# DefTree Documentation

***Release 0.2.0***

**mattias.hedberg**

**Feb 28, 2018**

# Contents

**1 Element** **1**

**2 Attribute** **3**

**3 DefTree** **5**

**4 Helpers** **7**

**5 Using DefTree** **9**
- 5.1 Example 1: Parsing Defold Documents . . . . . 9
- 5.2 Example 2: Finding interesting elements . . . . . 9
- 5.3 Example 3: Modifying existing scenes . . . . . 10
- 5.4 More Examples . . . . . 10

**6 Changelog** **11**
- 6.1 0.2.0 . . . . . 11
- 6.2 0.1.1 . . . . . 11
- 6.3 0.1.0 . . . . . 12

**7 Contributing to DefTree** **13**
- 7.1 Bug fixes, feature additions, etc. . . . . . 13
- 7.2 Guidelines . . . . . 13
- 7.3 Reporting Issues . . . . . 13

**8 DefTree** **15**
- 8.1 Installation . . . . . 15
- 8.2 Old Versions . . . . . 15

# CHAPTER 1

# Element

**class** deftree.**Element**(*name*)

Element class. This class defines the Element interface

**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`

**clear**()

Resets an element. This function removes all children, clears all attributes

**copy**()

Returns a deep copy of the current `Element`.

**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_all**()

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_attributes**()

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 `Attributes` are returned from the iterator.

**`iter_elements`**()
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.

**`iter_find_attributes`**(*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 `Attributes` whose name equals name, and if value is not None whose value equal value are returned from the iterator

**`iter_find_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

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

# 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.

# CHAPTER 3

# DefTree

**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_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.`**`SubElement`**(*parent*, *name*)

SubElement factory which creates an element instance with *name*, and appends it to an existing parent.

`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.

# CHAPTER 5

# Using DefTree

If you are not familiar with Defold files this is how the syntax looks, it is the 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 Example 1: Parsing Defold Documents

We can import this data by reading from a file:

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

## 5.2 Example 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_all():

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

Element.get_attribute() finds the first attributes with the given name in that element. This will return the attribute, which you can then get the parent from thus finding a particular node. For example:

```
attribute = element.get_attribute("id", '"logo"')
logo_node = attribute.get_parent()
```

## 5.3 Example 3: Modifying existing scenes

DefTree provides a simple way to build 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 (such as Attribute.value), as well as adding new children (for example with Element.append()).

Let's say we want to add 10 to all x value in a scene

```
for child in root.iter_find_attributes("x"):
    child.value =+ 10.0
```

The SubElement() function also provides a convenient way to create new sub-elements for a given element, adding new attributes to that is easy.

```
new_parent = deftree.SubElement(root, "layers")
deftree.Attribute(new_parent, "name", '"new_layer"')
```

## 5.4 More Examples

There are a lot more in depth examples in the folder examples of the repository

# CHAPTER 6

# Changelog

## 6.1 0.2.0

### 6.1.1 Added

- Raises ParseError when reading invalid documents

### 6.1.2 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

### 6.1.3 Removed

- Removed Element.add(), use Element.append() Element.insert()
- Removed Element.items(), use Element.iter_all()

## 6.2 0.1.1

### 6.2.1 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

### 6.2.2 Changed

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

## 6.3 0.1.0

### 6.3.1 Added

- First release of DefTree

# CHAPTER 7

# 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.

# CHAPTER 8

# DefTree

DefTree is a python module modify Defold documents, it is 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 >= 3.0.0

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

```
pip install deftree
```

## 8.2 Old Versions

You can download old distributions from PyPI.

# Index

A

append() (deftree.Element method), 1
Attribute (class in deftree), 3

C

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

D

DefTree (class in deftree), 5
dump() (deftree.DefTree method), 5

E

Element (class in deftree), 1

F

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

G

get_attribute() (deftree.Element method), 1
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), 1
iter_all() (deftree.Element method), 1
iter_attributes() (deftree.Element method), 1
iter_elements() (deftree.Element method), 2
iter_find_attributes() (deftree.Element method), 2
iter_find_elements() (deftree.Element method), 2

P

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

R

remove() (deftree.Element method), 2

S

SubElement() (in module deftree), 7

T

to_string() (in module deftree), 7

W

write() (deftree.DefTree method), 5