Epydoc 3.0.1

API Documentation

June 13, 2008

Contents

1 Package epydoc 9
   1.1 Variables .................................................. 12

2 Module epydoc.apidoc 14
   2.1 Functions .................................................. 14
   2.2 Variables .................................................. 15
   2.3 Class DottedName ............................................ 15
      2.3.1 Methods ............................................... 16
      2.3.2 Class Variables ....................................... 17
   2.4 Class DottedName.InvalidDottedName ....................... 18
      2.4.1 Methods ............................................... 18
      2.4.2 Properties ............................................ 18
   2.5 Class _Sentinel ............................................ 18
      2.5.1 Methods ............................................... 18
   2.6 Class APIDoc ................................................. 19
      2.6.1 Methods ............................................... 19
      2.6.2 Class Variables ....................................... 21
      2.6.3 Instance Variables .................................... 22
   2.7 Class VariableDoc ............................................ 23
      2.7.1 Methods ............................................... 24
      2.7.2 Properties ............................................ 25
      2.7.3 Instance Variables .................................... 25
   2.8 Class ValueDoc ................................................. 27
      2.8.1 Methods ............................................... 28
      2.8.2 Class Variables ....................................... 29
      2.8.3 Instance Variables .................................... 30
   2.9 Class GenericValueDoc ....................................... 32
      2.9.1 Methods ............................................... 32
      2.9.2 Class Variables ....................................... 33
      2.9.3 Instance Variables .................................... 33
   2.10 Class NamespaceDoc ......................................... 34
      2.10.1 Methods ............................................... 34
      2.10.2 Class Variables ....................................... 36
      2.10.3 Instance Variables .................................... 36
   2.11 Class ModuleDoc ............................................. 38
      2.11.1 Methods ............................................... 39
2.11.2 Class Variables .............................................. 40
2.11.3 Instance Variables ........................................... 41
2.12 Class ClassDoc .................................................. 43
  2.12.1 Methods ..................................................... 43
  2.12.2 Class Variables ............................................ 46
  2.12.3 Instance Variables .......................................... 46
2.13 Class RoutineDoc ................................................ 47
  2.13.1 Methods ..................................................... 48
  2.13.2 Class Variables ............................................ 48
  2.13.3 Instance Variables .......................................... 48
2.14 Class ClassMethodDoc ........................................... 52
  2.14.1 Methods ..................................................... 52
  2.14.2 Class Variables ............................................ 52
  2.14.3 Instance Variables .......................................... 52
2.15 Class StaticMethodDoc .......................................... 54
  2.15.1 Methods ..................................................... 54
  2.15.2 Class Variables ............................................ 54
  2.15.3 Instance Variables .......................................... 54
2.16 Class PropertyDoc .............................................. 56
  2.16.1 Methods ..................................................... 56
  2.16.2 Class Variables ............................................ 57
  2.16.3 Instance Variables .......................................... 57
2.17 Class DocIndex .................................................. 58
  2.17.1 Methods ..................................................... 59
  2.17.2 Instance Variables .......................................... 61

3 Module epydoc.checker ............................................ 62
  3.1 Variables ...................................................... 62
  3.2 Class DocChecker .............................................. 62
    3.2.1 Methods ..................................................... 63
    3.2.2 Class Variables ............................................ 64

4 Module epydoc.cli .................................................. 68
  4.1 Functions ...................................................... 69
  4.2 Variables ...................................................... 70
  4.3 Class ConsoleLogger .......................................... 72
    4.3.1 Methods ..................................................... 72
    4.3.2 Instance Variables .......................................... 73
  4.4 Class UnifiedProgressConsoleLogger ......................... 74
    4.4.1 Methods ..................................................... 74
    4.4.2 Instance Variables .......................................... 75
  4.5 Class HTMLLogger ............................................. 75
    4.5.1 Methods ..................................................... 75
    4.5.2 Class Variables ............................................ 76

5 Module epydoc.compat ............................................. 78
  5.1 Functions ...................................................... 78

