PyJSON5 Documentation

Release 1.6.0

René Kijewski

Nov 17, 2021

CONTENTS

A JSON5 serializer and parser library for Python 3.5 and later.

The serializer returns ASCII data that can safely be used in an HTML template. Apostrophes, ampersands, greater-than, and less-then signs are encoded as unicode escaped sequences. E.g. this snippet is safe for any and all input:

```
"<a onclick='alert(" + encode(data) + ")'>show message</a>"
```

Unless the input contains infinite or NaN values, the result will be valid JSON data.

All valid JSON5 1.0.0 and JSON data can be read, unless the nesting level is absurdly high.

CHAPTER

ONE

INSTALLATION

```
$ pip install pyjson5
```

CHAPTER

TWO

TABLE OF CONTENTS

2.1 Serializer / Encoder

The serializer returns ASCII data that can safely be used in an HTML template. Apostrophes, ampersands, greater-than, and less-then signs are encoded as unicode escaped sequences. E.g. this snippet is safe for any and all input:

```
"<a onclick='alert(" + encode(data) + ")'>show message</a>"
```

Unless the input contains infinite or NaN values, the result will be valid JSON data.

2.1.1 Quick Encoder Summary

encode(data, *[, options])	Serializes a Python object as a JSON5 compatible string.
encode_bytes(data, *[, options])	Serializes a Python object to a JSON5 compatible bytes string.
encode_callback(data, cb[, supply_bytes, ...])	Serializes a Python object into a callback function.
encode_io(data, fp[, supply_bytes, options])	Serializes a Python object into a file-object.
encode_noop(data, *[, options])	Test if the input is serializable.
dump(obj, fp, **kw)	Serializes a Python object to a JSON5 compatible string.
dumps(obj, **kw)	Serializes a Python object to a JSON5 compatible string.
Options	Customizations for the `encoder_*(...)` function family.
Json5EncoderException	Base class of any exception thrown by the serializer.
Json5UnstringifiableType([message, ...])	The encoder was not able to stringify the input, or it was told not to by the supplied `Options`.

2.1.2 Full Encoder Description

`pyjson5.`**`encode`**(*data*, *, *options=None*, ***options_kw*)

Serializes a Python object as a JSON5 compatible string.

```
encode(['Hello', 'world!']) == '["Hello","world!"]'
```

Parameters

- **data** (*object*) – Python object to serialize.
- **options** (*Optional[Options]*) – Extra options for the encoder. If `options` **and** `options_kw` are specified, then `options.update(**options_kw)` is used.

- **options_kw** – See Option's arguments.

Raises

- *Json5EncoderException* – An exception occured while encoding.
- **TypeError** – An argument had a wrong type.

Returns

Unless `float('inf')` or `float('nan')` is encountered, the result will be valid JSON data (as of RFC8259).

The result is always ASCII. All characters outside of the ASCII range are escaped.

The result safe to use in an HTML template, e.g. `<a onclick='alert({{ encode(url) }})'>show message</a>`. Apostrophes `"'"` are encoded as `"\u0027"`, less-than, greater-than, and ampersand likewise.

Return type str

pyjson5.**encode_bytes**(*data*, *, *options=None*, ***options_kw*)

Serializes a Python object to a JSON5 compatible bytes string.

```
encode_bytes(['Hello', 'world!']) == b'["Hello","world!"]'
```

Parameters

- **data** (*object*) – see *encode(...)*
- **options** (*Optional[Options]*) – see *encode(...)*
- **options_kw** – see *encode(...)*

Raises

- *Json5EncoderException* – An exception occured while encoding.
- **TypeError** – An argument had a wrong type.

Returns see *encode(...)*

Return type bytes

pyjson5.**encode_callback**(*data*, *cb*, *supply_bytes=False*, *, *options=None*, ***options_kw*)

Serializes a Python object into a callback function.

The callback function `cb` gets called with single characters and strings until the input `data` is fully serialized.

```
encode_callback(['Hello', 'world!'], print)
#prints:
# [
# "
# Hello
# "
# ,
# "
# world!
# "
" ]
```

Parameters

- **data** (*object*) – see *encode(...)*
- **cb** (*Callable[[Union[bytes|str]], None]*) – A callback function. Depending on the truthyness of `supply_bytes` either `bytes` or `str` is supplied.
- **supply_bytes** (*bool*) – Call `cb(...)` with a `bytes` argument if true, otherwise `str`.
- **options** (*Optional[Options]*) – see *encode(...)*
- **options_kw** – see *encode(...)*

Raises

- *Json5EncoderException* – An exception occured while encoding.
- **TypeError** – An argument had a wrong type.

