
DefTree Documentation

Release 2.1.3

mattias.hedberg

Nov 08, 2018

Contents

1	Element	1
2	Attribute	3
3	DefTree	5
4	Helpers	7
5	Using DefTree	9
5.1	Parsing Defold Documents	9
5.2	Finding interesting elements	10
5.3	Modifying existing scenes	10
6	Design	11
6.1	Defold Value vs Python Value	11
7	Contributing	13
7.1	Bug fixes, feature additions, etc.	13
7.2	Guidelines	13
7.3	Reporting Issues	13
8	Changelog	15
8.1	2.1.3	15
8.2	2.1.1	15
8.3	2.1.0	15
8.4	2.0.0	16
8.5	1.1.1	16
8.6	1.1.0	16
8.7	1.0.2	17
8.8	1.0.1	17
8.9	0.2.0	17
8.10	0.1.1	18
8.11	0.1.0	18
9	DefTree 2.1.3	19
9.1	Installation	19
9.2	Old Versions	19


```

class deftree.Element (name: Union[bytes, str])
    Element class. This class defines the Element interface

    add_attribute (name: Union[bytes, str], value: Union[bytes, str, float, int, bool]) →
        Union[deftree.DefTreeBool, deftree.DefTreeEnum, deftree.DefTreeFloat,
        deftree.DefTreeInt, deftree.DefTreeString]
        Creates an Attribute instance with name and value as a child to self.

    add_element (name: Union[bytes, str]) → deftree.Element
        Creates an Element instance with name as a child to self.

    append (item: Union[Element, Attribute])
        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])
        Iterates over the current element and returns all attributes. Only Attributes. Name and value are
        optional and used for filters.

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

    copy () → deftree.Element
        Returns a deep copy of the current Element.

    elements ([name])
        Iterates over the current element and returns all elements. If the optional argument name is not None only
        Element with a name equal to name is returned.

    get_attribute (name[, value])
        Returns the first Attribute instance whose name matches name and if value is not None whose value
        equal value. If no matching attribute is found it returns None.

    get_element (name: Union[bytes, str]) → deftree.Element
        Returns the first Element whose name matches name, if none is found returns None.

```

get_parent () → deftree.Element

Returns the parent of the current *Element*

index (item: Union[Element, Attribute]) → int

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

insert (index: int, item: Union[Element, Attribute])

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

iter () → Iterator[Union[Element, Attribute]]

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 ([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. If the optional argument name is not None only *Attribute* with a name equal to name is returned.

iter_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. If the optional argument name is not None only *Element* with a name equal to name is returned.

remove (child: Union[Element, Attribute])

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

set_attribute (name: Union[bytes, str], value: Union[bytes, str, float, int, bool])

Sets the first *Attribute* with name to value.

class `deftree.Attribute` (*parent: deftree.Element, name: Union[bytes, str], value*)

Attribute class. This class defines the Attribute interface.

get_parent () → `deftree.Element`

Returns the parent element of the attribute.

name

The name of the attribute, used to set and get the name

value

The value of the attribute, used to set and get the attributes value

class `deftree.DefTree`

DefTree class. This class represents an entire element hierarchy.

dump ()

Writes 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 () → str

Returns the path to the parsed document.

get_root () → `deftree.Element`

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. uses the parsed file as a default

CHAPTER 4

Helpers

`deftree.parse` (*source*: *Union[bytes, str]*) → *deftree.DefTree*
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.is_element` (*item*: *deftree.Element*) → bool
Returns True if the item is an *Element* else returns False

`deftree.is_attribute` (*item*: *deftree.Attribute*) → bool
Returns True if the item is an *Attribute* else returns False

`deftree.to_string` (*element*[, *parser*])
Generates a string representation of the *Element*, including all children. *element* is a *Element* instance.

`deftree.dump` (*element*[, *parser*])
Writes the 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()`).

E.g. if 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 Attributes that are of number types support basic math functions directly

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

You can either use the set value if you are getting, or you can use its `.value` property

```
attribute = element.get_attribute("layer")
attribute.value = "new_layer"
```

We will probably then overwrite the file

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

Here I would like to go over some important details concerning implementation that may help when working with DefTree.

6.1 Defold Value vs Python Value

To simplify working with attributes I decided to split how the value looks for Defold and how it looks for python. Not only does this simplify working with attributes it also enables us to do some sanity checking to ensure that we do not set a value that was an int to a float, as this would make the file corrupt for the Defold editor.

