
xml_python

Chris Norman

Jan 07, 2021

CONTENTS:

1	Installation	3
2	Installing Using pip	5
3	Install Using Git	7
4	Running Tests	9
5	Building Documentation	11
6	xml_python	13
6.1	xml_python package	13
7	Indices and tables	17
	Python Module Index	19
	Index	21

This module allows you to create Python objects from XML strings.

INSTALLATION

INSTALLING USING PIP

It is recommended that you install `xml_python` using `pip`:

```
pip install xml_python
```


INSTALL USING GIT

Alternatively, you could install using git:

```
git clone https://github.com/chrisnorman7/xml_python.git
cd xml_python
python setup.py install
```


RUNNING TESTS

To run the tests, you will need to install `pytest`:

```
pip install pytest
```

Then to run the tests:

```
py.test
```


BUILDING DOCUMENTATION

You can always find the most up to date version of the docs on [Read the Docs](#), but you can also build them yourself:

```
pip install -Ur docs/requirements.txt
python setup.py build_sphinx
```


XML_PYTHON

6.1 xml_python package

6.1.1 Module contents

The xml_python library.

A library for converting XML into python objects.

Provides the Builder class.

```
class xml_python.Builder (maker: Callable[[Optional[InputObjectType],  
xml.etree.ElementTree.Element], OutputObjectType], name: Op-  
tional[str] = None, parsers: Dict[str, Callable[[OutputObjectType,  
xml.etree.ElementTree.Element], None]] = NOTHING, builders: Dict[str,  
Builder] = NOTHING)
```

Bases: Generic[xml_python.InputObjectType, xml_python.OutputObjectType]

A builder for returning python objects from xml Element instances.

Given a single node and a input object, a builder can transform the input object according to the data found in the provided element.

To parse a tag, use the `parse_element()` method.

Variables

- **maker** – The method which will make an initial object for this builder to work on.
- **name** – A name to differentiate a builder when debugging.
- **~Builder>parsers** – A dictionary of parser functions mapped to tag names.
- **builders** – A dictionary a sub builders, mapped to tag names.

add_builder (tag: str, builder: xml_python.Builder) → None

Add a sub builder to this builder.

In the same way that parsers know how to handle a single tag, sub builders can handle many tags recursively.

Parameters

- **tag** – The name of the top level tag this sub builder knows how to handle.
- **builder** – The builder to add.

add_parser (tag: str, func: Callable[[OutputObjectType, xml.etree.ElementTree.Element], None]) → None

Add a parser to this builder.

Parameters

- **tag** – The name of the tag that this parser knows how to handle.
- **func** – The function which will do the actual parsing.

build (*input_object: Optional[InputObjectType], element: xml.etree.ElementTree.Element*) → OutputObjectType
Make the initial object, then handle the element.

If you are looking to build an object from an xml element, this is likely the method you want.

Parameters

- **input_object** – An optional input object for the provided elements to work on.
- **element** – The element to handle.

builders: Dict[str, xml_python.Builder]

handle_elements (*subject: OutputObjectType, elements: List[xml.etree.ElementTree.Element]*) → None
Handle each of the given elements.

This method is usually called by the `make_and_handle()` method.

Parameters

- **subject** – The object to manipulate with the given elements.
- **elements** – The elements to work through.

handle_file (*fileobj: IO[str]*) → OutputObjectType
Handle a file-like object.

Parameters fileobj – The file-like object to read XML from.

handle_filename (*filename: str*) → OutputObjectType
Return an element made from a file with the given name.

Parameters filename – The name of the file to load.

handle_string (*xml: str*) → OutputObjectType
Parse and handle an element from a string.

Parameters xml – The xml string to parse.

maker: Callable[[Optional[InputObjectType], xml.etree.ElementTree.Element], OutputObj]

name: Optional[str]

parser (*tag: str*) → Callable[[Callable[[OutputObjectType, xml.etree.ElementTree.Element], None]], Callable[[OutputObjectType, xml.etree.ElementTree.Element], None]]
Add a new parser to this builder.

Parsers work on the name on a tag. So if you wish to work on the XML `<title>Hello, title</title>`, you need a parser that knows how to handle the `title` tag.

Parameters tag – The tag name this parser will handle.

parsers: Dict[str, Callable[[OutputObjectType, xml.etree.ElementTree.Element], None]]

exception xml_python.BuilderError

Bases: Exception

Base exception.

class xml_python.NoneType

Bases: object

exception `xml_python.UnhandledElement`

Bases: `xml_python.BuilderError`

No such parser has been defined.

INDICES AND TABLES

- `genindex`
- `modindex`
- `search`

PYTHON MODULE INDEX

X

`xml_python`, [13](#)

A

`add_builder()` (*xml_python.Builder method*), 13
`add_parser()` (*xml_python.Builder method*), 13

B

`build()` (*xml_python.Builder method*), 14
`Builder` (*class in xml_python*), 13
`BuilderError`, 14
`builders` (*xml_python.Builder attribute*), 14

H

`handle_elements()` (*xml_python.Builder method*),
 14
`handle_file()` (*xml_python.Builder method*), 14
`handle_filename()` (*xml_python.Builder method*),
 14
`handle_string()` (*xml_python.Builder method*), 14

M

`maker` (*xml_python.Builder attribute*), 14
`module`
 `xml_python`, 13

N

`name` (*xml_python.Builder attribute*), 14
`NoneType` (*class in xml_python*), 14

P

`parser()` (*xml_python.Builder method*), 14
`parsers` (*xml_python.Builder attribute*), 14

U

`UnhandledElement`, 15

X

`xml_python`
 `module`, 13