Returns The supplied argument `cb`.

Return type Callable[[Union[bytes|str]], None]

`pyjson5.`**`encode_io`**(*data*, *fp*, *supply_bytes=True*, *, *options=None*, ***options_kw*)

Serializes a Python object into a file-object.

The return value of `fp.write(...)` is not checked. If `fp` is unbuffered, then the result will be garbage!

Parameters

- **data** (*object*) – see *encode(...)*
- **fp** (*IOBase*) – A file-like object to serialize into.
- **supply_bytes** (*bool*) – Call `fp.write(...)` with a `bytes` argument if true, otherwise `str`.
- **options** (*Optional[Options]*) – see *encode(...)*
- **options_kw** – see *encode(...)*

Raises

- *Json5EncoderException* – An exception occured while encoding.
- **TypeError** – An argument had a wrong type.

Returns The supplied argument `fp`.

Return type IOBase

`pyjson5.`**`encode_noop`**(*data*, *, *options=None*, ***options_kw*)

Test if the input is serializable.

Most likely you want to serialize `data` directly, and catch exceptions instead of using this function!

```
encode_noop({47: 11}) == True
encode_noop({47: object()}) == False
```

Parameters

- **data** (*object*) – see *encode(...)*
- **options** (*Optional[Options]*) – see *encode(...)*
- **options_kw** – see *encode(...)*

Returns `True` iff `data` is serializable.

Return type bool

class pyjson5.**Options**

Customizations for the `encoder_*(...)` function family.

Immutable. Use *Options.update(**kw)* to create a **new** Options instance.

Parameters

- **quotationmark** (*str*|*None*) –
  - **str**: One character string that is used to surround strings.
  - **None**: Use default: `'"'`.
- **tojson** (*str*|*False*|*None*) –
  - **str:** A special method to call on objects to return a custom JSON encoded string. Must return ASCII data!
  - **False:** No such member exists. (Default.)
  - **None:** Use default.
- **mappingtypes** (*Iterable[type]*|*False*|*None*) –
  - **Iterable[type]:** Classes that should be encoded to objects. Must be iterable over their keys, and implement `__getitem__`.
  - **False:** There are no objects. Any object will be encoded as list of keys as in list(obj).
  - **None:** Use default: `[collections.abc.Mapping]`.

mappingtypes

The creation argument `mappingtypes`. `()` if `False` was specified.

quotationmark

The creation argument `quotationmark`.

tojson

The creation argument `tojson`. `None` if `False` was specified.

update(*self*, **args*, ***kw*)

Creates a new Options instance by modifying some members.

2.1.3 Encoder Compatibility Functions

pyjson5.**dump**(*obj*, *fp*, ***kw*)

Serializes a Python object to a JSON5 compatible string.

Use *encode_io(...)* instead!

```
dump(obj, fp) == encode_io(obj, fp)
```

Parameters

- **obj** (*object*) – Python object to serialize.
- **fp** (*IOBase*) – A file-like object to serialize into.
- **kw** – Silently ignored.

pyjson5.**dumps**(*obj*, ***kw*)

Serializes a Python object to a JSON5 compatible string.

Use *encode(...)* instead!

```
dumps(obj) == encode(obj)
```

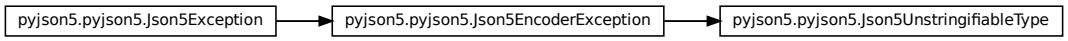
Parameters

- **obj** (*object*) – Python object to serialize.
- **kw** – Silently ignored.

Returns see *encode(...)*

Return type str

2.1.4 Encoder Exceptions



class pyjson5.**Json5EncoderException**

Base class of any exception thrown by the serializer.

message

Human readable error description

with_traceback()

Exception.with_traceback(tb) – set self.__traceback__ to tb and return self.

class pyjson5.**Json5UnstringifiableType**(*message=None*, *unstringifiable=None*)

The encoder was not able to stringify the input, or it was told not to by the supplied `Options`.

message

Human readable error description

unstringifiable

The value that caused the problem.

with_traceback()

Exception.with_traceback(tb) – set self.__traceback__ to tb and return self.

