martinblech / **xmltodict** Public

Code Issues Pull requests 1 Actions Projects Wiki Security and quality

2 Branches 40 Tags Go to file Go to file Code

martinblech docs(readme): clarify whitespace preservation with pretty printing

77b55aa · 2 months ago

.github	Create FUNDING.yml	2 months ago
tests	fix(unparse): add bytes_errors p…	3 months ago
.gitignore	ignore MANIFEST	14 years ago
.pre-commit-config.yaml	chore: commit message checks	8 months ago
.release-please-manifest.j…	chore(master): release 1.0.4	3 months ago
AGENTS.md	docs: create AGENTS.md for co…	8 months ago
CHANGELOG.md	chore(master): release 1.0.4	3 months ago
CONTRIBUTING.md	docs: update CONTRIBUTING.md	8 months ago
LICENSE	updated (c) notice to acknowled…	14 years ago
MANIFEST.in	build: no need for README.md i…	7 months ago
README.md	docs(readme): clarify whitespace…	2 months ago
SECURITY.md	docs: add SECURITY.md	8 months ago
pyproject.toml	chore(master): release 1.0.4	3 months ago
release-please-config.json	build: migrate packaging to pypr…	8 months ago
tox.ini	build: migrate packaging to pypr…	8 months ago
xmltodict.py	fix(unparse): add bytes_errors p…	3 months ago

README Contributing MIT license Security

xmltodict

`xmltodict` is a Python module that makes working with XML feel like you are working with JSON, as in this "spec":


```
>>> print(json.dumps(xmltodict.parse("""
...  <mydocument has="an attribute">
...    <and>
...      <many>elements</many>
...      <many>more elements</many>
...    </and>
...    <plus a="complex">
...      element as well
...    </plus>
...  </mydocument>
...  """), indent=4))
{
    "mydocument": {
        "@has": "an attribute",
        "and": {
            "many": [
                "elements",
                "more elements"
            ]
        },
        "plus": {
            "@a": "complex",
            "#text": "element as well"
        }
    }
}
```

Namespace support

By default, `xmltodict` does no XML namespace processing (it just treats namespace declarations as regular node attributes), but passing `process_namespaces=True` will make it expand namespaces for you:

```
>>> xml = """
... <root xmlns="http://defaultns.com/"
...       xmlns:a="http://a.com/"
...       xmlns:b="http://b.com/">
...   <x>1</x>
...   <a:y>2</a:y>
...   <b:z>3</b:z>
... </root>
... """
>>> xmltodict.parse(xml, process_namespaces=True) == {
...     'http://defaultns.com/:root': {
...         'http://defaultns.com/:x': '1',
...         'http://a.com/:y': '2',
```

```
...         'http://b.com/:z': '3',
...     }
... }
True
```

It also lets you collapse certain namespaces to shorthand prefixes, or skip them altogether:

```
>>> namespaces = {
...     'http://defaultns.com/': None, # skip this namespace
...     'http://a.com/': 'ns_a', # collapse "http://a.com/" -> "ns_a"
... }
>>> xmltodict.parse(xml, process_namespaces=True, namespaces=namespaces) == {
...     'root': {
...         'x': '1',
...         'ns_a:y': '2',
...         'http://b.com/:z': '3',
...     },
... }
True
```

Streaming mode

`xmltodict` is very fast (Expat-based) and has a streaming mode with a small memory footprint, suitable for big XML dumps like Discogs or Wikipedia:

```
>>> def handle_artist(_, artist):
...     print(artist['name'])
...     return True
>>>
>>> xmltodict.parse(GzipFile('discogs_artists.xml.gz'),
...     item_depth=2, item_callback=handle_artist)
A Perfect Circle
Fantômas
King Crimson
Chris Potter
...
```

It can also be used from the command line to pipe objects to a script like this:

```
import sys, marshal
while True:
    _, article = marshal.load(sys.stdin)
    print(article['title'])
```

```
$ bunzip2 enwiki-pages-articles.xml.bz2 | xmltodict.py 2 | myscript.py
AccessibleComputing
Anarchism
AfghanistanHistory
```

```
AfghanistanGeography
AfghanistanPeople
AfghanistanCommunications
Autism
...
```

Or just cache the dicts so you don't have to parse that big XML file again. You do this only once:

```
$ bunzip2 enwiki-pages-articles.xml.bz2 | xmltodict.py 2 | gzip > enwiki.dicts.gz
```

And you reuse the dicts with every script that needs them:

```
$ gunzip enwiki.dicts.gz | script1.py
$ gunzip enwiki.dicts.gz | script2.py
...
```

Roundtripping

You can also convert in the other direction, using the `unparse()` method:

```
>>> mydict = {
...     'response': {
...             'status': 'good',
...             'last_updated': '2014-02-16T23:10:12Z',
...     }
... }
>>> print(unparse(mydict, pretty=True))
<?xml version="1.0" encoding="utf-8"?>
<response>
    <status>good</status>
    <last_updated>2014-02-16T23:10:12Z</last_updated>
</response>
```