6 Module epydoc.docbuilder ........................................ 79
  6.1 Functions ...................................................... 79
  6.2 Variables ...................................................... 84
  6.3 Class BuildOptions ........................................... 85
    6.3.1 Methods ..................................................... 85
<table>
<thead>
<tr>
<th>Section</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>6.4</td>
<td>Class _ProgressEstimator</td>
<td>86</td>
</tr>
<tr>
<td>6.4.1</td>
<td>Methods</td>
<td>86</td>
</tr>
<tr>
<td>7</td>
<td>Module epydoc.docintrospecter</td>
<td>87</td>
</tr>
<tr>
<td>7.1</td>
<td>Functions</td>
<td>87</td>
</tr>
<tr>
<td>7.2</td>
<td>Variables</td>
<td>91</td>
</tr>
<tr>
<td>7.3</td>
<td>Class _DevNull</td>
<td>92</td>
</tr>
<tr>
<td></td>
<td>7.3.1 Methods</td>
<td>92</td>
</tr>
<tr>
<td>8</td>
<td>Module epydoc.docparser</td>
<td>93</td>
</tr>
<tr>
<td>8.1</td>
<td>Functions</td>
<td>93</td>
</tr>
<tr>
<td>8.2</td>
<td>Variables</td>
<td>101</td>
</tr>
<tr>
<td>8.3</td>
<td>Class ParseError</td>
<td>104</td>
</tr>
<tr>
<td></td>
<td>8.3.1 Methods</td>
<td>104</td>
</tr>
<tr>
<td></td>
<td>8.3.2 Properties</td>
<td>104</td>
</tr>
<tr>
<td>9</td>
<td>Module epydoc.docstringparser</td>
<td>105</td>
</tr>
<tr>
<td>9.1</td>
<td>Functions</td>
<td>105</td>
</tr>
<tr>
<td>9.2</td>
<td>Variables</td>
<td>109</td>
</tr>
<tr>
<td>9.3</td>
<td>Class DocstringField</td>
<td>111</td>
</tr>
<tr>
<td></td>
<td>9.3.1 Methods</td>
<td>111</td>
</tr>
<tr>
<td></td>
<td>9.3.2 Instance Variables</td>
<td>111</td>
</tr>
<tr>
<td>10</td>
<td>Package epydoc.docwriter</td>
<td>113</td>
</tr>
<tr>
<td>11</td>
<td>Module epydoc.docwriter.dotgraph</td>
<td>114</td>
</tr>
<tr>
<td>11.1</td>
<td>Functions</td>
<td>116</td>
</tr>
<tr>
<td>11.2</td>
<td>Variables</td>
<td>119</td>
</tr>
<tr>
<td>11.3</td>
<td>Class DotGraph</td>
<td>120</td>
</tr>
<tr>
<td></td>
<td>11.3.1 Methods</td>
<td>120</td>
</tr>
<tr>
<td></td>
<td>11.3.2 Class Variables</td>
<td>122</td>
</tr>
<tr>
<td></td>
<td>11.3.3 Instance Variables</td>
<td>123</td>
</tr>
<tr>
<td>11.4</td>
<td>Class DotGraphNode</td>
<td>124</td>
</tr>
<tr>
<td></td>
<td>11.4.1 Methods</td>
<td>124</td>
</tr>
<tr>
<td></td>
<td>11.4.2 Class Variables</td>
<td>124</td>
</tr>
<tr>
<td>11.5</td>
<td>Class DotGraphEdge</td>
<td>124</td>
</tr>
<tr>
<td></td>
<td>11.5.1 Methods</td>
<td>124</td>
</tr>
<tr>
<td></td>
<td>11.5.2 Instance Variables</td>
<td>125</td>
</tr>
<tr>
<td>11.6</td>
<td>Class DotGraphUmlClassNode</td>
<td>125</td>
</tr>
<tr>
<td></td>
<td>11.6.1 Methods</td>
<td>128</td>
</tr>
<tr>
<td></td>
<td>11.6.2 Class Variables</td>
<td>130</td>
</tr>
<tr>
<td></td>
<td>11.6.3 Instance Variables</td>
<td>132</td>
</tr>
<tr>
<td>11.7</td>
<td>Class DotGraphUmlModuleNode</td>
<td>133</td>
</tr>
<tr>
<td></td>
<td>11.7.1 Methods</td>
<td>133</td>
</tr>
<tr>
<td></td>
<td>11.7.2 Class Variables</td>
<td>134</td>
</tr>
<tr>
<td>12</td>
<td>Module epydoc.docwriter.html</td>
<td>135</td>
</tr>
<tr>
<td>12.1</td>
<td>Functions</td>
<td>135</td>
</tr>
<tr>
<td>12.2</td>
<td>Class HTMLWriter</td>
<td>137</td>
</tr>
<tr>
<td></td>
<td>12.2.1 Methods</td>
<td>137</td>
</tr>
<tr>
<td></td>
<td>12.2.2 Class Variables</td>
<td>148</td>
</tr>
<tr>
<td></td>
<td>12.2.3 Instance Variables</td>
<td>151</td>
</tr>
<tr>
<td>12.3</td>
<td>Class _HTMLDocstringLinker</td>
<td>153</td>
</tr>
<tr>
<td></td>
<td>12.3.1 Methods</td>
<td>154</td>
</tr>
</tbody>
</table>
20.2 Variables .................................................. 185
20.3 Class GUILogger ........................................... 188
  20.3.1 Methods .............................................. 188
  20.3.2 Class Variables ...................................... 189
20.4 Class EpydocGUI ........................................... 190
  20.4.1 Methods .............................................. 190