2.2 Parser / Decoder

All valid JSON5 1.0.0 and JSON data can be read, unless the nesting level is absurdly high.

2.2.1 Quick Decoder Summary

decode(data[, maxdepth, some])	Decodes JSON5 serialized data from an `str` object.
decode_latin1(data[, maxdepth, some])	Decodes JSON5 serialized data from a `bytes` object.
decode_buffer(obj[, maxdepth, some, wordlength])	Decodes JSON5 serialized data from an object that supports the buffer protocol, e.g.
decode_callback(cb[, maxdepth, some, args])	Decodes JSON5 serialized data by invoking a callback.
decode_io(fp[, maxdepth, some])	Decodes JSON5 serialized data from a file-like object.
load(fp, **kw)	Decodes JSON5 serialized data from a file-like object.
loads(s, *[, encoding])	Decodes JSON5 serialized data from a string.
Json5DecoderException([message, result])	Base class of any exception thrown by the parser.
Json5NestingTooDeep	The maximum nesting level on the input data was exceeded.
Json5EOF	The input ended prematurely.
Json5IllegalCharacter([message, result, ...])	An unexpected character was encountered.
Json5ExtraData([message, result, character])	The input contained extranous data.
Json5IllegalType([message, result, value])	The user supplied callback function returned illegal data.

2.2.2 Full Decoder Description

`pyjson5.`**`decode`**(*data*, *maxdepth=None*, *some=False*)

Decodes JSON5 serialized data from an `str` object.

```
decode('["Hello", "world!"]') == ['Hello', 'world!']
```

Parameters

- **data** (*str*) – JSON5 serialized data
- **maxdepth** (*Optional[int]*) – Maximum nesting level before are the parsing is aborted.
  - If `None` is supplied, then the value of the global variable `DEFAULT_MAX_NESTING_LEVEL` is used instead.
  - If the value is `0`, then only literals are accepted, e.g. `false`, `47.11`, or `"string"`.
  - If the value is negative, then the any nesting level is allowed until Python's recursion limit is hit.
- **some** (*bool*) – Allow trailing junk.

Raises

- *Json5DecoderException* – An exception occured while decoding.
- **TypeError** – An argument had a wrong type.

Returns Deserialized data.

Return type object

`pyjson5.`**`decode_latin1`**(*data*, *maxdepth=None*, *some=False*)

Decodes JSON5 serialized data from a `bytes` object.

```
decode_latin1(b'["Hello", "world!"]') == ['Hello', 'world!']
```

Parameters

- **data** (*bytes*) – JSON5 serialized data, encoded as Latin-1 or ASCII.
- **maxdepth** (*Optional[int]*) – see *decode(...)*
- **some** (*bool*) – see *decode(...)*

Raises

- ***Json5DecoderException*** – An exception occured while decoding.
- **TypeError** – An argument had a wrong type.

Returns see *decode(...)*

Return type object

pyjson5.**decode_buffer**(*obj*, *maxdepth=None*, *some=False*, *wordlength=None*)

Decodes JSON5 serialized data from an object that supports the buffer protocol, e.g. `bytearray`.

```
obj = memoryview(b'["Hello", "world!"]')

decode_buffer(obj) == ['Hello', 'world!']
```

Parameters

- **data** (*object*) – JSON5 serialized data. The argument must support Python's buffer protocol, i.e. `memoryview(...)` must work. The buffer must be contigious.
- **maxdepth** (*Optional[int]*) – see *decode(...)*
- **some** (*bool*) – see *decode(...)*
- **wordlength** (*Optional[int]*) – Must be 0, 1, 2, 4 to denote UTF-8, UCS1, USC2 or USC4 data, resp. Surrogates are not supported. Decode the data to an `str` if need be. If `None` is supplied, then the buffer's `itemsize` is used.

Raises

- ***Json5DecoderException*** – An exception occured while decoding.
- **TypeError** – An argument had a wrong type.
- **ValueError** – The value of `wordlength` was invalid.

