 Changed

- Elements `__setitem__` raises exception on invalid types
  - Elements `__next__` implementation was broken
  - `serialize()` is now a class method
- 

## 6.5 0.1.0

### 6.5.1 Added

- First release of DefTree



---

## Contributing to DefTree

---

Bug fixes, feature additions, tests, documentation and more can be contributed via [issues](#) and/or [pull requests](#). All contributions are welcome.

### 7.1 Bug fixes, feature additions, etc.

Please send a pull request to the master branch. Please include [documentation](#) and [tests](#) for new or changed features. Tests or documentation without bug fixes or feature additions are welcome too.

- Fork the DefTree repository.
- Create a branch from master.
- Develop bug fixes, features, tests, etc.
- Run the test suite.
- Create a pull request to pull the changes from your branch to the DefTree master.

### 7.2 Guidelines

- Separate code commits from reformatting commits.
- Provide tests for any newly added code.
- Follow PEP8.

### 7.3 Reporting Issues

When reporting issues, please include code that reproduces the issue. The best reproductions are self-contained scripts with minimal dependencies.



DefTree is a python module modify [Defold](#) documents, it's first implementation was inspired by the `xml.ElementTree` library.

It reads any Defold document into a object tree hierarchy, the module have three main concepts

1. DefTree represents the whole Defold document as a tree and
2. Element represents a single node or block in this tree and
3. Attribute represent a name value pair

## 8.1 Installation

---

**Note:** DefTree is only supported by python  $\geq 3.3.0$

---

DefTree is a native python implementation and thus should work under the most common platforms that supports python. The package is distributed in the wheel format and is easily installed with pip.

```
pip install deftree
```

## 8.2 Old Versions

You can download old distributions from [PyPI](#).



## A

`add_attribute()` (deftree.Element method), 1  
`add_element()` (deftree.Element method), 1  
`append()` (deftree.Element method), 1  
`Attribute` (class in deftree), 3  
`attributes()` (deftree.Element method), 1

## C

`clear()` (deftree.Element method), 1  
`copy()` (deftree.Element method), 1

## D

`DefTree` (class in deftree), 5  
`dump()` (deftree.DefTree method), 5  
`dump()` (in module deftree), 7

## E

`Element` (class in deftree), 1  
`elements()` (deftree.Element method), 1

## F

`from_string()` (deftree.DefTree method), 5  
`from_string()` (in module deftree), 7

## G

`get_attribute()` (deftree.Element method), 1  
`get_document_path()` (deftree.DefTree method), 5  
`get_element()` (deftree.Element method), 1  
`get_parent()` (deftree.Attribute method), 3  
`get_parent()` (deftree.Element method), 1  
`get_root()` (deftree.DefTree method), 5

## I

`index()` (deftree.Element method), 1  
`insert()` (deftree.Element method), 2  
`iter()` (deftree.Element method), 2  
`iter_elements()` (deftree.Element method), 2

## P

`parse()` (deftree.DefTree method), 5  
`parse()` (in module deftree), 7

## R

`remove()` (deftree.Element method), 2

## S

`set_attribute()` (deftree.Element method), 2

## T

`to_string()` (in module deftree), 7

## V

`validate()` (in module deftree), 7

## W

`write()` (deftree.DefTree method), 5