21 Module epydoc.log ........................................... 192
  21.1 Functions ............................................... 192
  21.2 Variables ............................................... 193
  21.3 Class Logger ............................................. 194
    21.3.1 Methods ........................................... 194
  21.4 Class SimpleLogger ...................................... 195
    21.4.1 Methods ........................................... 195

22 Package epydoc.markup ...................................... 197
  22.1 Functions ............................................... 197
  22.2 Variables ............................................... 198
  22.3 Class ParsedDocstring ................................. 199
    22.3.1 Methods ........................................... 200
  22.4 Class Field ............................................. 201
    22.4.1 Methods ........................................... 201
  22.5 Class DocstringLinker .................................. 202
    22.5.1 Methods ........................................... 202
  22.6 Class ConcatenatedDocstring ......................... 203
    22.6.1 Methods ........................................... 203
  22.7 Class ParseError ....................................... 204
    22.7.1 Methods ........................................... 204
    22.7.2 Properties ......................................... 206
    22.7.3 Instance Variables ............................... 206

23 Module epydoc.markup.doctest ............................... 207
  23.1 Functions ............................................... 207
  23.2 Class DoctestColorizer ................................ 208
    23.2.1 Methods ........................................... 208
    23.2.2 Class Variables ................................... 209
  23.3 Class XMLDoctestColorizer ......................... 211
    23.3.1 Methods ........................................... 212
    23.3.2 Class Variables ................................... 212
  23.4 Class HTMLDoctestColorizer ...................... 213
    23.4.1 Methods ........................................... 213
    23.4.2 Class Variables ................................... 213
  23.5 Class LaTeXDoctestColorizer .................... 214
    23.5.1 Methods ........................................... 214
    23.5.2 Class Variables ................................... 214