Defold will always enclose a string within two quotes like `"/main/defold.png"`. To make it easier for us to work with it DefTree reports this as `/main/defold.png`, i.e. without the quotes. As an example, let us assume we have a file that looks as follows:

```
nodes {  
  id: "sprite"  
  blend_mode: BLEND_MODE_ALPHA  
  inherit_alpha: true  
}
```

This enables the user to do this:

```
tree = root.parse(my_atlas)  
root.get_root()  
  
for ele in root.get_element("nodes"):  
    node_id = ele.get_attribute("id")  
    alpha = ele.get_attribute("inherit_alpha")  
    if node_id == "sprite" and alpha:  
        ...
```

in contrast to:

```
tree = root.parse(my_atlas)
root.get_root()

for ele in root.get_element("nodes"):
    node_id= ele.get_attribute("id")
    alpha = ele.get_attribute("inherit_alpha")
    if node_id == "/"sprite/" and alpha == "true": # or '"sprite"'
        ...
```

The former is a lot more readable and not as error prone, as I see it.

6.1.1 Attribute types

The attribute's type is decided on creation and follow the logic below:

If the value is of type(bool) or a string equal to “true” or “false” it is considered a bool.

If the value consists of only capital letters and underscore (regex'd against `[A-Z_]+`) it is considered an enum.

If the value is of type(float) or it looks like a float (regex'd against `[-\d]+\.\d+[eE-]+\d+|[-\d]+\.\d+`) it is considered a float.

If the value is of type(int) or can be converted with `int()` it is considered an int.

Else it is considered a string.

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 will reproduce the issue. The best reproductions are self-contained scripts with minimal dependencies.

8.1 2.1.3

8.1.1 Changed

- #6 Science annotation are now serialized properly

8.2 2.1.1

8.2.1 Added

- Added split and rsplit for DefTreeString
- Added __contains__ for DefTreeString
- #5 Added type hints for arguments

8.3 2.1.0

8.3.1 Added

- Added type hints for return types, helps IDEs with autocomplete

8.3.2 Changed

- DefTree.write() argument file_path is now optional, uses the parsers file path as default

8.4 2.0.0

8.4.1 Added

- Added the following functions for the DefTreeString implementation: `endswith`, `startswith`, `strip`, `rstrip`, `count`, `index`, `rindex`, `replace`
- Added Attribute implementation for `len()`

8.4.2 Changed

- `repr()` for Elements and Attributes now returns a proper formatted representations of the object
- `__str__` on Attributes removed, now defaults back to `repr()`
- uses python standard library `copy` for getting a copy of a elements

8.4.3 Removed

- Removed `Element._set_attribute_name()`, name of attributes should be changed with `Attribute.name`
-

8.5 1.1.1

8.5.1 Changed

- Fixed a bug where a negative number would be evaluated as a string
-

8.6 1.1.0

8.6.1 Added

- Added `Element.iter_attributes` to iterate over the elements and its children's elements attributes

8.6.2 Changed

- Only imports `re.compile` from `re` instead of the whole of `re`
 - The string value of an attribute can now be get with `Attribute.string`
 - The `Attribute.value` and the value `Attribute()` returns should be the same
 - Now reports the python value when calling the `__str__` method instead of the defold value
 - `is_element` and `is_attribute` are no longer flagged as internal
 - improved type checking when setting attribute values
-

8.7 1.0.2

8.7.1 Changed

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

8.8 1.0.1

8.8.1 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 `defree.dump` and `defree.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

8.8.2 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.

8.8.3 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
-

8.9 0.2.0

8.9.1 Added

- Raises `ParseError` when reading invalid documents

8.9.2 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

8.9.3 Removed

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

8.10 0.1.1

8.10.1 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

8.10.2 Changed

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

8.11 0.1.0

8.11.1 Added

- First release of DefTree

CHAPTER 9

DefTree 2.1.3

DefTree is a python module for modifying [Defold](#) documents. The first implementation was inspired by the `xml.ElementTree` library.

DefTree reads any Defold document into an object tree hierarchy and follow the these three main concepts

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

9.1 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
```

You need to install the backport of the standard library [typing](#) module if you are running Python versions older than 3.5

```
pip install typing
```

9.2 Old Versions

Old distributions may be accessed via [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), 2
insert() (deftree.Element method), 2
is_attribute() (in module deftree), 7
is_element() (in module deftree), 7
iter() (deftree.Element method), 2
iter_attributes() (deftree.Element method), 2

iter_elements() (deftree.Element method), 2

N

name (deftree.Attribute attribute), 3

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
value (deftree.Attribute attribute), 3

W

write() (deftree.DefTree method), 5