Returns see *decode(...)*

Return type object

pyjson5.**decode_callback**(*cb*, *maxdepth=None*, *some=False*, *args=None*)

Decodes JSON5 serialized data by invoking a callback.

```
cb = iter('["Hello","world!"]').__next__

decode_callback(cb) == ['Hello', 'world!']
```

Parameters

- **cb** (*Callable[Any, Union[str|bytes|bytearray|int|None]]*) – A function to get values from. The functions is called like `cb(*args)`, and it returns:

  - **str, bytes, bytearray:** `len(...) == 0` denotes exhausted input. `len(...) == 1` is the next character.
  - **int:** `< 0` denotes exhausted input. `>= 0` is the ordinal value of the next character.
  - **None:** input exhausted
- **maxdepth** (*Optional[int]*) – see *decode(...)*
- **some** (*bool*) – see *decode(...)*
- **args** (*Optional[Iterable[Any]]*) – Arguments to call `cb` with.

Raises

- *Json5DecoderException* – An exception occured while decoding.
- **TypeError** – An argument had a wrong type.

Returns see *decode(...)*

Return type object

`pyjson5.decode_io(fp, maxdepth=None, some=True)`

Decodes JSON5 serialized data from a file-like object.

```
fp = io.StringIO("""
    ['Hello', /* TODO look into specs whom to greet */]
    'Wolrd' // FIXME: look for typos
""")

decode_io(fp) == ['Hello']
decode_io(fp) == 'Wolrd'

fp.seek(0)

decode_io(fp, some=False)
# raises Json5ExtraData('Extra data U+0027 near 56', ['Hello'], "")
```

Parameters

- **fp** (*IOBase*) – A file-like object to parse from.
- **maxdepth** (*Optional[int] = None*) – see *decode(...)*
- **some** (*bool*) – see *decode(...)*

Raises

- *Json5DecoderException* – An exception occured while decoding.
- **TypeError** – An argument had a wrong type.

Returns see *decode(...)*

Return type object

2.2.3 Decoder Compatibility Functions

`pyjson5.load(fp, **kw)`

Decodes JSON5 serialized data from a file-like object.

Use `decode_io(...)` instead!

```
load(fp) == decode_io(fp, None, False)
```

Parameters

- **fp** (*IOBase*) – A file-like object to parse from.
- **kw** – Silently ignored.

Returns see `decode_io(...)`

Return type str

`pyjson5.loads(s, *, encoding='UTF-8', **kw)`

Decodes JSON5 serialized data from a string.

Use `decode(...)` instead!

```
loads(s) == decode(s)
```

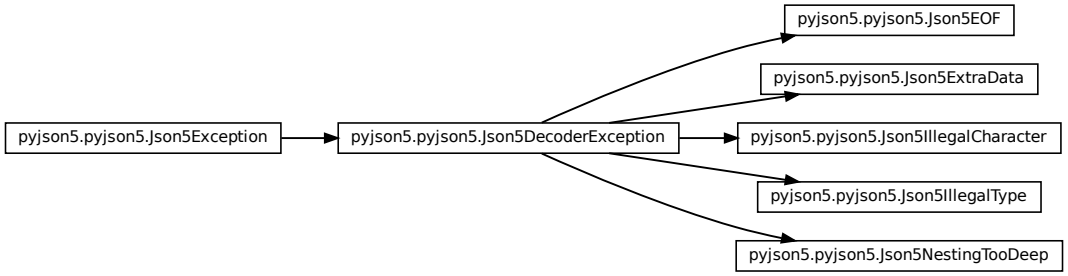
Parameters

- **s** (*object*) – Unless the argument is an `str`, it gets decoded according to the parameter `encoding`.
- **encoding** (*str*) – Codec to use if `s` is not an `str`.
- **kw** – Silently ignored.

Returns see `decode(...)`

Return type object

2.2.4 Decoder Exceptions



class pyjson5.**Json5DecoderException**(*message=None*, *result=None*, **args*)

Base class of any exception thrown by the parser.

message

Human readable error description

result

Deserialized data up until now.

with_traceback()

Exception.with_traceback(tb) – set self.__traceback__ to tb and return self.

class pyjson5.**Json5NestingTooDeep**

The maximum nesting level on the input data was exceeded.

message

Human readable error description

result

Deserialized data up until now.

with_traceback()

Exception.with_traceback(tb) – set self.__traceback__ to tb and return self.

class pyjson5.**Json5EOF**

The input ended prematurely.

message

Human readable error description

result

Deserialized data up until now.

with_traceback()