24 Module epydoc.markup.epytext ............................... 216
  24.1 Functions ............................................... 217
  24.2 Variables ............................................... 223
  24.3 Class Element .......................................... 226
    24.3.1 Methods ........................................... 226
    24.3.2 Instance Variables ................................ 226
  24.4 Class Token ............................................ 226
<table>
<thead>
<tr>
<th>Section</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>28.2</td>
<td>Variables</td>
<td>251</td>
</tr>
<tr>
<td>28.3</td>
<td>Class OptimizedReporter</td>
<td>251</td>
</tr>
<tr>
<td>28.3.1</td>
<td>Methods</td>
<td>251</td>
</tr>
<tr>
<td>28.3.2</td>
<td>Class Variables</td>
<td>252</td>
</tr>
<tr>
<td>28.3.3</td>
<td>Instance Variables</td>
<td>252</td>
</tr>
<tr>
<td>28.4</td>
<td>Class ParsedRstDocstring</td>
<td>252</td>
</tr>
<tr>
<td>28.4.1</td>
<td>Methods</td>
<td>252</td>
</tr>
<tr>
<td>28.4.2</td>
<td>Instance Variables</td>
<td>254</td>
</tr>
<tr>
<td>28.5</td>
<td>Class _EpydocReader</td>
<td>254</td>
</tr>
<tr>
<td>28.5.1</td>
<td>Methods</td>
<td>255</td>
</tr>
<tr>
<td>28.5.2</td>
<td>Class Variables</td>
<td>255</td>
</tr>
<tr>
<td>28.6</td>
<td>Class _DocumentPseudoWriter</td>
<td>256</td>
</tr>
<tr>
<td>28.6.1</td>
<td>Methods</td>
<td>256</td>
</tr>
<tr>
<td>28.6.2</td>
<td>Class Variables</td>
<td>256</td>
</tr>
<tr>
<td>28.6.3</td>
<td>Instance Variables</td>
<td>257</td>
</tr>
<tr>
<td>28.7</td>
<td>Class _SummaryExtractor</td>
<td>257</td>
</tr>
<tr>
<td>28.7.1</td>
<td>Methods</td>
<td>257</td>
</tr>
<tr>
<td>28.7.2</td>
<td>Class Variables</td>
<td>258</td>
</tr>
<tr>
<td>28.8</td>
<td>Class _TermsExtractor</td>
<td>258</td>
</tr>
<tr>
<td>28.8.1</td>
<td>Methods</td>
<td>258</td>
</tr>
<tr>
<td>28.8.2</td>
<td>Class Variables</td>
<td>259</td>
</tr>
<tr>
<td>28.8.3</td>
<td>Instance Variables</td>
<td>259</td>
</tr>
<tr>
<td>28.9</td>
<td>Class _SplitFieldsTranslator</td>
<td>259</td>
</tr>
<tr>
<td>28.9.1</td>
<td>Methods</td>
<td>259</td>
</tr>
<tr>
<td>28.9.2</td>
<td>Class Variables</td>
<td>260</td>
</tr>
<tr>
<td>28.9.3</td>
<td>Instance Variables</td>
<td>260</td>
</tr>
<tr>
<td>28.10</td>
<td>Class _EpydocDocumentClass</td>
<td>260</td>
</tr>
<tr>
<td>28.10.1</td>
<td>Methods</td>
<td>260</td>
</tr>
<tr>
<td>28.10.2</td>
<td>Class Variables</td>
<td>261</td>
</tr>
<tr>
<td>28.11</td>
<td>Class _EpydocLaTeXTranslator</td>
<td>261</td>
</tr>
<tr>
<td>28.11.1</td>
<td>Methods</td>
<td>261</td>
</tr>
<tr>
<td>28.11.2</td>
<td>Class Variables</td>
<td>263</td>
</tr>
<tr>
<td>28.12</td>
<td>Class _EpydocHTMLTranslator</td>
<td>263</td>
</tr>
<tr>
<td>28.12.1</td>
<td>Methods</td>
<td>263</td>
</tr>
<tr>
<td>28.12.2</td>
<td>Class Variables</td>
<td>265</td>
</tr>
<tr>
<td>28.13</td>
<td>Class dotgraph</td>
<td>266</td>
</tr>
<tr>
<td>28.13.1</td>
<td>Methods</td>
<td>266</td>
</tr>
<tr>
<td>28.13.2</td>
<td>Class Variables</td>
<td>267</td>
</tr>
<tr>
<td>28.13.3</td>
<td>Instance Variables</td>
<td>267</td>
</tr>
</tbody>
</table>

29 Package epydoc.test

29.1 Functions                                             268

30 Module epydoc.test.util

30.1 Functions                                             269

31 Module epydoc.util

31.1 Functions                                             271
31.2 Variables                                             272
31.3 Class RunSubprocessError                              273
| 31.3.1 | Methods                                         | 273 |
| 31.3.2 | Properties                                      | 273 |
31.4 Class TerminalController                             273
31.4.1 Methods ................................................................. 274
31.4.2 Class Variables .................................................... 274

Index .............................................................................. 277
1 Package epydoc

Automatic Python reference documentation generator. Epydoc processes Python modules and docstrings to generate formatted API documentation, in the form of HTML pages. Epydoc can be used via a command-line interface (epydoc.cli) and a graphical interface (epydoc.gui). Both interfaces let the user specify a set of modules or other objects to document, and produce API documentation using the following steps:

1. Extract basic information about the specified objects, and objects that are related to them (such as the values defined by a module). This can be done via introspection, parsing, or both:
   - *Introspection* imports the objects, and examines them directly using Python’s introspection mechanisms.
   - *Parsing* reads the Python source files that define the objects, and extracts information from those files.

