
Meta Documentation

Release 0.4.1

Sean Ross-Ross

January 22, 2015

1	Example	3
2	Notes	5
3	Bugs	7
4	Testing	9
4.1	Meta API	9
4.2	License	12
5	Indices and tables	13
	Python Module Index	15

A Pure Python module containing a framework to manipulate and analyze python ast's and bytecode.

Example

This shows how to take python source to a code object and back again from within python:

```
import meta, ast
source = '''
a = 1
b = 2
c = (a ** b)
'''

mod = ast.parse(source, '<nofile>', 'exec')
code = compile(mod, '<nofile>', 'exec')

mod2 = meta.decompile(code)
source2 = meta.dump_python_source(mod2)

assert source == source2
```

This shows the depyc script. The script compiles itself, and then the compiled script extracts itself:

```
DEPYC_FILE=`python -c"import meta.scripts.depyc; print meta.scripts.depyc.__file__"`
depyc $DEPYC_FILE --pyc > depycX.pyc
python -m depycX depycX.pyc --python > depycX.py
echo depycX.py
```

Notes

- Meta is python3 compliant (mostly)

Bugs

- The decompiler does not yet support complex list/set/dict - comprehensions

```
python -m unittest discover meta
```

Contents:

4.1 Meta API

Meta is a suite of tools to manipulate python ast and byte code.

4.1.1 `meta.asttools` - operate on python ast nodes

Module to augment and analyze python ast nodes.

This module uses the python `ast` module exclusively not the deprecated `compiler.ast`.

```
class meta.asttools.Undefined
```

```
meta.asttools.cmp_ast (node1, node2)
```

Compare if two nodes are equal.

```
meta.asttools.print_ast (ast, indent=' ', initlevel=0, newline='\n', file=<open file '<stdout>', mode
                        'w' at 0x7f7de8452150>)
```

Pretty print an ast node.

Parameters

- **ast** – the ast to print.
- **indent** – how far to indent a newline.
- **initlevel** – starting indent level
- **newline** – The newline character.
- **file** – file object to print to

To print a short ast you may want to use:

```
node = ast.parse(source)
print_ast(node, indent=' ', newline='')
```

```
meta.asttools.str_ast (ast, indent=' ', newline='\n')
```

Returns a string representing the ast.

Parameters

- **ast** – the ast to print.
- **indent** – how far to indent a newline.
- **newline** – The newline character.

`meta.asttools.python_source(ast, file=<open file '<stdout>', mode 'w' at 0x7f7de8452150>)`

Generate executable python source code from an ast node.

Parameters

- **ast** – ast node
- **file** – file to write output to.

`meta.asttools.dump_python_source(ast)`

Returns a string containing executable python source code from an ast node.

Parameters

- **ast** – ast node
- **file** – file to write output to.

`meta.asttools.lhs(node)`

Return a set of symbols in *node* that are assigned.

Parameters *node* – ast node

Returns set of strings.

`meta.asttools.rhs(node)`

Return a set of symbols in *node* that are used.

Parameters *node* – ast node

Returns set of strings.

`meta.asttools.conditional_lhs(node)`

Group outputs into conditional and stable :param node: ast node

Returns tuple of (conditional, stable)

`meta.asttools.conditional_symbols(node)`

Group lhs and rhs into conditional, stable and undefined :param node: ast node

Returns tuple of (conditional_lhs, stable_lhs),(conditional_rhs, stable_rhs), undefined

`meta.asttools.get_symbols(node, ctx_types=(<class '_ast.Load'>, <class '_ast.Store'>))`

Returns all symbols defined in an ast node.

if *ctx_types* is given, then restrict the symbols to ones with that context.

Parameters

- **node** – ast node
- **ctx_types** – type or tuple of types that may be found assigned to the *ctx* attribute of an ast Name node.

`meta.asttools.make_graph(node, call_deps=False)`

Create a dependency graph from an ast node.

Parameters

- **node** – ast node.

- **call_deps** – if true, then the graph will create a cyclic dependance for all function calls. (i.e for *a.b(c)* a depends on b and b depends on a)

Returns a tuple of (graph, undefined)

4.1.2 `meta.decompiler` - decompile code objects into ast nodes

Decompiler module.

This module can decompile arbitrary code objects into a python ast.

`meta.decompiler.decompile_func` (*func*)

Decompile a function into ast.FunctionDef node.

Parameters **func** – python function (can not be a built-in)

Returns ast.FunctionDef instance.

`meta.decompiler.compile_func` (*ast_node, filename, globals, **defaults*)

Compile a function from an ast.FunctionDef instance.

Parameters

- **ast_node** – ast.FunctionDef instance
- **filename** – path where function source can be found.
- **globals** – will be used as func_globals

Returns A python function object

`meta.decompiler.decompile_pyc` (*bin_pyc, output=<open file '<stdout>', mode 'w' at 0x7f7de8452150>*)

decompile apython pyc or pyo binary file.

Parameters

- **bin_pyc** – input file objects
- **output** – output file objects

4.1.3 `meta.bytecodetools` - operate on python byte-code

Python byte-code tools expands on the Python dis module.

class `meta.bytecodetools.Instruction` (*i=-1, op=None, lineno=None*)

A Python byte-code instruction.

class `meta.bytecodetools.disassembler`

Disassemble a code object.

Parameters

- **co** – code object
- **lasti** – internal

Yields Instructions.

class `meta.bytecodetools.ByteCodeConsumer` (*code*)

ByteCodeVisitor

class `meta.bytecodetools.StackedByteCodeConsumer` (*code*)

A consumer with the concept of a stack.

4.2 License

Copyright © 2011 Enthought Inc. All Rights Reserved.

Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

1. Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.
2. Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.
3. The name of the author may not be used to endorse or promote products derived from this software without specific prior written permission.

THIS SOFTWARE IS PROVIDED BY [LICENSOR] “AS IS” AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE AUTHOR BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

Indices and tables

- *genindex*
- *modindex*
- *search*

m

`meta.asttools`, 9

`meta.bytecodetools`, 11

`meta.decompiler`, 11

B

ByteCodeConsumer (class in meta.bytecodetools), 11

C

cmp_ast() (in module meta.asttools), 9

compile_func() (in module meta.decompiler), 11

conditional_lhs() (in module meta.asttools), 10

conditional_symbols() (in module meta.asttools), 10

D

decompile_func() (in module meta.decompiler), 11

decompile_pyc() (in module meta.decompiler), 11

disassembler (class in meta.bytecodetools), 11

dump_python_source() (in module meta.asttools), 10

G

get_symbols() (in module meta.asttools), 10

I

Instruction (class in meta.bytecodetools), 11

L

lhs() (in module meta.asttools), 10

M

make_graph() (in module meta.asttools), 10

meta.asttools (module), 9

meta.bytecodetools (module), 11

meta.decompiler (module), 11

P

print_ast() (in module meta.asttools), 9

python_source() (in module meta.asttools), 10

R

rhs() (in module meta.asttools), 10

S

StackedByteCodeConsumer (class in meta.bytecodetools), 11

str_ast() (in module meta.asttools), 9

U

Undefined (class in meta.asttools), 9