Exception.with_traceback(tb) – set self.__traceback__ to tb and return self.

class pyjson5.**Json5IllegalCharacter**(*message=None*, *result=None*, *character=None*, **args*)

An unexpected character was encountered.

character

Illegal character.

message

Human readable error description

result

Deserialized data up until now.

with_traceback()

Exception.with_traceback(tb) – set self.__traceback__ to tb and return self.

class pyjson5.**Json5ExtraData**(*message=None*, *result=None*, *character=None*, **args*)

The input contained extranous data.

character

Extranous character.

message

Human readable error description

result

Deserialized data up until now.

with_traceback()
Exception.with_traceback(tb) – set self.__traceback__ to tb and return self.

class pyjson5.**Json5IllegalType**(*message=None*, *result=None*, *value=None*, **args*)
The user supplied callback function returned illegal data.

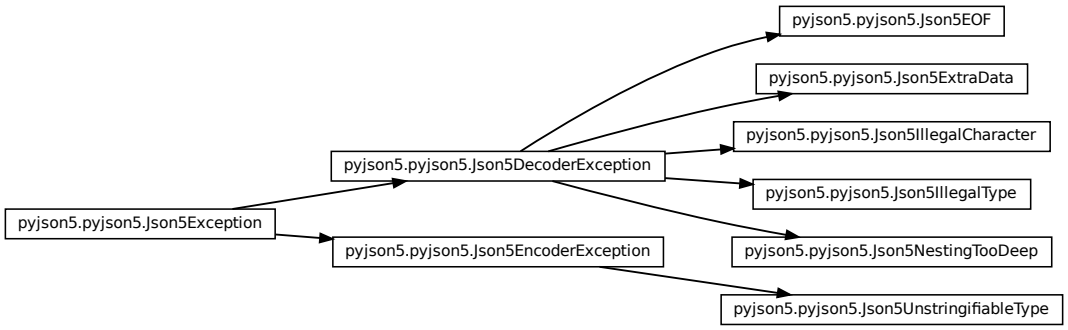
message
Human readable error description

result
Deserialized data up until now.

value
Value that caused the problem.

with_traceback()
Exception.with_traceback(tb) – set self.__traceback__ to tb and return self.

2.3 Exceptions



class pyjson5.**Json5Exception**(*message=None*, **args*)
Base class of any exception thrown by PyJSON5.

message
Human readable error description

with_traceback()
Exception.with_traceback(tb) – set self.__traceback__ to tb and return self.

2.4 Performance

This library is written in Cython for a better performance than a pure-Python implementation could give you.

2.4.1 Decoder Performance

The library has about the same speed as the shipped `json` module for *pure* JSON data.

- Dataset: https://github.com/zemirco/sf-city-lots-json
- Version: Python 3.9.1+ (default, Feb 5 2021, 13:46:56)
- CPU: AMD Ryzen 7 2700 @ 3.7GHz
- *`pyjson5.decode()`*: **2.08 s** ± 7.49 ms per loop *(lower is better)*
- `json.loads()`: **2.71 s** ± 12.1 ms per loop
- The decoder works correcty: `json.loads(content) == pyjson5.loads(content)`

2.4.2 Encoder Performance

The encoder generates pure JSON data if there are no infinite or NaN values in the input, which are invalid in JSON. The serialized data is XML-safe, i.e. there are no cheverons <>, ampersands &, apostrophes `'` or control characters in the output. The output is always ASCII regardless if you call *`pyjson5.encode()`* or *`pyjson5.encode_bytes()`*.

- Dataset: https://github.com/zemirco/sf-city-lots-json
- Python 3.9.1+ (default, Feb 5 2021, 13:46:56)
- CPU: AMD Ryzen 7 2700 @ 3.7GHz
- *`pyjson5.encode()`*: **1.37** s ± 19.2 per loop *(lower is better)*
- `json.dumps()`: **3.66** s ± 72.6 ms per loop
- `json.dumps()` + `xml.sax.saxutils.escape()`: **4.01** s ± 21.3 ms per loop
- The encoder works correcty: `obj == json.loads(pyjson5.encode(obj))`