2. Combine and process that information.
   - *Merging*: Merge the information obtained from introspection & parsing each object into a single structure.
   - *Linking*: Replace any “pointers” that were created for imported variables with the documentation that they point to.
   - *Naming*: Assign unique canonical names to each of the specified objects, and any related objects.
   - *Docstrings*: Parse the docstrings of each of the specified objects.
   - *Inheritance*: Add variables to classes for any values that they inherit from their base classes.

3. Generate output. Output can be generated in a variety of formats:
   - An HTML webpage.
   - A LaTeX document (which can be rendered as a PDF file)
   - A plaintext description.
Package Organization

The epydoc package contains the following subpackages and modules:
The user interfaces are provided by the `gui` and `cli` modules. The `apidoc` module defines the basic data types used to record information about Python objects. The programmatic interface to epydoc is provided by `docbuilder`. Docstring markup parsing is handled by the `markup` package, and output generation is handled by the `docwriter` package. See the submodule list for more information about the submodules and subpackages.

**Author:** Edward Loper

**Requires:** Python 2.3+

**Version:** 3.0.1

**See Also:** The epydoc webpage, The epytext markup language manual

**To Do:**
- Create a better default top_page than trees.html.
- Fix trees.html to work when documenting non-top-level modules/packages
- Implement @include
- Optimize epytext
- More doctests
• When introspecting, limit how much introspection you do (eg, don’t construct docs for imported
modules’ vars if it’s not necessary)

Bug: UserDict.* is interpreted as imported .. why??

License: IBM Open Source License

Copyright: © 2006 Edward Loper

Contributors (Alphabetical Order):
• Glyph Leikowitz
• Edward Loper
• Bruce Mitchener
• Jeff O’Halloran
• Simon Pamies
• Christian Reis
• Daniele Varrazzo
• Jonathan Guyer

1.1 Variables

DEBUG
True if debugging is turned on.

Value: True

__author__
The primary author of epydoc

Value: ‘Edward Loper <edloper@gradient.cis.upenn.edu>’

__license__
The license governing the use and distribution of epydoc

Value: ‘IBM Open Source License’

__url__
The URL for epydoc’s homepage

Value: ‘http://epydoc.sourceforge.net’
<table>
<thead>
<tr>
<th><strong>version</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td>The version of epydoc</td>
</tr>
<tr>
<td>Value: 3.0.1</td>
</tr>
</tbody>
</table>
2 Module epydoc.apidoc

Classes for encoding API documentation about Python programs. These classes are used as a common representation for combining information derived from introspection and from parsing.

The API documentation for a Python program is encoded using a graph of APIDoc objects, each of which encodes information about a single Python variable or value. APIDoc has two direct subclasses: VariableDoc, for documenting variables; and ValueDoc, for documenting values. The ValueDoc class is subclassed further, to define the different pieces of information that should be recorded about each value type:

The distinction between variables and values is intentionally made explicit. This allows us to distinguish information about a variable itself (such as whether it should be considered 'public' in its containing namespace) from information about the value it contains (such as what type the value has). This distinction is also important because several variables can contain the same value: each variable should be described by a separate VariableDoc; but we only need one ValueDoc, since they share a single value.

To Do: Add a cache to canonical name lookup?

2.1 Functions

reachable_valdocs(root, **filters)

Return a list of all ValueDocs that can be reached, directly or indirectly from the given root list of ValueDocs.

Parameters

filters: A set of filters that can be used to prevent reachable_valdocs from following specific link types when looking for ValueDocs that can be reached from the root set. See APIDoc.apidoc_links for a more complete description.

.flatten(lst, out=None)

Return a flattened version of lst.
 variables

**pp_apidoc**(api_doc, doublespace=0, depth=5, exclude=(),
include=(), backpointers=None)

**Parameters**

- doublespace: If true, then extra lines will be inserted to make the output more readable.
- depth: The maximum depth that pp_apidoc will descend into descendent VarDocs. To put no limit on depth, use depth=-1.
- exclude: A list of names of attributes whose values should not be shown.
- backpointers: For internal use.

**Return Value**

A multiline pretty-printed string representation for the given APIDoc.