Text values for nodes can be specified with the `cdata_key` key in the python dict, while node properties can be specified with the `attr_prefix` prefixed to the key name in the python dict. The default value for `attr_prefix` is `@` and the default value for `cdata_key` is `#text`.

```
>>> import xmltodict
>>>
>>> mydict = {
...     'text': {
...         '@color':'red',
...         '@stroke':'2',
...         '#text':'This is a test'
...     }
... }
>>> print(xmltodict.unparse(mydict, pretty=True))
```

```
<?xml version="1.0" encoding="utf-8"?>
<text stroke="2" color="red">This is a test</text>
```

Lists that are specified under a key in a dictionary use the key as a tag for each item. But if a list does have a parent key, for example if a list exists inside another list, it does not have a tag to use and the items are converted to a string as shown in the example below. To give tags to nested lists, use the `expand_iter` keyword argument to provide a tag as demonstrated below. Note that using `expand_iter` will break roundtripping.

```
>>> mydict = {
...     "line": {
...         "points": [
...             [1, 5],
...             [2, 6],
...         ]
...     }
... }
>>> print(xmltodict.unparse(mydict, pretty=True))
<?xml version="1.0" encoding="utf-8"?>
<line>
        <points>[1, 5]</points>
        <points>[2, 6]</points>
</line>
>>> print(xmltodict.unparse(mydict, pretty=True, expand_iter="coord"))
<?xml version="1.0" encoding="utf-8"?>
<line>
        <points>
                <coord>1</coord>
                <coord>5</coord>
        </points>
        <points>
                <coord>2</coord>
                <coord>6</coord>
        </points>
</line>
```

API Reference

xmltodict.parse()

Parse XML input into a Python dictionary.

- `xml_input`: XML input as a string, file-like object, or generator of strings.
- `encoding=None`: Character encoding for the input XML.
- `expat=expat`: XML parser module to use.
- `process_namespaces=False`: Expand XML namespaces if True.
- `namespace_separator=':'`: Separator between namespace URI and local name.
- `disable_entities=True`: Disable entity parsing for security.

- `process_comments=False`: Include XML comments if True. Comments can be preserved when enabled, but by default they are ignored. Multiple top-level comments may not be preserved in exact order.
- `xml_attribs=True`: Include attributes in output dict (with `attr_prefix`).
- `attr_prefix='@'`: Prefix for XML attributes in the dict.
- `cdata_key='#text'`: Key for text content in the dict.
- `force_cdata=False`: Force text content to be wrapped as CDATA for specific elements. Can be a boolean (True/False), a tuple of element names to force CDATA for, or a callable function that receives (path, key, value) and returns True/False.
- `cdata_separator=''`: Separator string to join multiple text nodes. This joins adjacent text nodes. For example, set to a space to avoid concatenation.
- `postprocessor=None`: Function to modify parsed items.
- `dict_constructor=dict`: Constructor for dictionaries (e.g., dict).
- `strip_whitespace=True`: Remove leading/trailing whitespace in text nodes. Default is True; this trims whitespace in text nodes. Set to False to preserve whitespace exactly. When `process_comments=True`, this same flag also trims comment text; disable `strip_whitespace` if you need to preserve comment indentation or padding. If you preserve whitespace text nodes, avoid combining this with `unparse(..., pretty=True)`: the preserved text nodes and pretty-print indentation/newlines will both be emitted.
- `namespaces=None`: Mapping of namespaces to prefixes, or None to keep full URIs.
- `force_list=None`: Force list values for specific elements. Can be a boolean (True/False), a tuple of element names to force lists for, or a callable function that receives (path, key, value) and returns True/False. Useful for elements that may appear once or multiple times to ensure consistent list output.
- `item_depth=0`: Depth at which to call `item_callback`.
- `item_callback=lambda *args: True`: Function called on items at `item_depth`.
- `comment_key='#comment'`: Key used for XML comments when `process_comments=True`. Only used when `process_comments=True`. Comments can be preserved but multiple top-level comments may not retain order.

xmltodict.unparse()

Convert a Python dictionary back into XML.

- `input_dict`: Dictionary to convert to XML.
- `output=None`: File-like object to write XML to; returns string if None.
- `encoding='utf-8'`: Encoding of the output XML.
- `bytes_errors='replace'`: Error handler used when decoding byte values during unparse (for example `'replace'`, `'strict'`, `'ignore'`).
- `full_document=True`: Include XML declaration if True.
- `short_empty_elements=False`: Use short tags for empty elements (`<tag/>`).
- `attr_prefix='@'`: Prefix for dictionary keys representing attributes.

- `cdata_key='#text'`: Key for text content in the dictionary.
- `pretty=False`: Pretty-print the XML output. This adds indentation/newlines for readability, so it should generally not be combined with parsed whitespace-preserving content when exact round-tripping matters.
- `indent='\t'`: Indentation string for pretty printing. May also be an integer number of spaces.
- `newl='\n'`: Newline character for pretty printing.
- `expand_iter=None`: Tag name to use for items in nested lists (breaks roundtripping).