2.4.3 Benchmark

Using Ultrajson's benchmark you can tell for which kind of data PyJSON5 is fast, and for which data it is slow in comparison *(higher is better)*:

	json	pyjson5	ujson	orjson
Array with 256 doubles				
encode	6,425	81,202	28,966	83,836
decode	16,759	34,801	34,794	80,655
Array with 256 strings				
encode	36,969	73,165	35,574	113,082
decode	42,730	38,542	38,386	60,732
Array with 256 UTF-8 strings				
encode	3,458	3,134	4,024	31,677
decode	2,428	2,498	2,491	1,750
Array with 256 True values				
encode	130,441	282,703	131,279	423,371
decode	220,657	262,690	264,485	262,283
Array with 256 dict{string, int} pairs				
encode	11,621	10,014	18,148	73,905
decode	17,802	19,406	19,391	23,478
Dict with 256 arrays with 256 dict{string, int} pairs				
encode	40	38	68	213
decode	43	49	48	51
Medium complex object				
encode	8,704	11,922	15,319	49,677
decode	12,567	14,042	13,985	19,481
Complex object				
encode	672	909	731	
decode	462	700	700	

- ujson == 4.0.3.dev9
- orjson == 3.5.1

2.5 Changelog

1.6.0

- Fallback to encode `vars(obj)` if `obj` is not stringifyable, e.g. to serialize dataclasses.
- Update documentation to use newer sphinx version.
- Use dependabot to keep dependencies current.
- Update fast_double_parser.

1.5.3

- Add PEP 484 type hints (by Pascal Corpet)
- Update JSONTestSuite

1.5.2

- Add file extensions to fix compilation with current Apple SDKs
- Update fast_double_parser to v0.5.0
- Update to Unicode 14.0.0d18

1.5.1

- Update up Unicode 14.0.0d9

1.5.0

- Faster floating-point number encoding using Junekey Jeon's Dragonbox algorithm implemented by Alexander Bolz
- Removed a lot of configuration options from pyjson5.Options()

1.4.9

- Faster floating-point number decoding using fast_double_parser by Daniel Lemire

1.4.8

- Update up Unicode 13.0.0
- Don't use non-standard `__uint128`
- Add PyPy compatibility
- Add `decode_utf8(byte-like)`

1.4.7

- Allow `\uXXXX` sequences in identifier names
- Update to Unicode 12.1.0
- Optimized encoder and decoder for a little better speed
- Setup basic CI environment
- Parse `\uXXXX` in literal keys
- Understand "0."
- Add CI tests
- Reject unescaped newlines in strings per spec
- Allow overriding default quotation mark
- Make Options objects pickle-able
- Bump major version number

0.4.6

- Fix PyUnicode_AsUTF8AndSize()'s signature

0.4.5

- Don't use C++14 features, only C++11

0.4.4

- Better documentation
- Optimized encoder for a little better speed

0.4.3

- Initial release

CHAPTER

THREE

QUICK SUMMARY

decode(data[, maxdepth, some])	Decodes JSON5 serialized data from an `str` object.
decode_buffer(obj[, maxdepth, some, wordlength])	Decodes JSON5 serialized data from an object that supports the buffer protocol, e.g.
decode_callback(cb[, maxdepth, some, args])	Decodes JSON5 serialized data by invoking a callback.
decode_io(fp[, maxdepth, some])	Decodes JSON5 serialized data from a file-like object.
load(fp, **kw)	Decodes JSON5 serialized data from a file-like object.
loads(s, *[, encoding])	Decodes JSON5 serialized data from a string.
encode(data, *[, options])	Serializes a Python object as a JSON5 compatible string.
encode_bytes(data, *[, options])	Serializes a Python object to a JSON5 compatible bytes string.
encode_callback(data, cb[, supply_bytes, ...])	Serializes a Python object into a callback function.
encode_io(data, fp[, supply_bytes, options])	Serializes a Python object into a file-object.
encode_noop(data, *[, options])	Test if the input is serializable.
dump(obj, fp, **kw)	Serializes a Python object to a JSON5 compatible string.
dumps(obj, **kw)	Serializes a Python object to a JSON5 compatible string.
Options	Customizations for the `encoder_*(...)` function family.
Json5EncoderException	Base class of any exception thrown by the serializer.
Json5DecoderException([message, result])	Base class of any exception thrown by the parser.

CHAPTER

FOUR

COMPATIBILITY

At least CPython / PyPy 3.5, and a C++11 compatible compiler (such as GCC 5.2+) is needed.

Glossary / Index

INDEX