> **Note:** When building XML from dictionaries, keys whose values are empty lists are skipped. For example, `{'a': []}` produces no `<a>` element. Add a placeholder child (for example, `{'a': ['']}`) if an explicit empty container element is required in the output.

Note: xmltodict aims to cover the common 90% of cases. It does not preserve every XML nuance (attribute order, mixed content ordering, multiple top-level comments). For exact fidelity, use a full XML library such as lxml.

Examples

Selective force_cdata

The `force_cdata` parameter can be used to selectively force CDATA wrapping for specific elements:

```
>>> xml = '<a><b>data1</b><c>data2</c><d>data3</d></a>'
>>> # Force CDATA only for 'b' and 'd' elements
>>> xmltodict.parse(xml, force_cdata=('b', 'd'))
{'a': {'b': {'#text': 'data1'}, 'c': 'data2', 'd': {'#text': 'data3'}}}

>>> # Force CDATA for all elements (original behavior)
>>> xmltodict.parse(xml, force_cdata=True)
{'a': {'b': {'#text': 'data1'}, 'c': {'#text': 'data2'}, 'd': {'#text': 'data3'}}}

>>> # Use a callable for complex logic
>>> def should_force_cdata(path, key, value):
...     return key in ['b', 'd'] and len(value) > 4
>>> xmltodict.parse(xml, force_cdata=should_force_cdata)
{'a': {'b': {'#text': 'data1'}, 'c': 'data2', 'd': {'#text': 'data3'}}}
```

Selective force_list

The `force_list` parameter can be used to selectively force list values for specific elements:

```
>>> xml = '<a><b>data1</b><b>data2</b><c>data3</c></a>'
>>> # Force lists only for 'b' elements
>>> xmltodict.parse(xml, force_list=('b',))
{'a': {'b': ['data1', 'data2'], 'c': 'data3'}}

>>> # Force lists for all elements (original behavior)
```

```
>>> xmltodict.parse(xml, force_list=True)
{'a': [{'b': ['data1', 'data2'], 'c': ['data3']}]}

>>> # Use a callable for complex logic
>>> def should_force_list(path, key, value):
...     return key in ['b'] and isinstance(value, str)
>>> xmltodict.parse(xml, force_list=should_force_list)
{'a': {'b': ['data1', 'data2'], 'c': 'data3'}}
```

Ok, how do I get it?

Using pypi

You just need to

```
$ pip install xmltodict
```

Using conda

For installing `xmltodict` using Anaconda/Miniconda (*conda*) from the conda-forge channel all you need to do is:

```
$ conda install -c conda-forge xmltodict
```

RPM-based distro (Fedora, RHEL, …)

There is an official Fedora package for xmltodict.

```
$ sudo yum install python3-xmltodict
```

Arch Linux

There is an official Arch Linux package for xmltodict.

```
$ sudo pacman -S python-xmltodict
```

Debian-based distro (Debian, Ubuntu, …)

There is an official Debian package for xmltodict.

```
$ sudo apt install python-xmltodict
```

FreeBSD

There is an official FreeBSD port for xmltodict.

```
$ pkg install py36-xmltodict
```

openSUSE/SLE (SLE 15, Leap 15, Tumbleweed)

There is an official openSUSE package for xmltodict.

```
# Python2
$ zypper in python2-xmltodict

# Python3
$ zypper in python3-xmltodict
```

Type Annotations

For type checking support, install the external types package:

```
# Using pypi
$ pip install types-xmltodict

# Using conda
$ conda install -c conda-forge types-xmltodict
```

Security Notes

A CVE (CVE-2025-9375) was filed against `xmltodict` but is disputed. The root issue lies in Python's `xml.sax.saxutils.XMLGenerator` API, which does not validate XML element names and provides no built-in way to do so. Since `xmltodict` is a thin wrapper that passes keys directly to `XMLGenerator`, the same issue exists in the standard library itself.

It has been suggested that `xml.sax.saxutils.escape()` represents a secure usage path. This is incorrect: `escape()` is intended only for character data and attribute values, and can produce invalid XML when misapplied to element names. There is currently no secure, documented way in Python's standard library to validate XML element names.

Despite this, Fluid Attacks chose to assign a CVE to `xmltodict` while leaving the identical behavior in Python's own standard library unaddressed. Their disclosure process also gave only 10 days from first contact to publication—well short of the 90-day industry norm—leaving no real opportunity for maintainer response. These actions reflect an inconsistency of standards and priorities that raise concerns about

Releases 6

v1.0.4 Latest
on Feb 22

+ 5 releases

Sponsor this project

martinblech Martín Blech

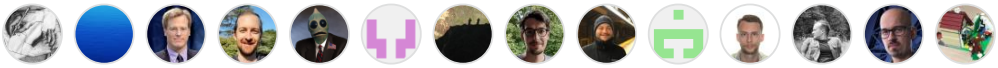
Sponsor

Learn more about GitHub Sponsors

Packages

No packages published

Contributors 49



+ 35 contributors

Languages

Python 100.0%