**pp_list**(api_doc, items, doublespace, depth, exclude, include,
backpointers, is_last)

**pp_dict**(api_doc, dict, doublespace, depth, exclude, include,
backpointers, is_last)

**pp_apidoc**(api_doc, val, doublespace, depth, exclude, include,
backpointers, is_last)

**pp_val**(api_doc, val, doublespace, depth, exclude, include,
backpointers)

### 2.2 Variables

**UNKNOWN**

A special value used to indicate that a given piece of information about an object is unknown. This is used as the default value for all instance variables.

Value: _Sentinel('UNKNOWN')

### 2.3 Class DottedName

A sequence of identifiers, separated by periods, used to name a Python variable, value, or argument. The identifiers that make up a dotted name can be accessed using the indexing operator:

```python
>>> name = DottedName('epydoc', 'api_doc', 'DottedName')
>>> print name
epydoc.apidoc.DottedName
>>> name[1]
```
2.3.1 Methods

```python
__init__(self, *pieces, **options)
```

Construct a new dotted name from the given sequence of pieces, each of which can be either a string or a DottedName. Each piece is divided into a sequence of identifiers, and these sequences are combined together (in order) to form the identifier sequence for the new DottedName. If a piece contains a string, then it is divided into substrings by splitting on periods, and each substring is checked to see if it is a valid identifier.

As an optimization, `pieces` may also contain a single tuple of values. In that case, that tuple will be used as the DottedName’s identifiers; it will not be checked to see if it’s valid.

**Parameters**

- `strict`: if true, then raise an `InvalidDottedName` if the given name is invalid.

```python
__repr__(self)
```

```python
__str__(self)
```

Return the dotted name as a string formed by joining its identifiers with periods:

```python
>>> print DottedName('epydoc', 'api_doc', DottedName)
epydoc.apidoc.DottedName
```

```python
__add__(self, other)
```

Return a new DottedName whose identifier sequence is formed by adding `other`’s identifier sequence to `self`’s.

```python
__radd__(self, other)
```

Return a new DottedName whose identifier sequence is formed by adding `self`’s identifier sequence to `other`’s.

```python
__getitem__(self, i)
```

Return the i-th identifier in this DottedName. If i is a non-empty slice, then return a DottedName built from the identifiers selected by the slice. If i is an empty slice, return an empty list (since empty DottedNames are not valid).

```python
__hash__(self)
```

```python
__cmp__(self, other)
```

Compare this dotted name to `other`. Two dotted names are considered equal if their identifier subsequences are equal. Ordering between dotted names is lexicographic, in order of identifier from left to right.
**_len_(self)**

Return the number of identifiers in this dotted name.

**container(self)**

Return the DottedName formed by removing the last identifier from this dotted name’s identifier sequence. If this dotted name only has one name in its identifier sequence, return `None` instead.

**dominates(self, name, strict=False)**

Return true if this dotted name is equal to a prefix of `name`. If `strict` is true, then also require that `self!=name`.

```python
>>> DottedName('a.b').dominates(DottedName('a.b.c.d'))
True
```

**contextualize(self, context)**

If `self` and `context` share a common ancestor, then return a name for `self`, relative to that ancestor. If they do not share a common ancestor (or if `context` is `UNKNOWN`), then simply return `self`.

This is used to generate shorter versions of dotted names in cases where users can infer the intended target from the context.

**Parameters**
- `context` *(type=DottedName)*

**Return Value**
- `DottedName`

### 2.3.2 Class Variables

**UNREACHABLE**

Value: `'??'`

**_IDENTIFIER_RE**

Value: `re.compile(r'(?x)(?:script-)?\w+\'?(?:-\d+)?$')`

**_ok_identifiers**

A cache of identifier strings that have been checked against `_IDENTIFIER_RE` and found to be acceptable.

Value: `set(['??', '??-1', '??-10', '??-11', '??-12', '??-13', '??...'])`
2.4 Class DottedName.InvalidDottedName

An exception raised by the DottedName constructor when one of its arguments is not a valid dotted name.

2.4.1 Methods

Inherited from exceptions.ValueError: __init__(), __new__()  

Inherited from exceptions.BaseException: __delattr__() , __getattr__() , __getattribute__() , __getitem__() , __getslice__() , __reduce__() , __reduce_ex__() , __repr__() , __setattr__() , __setstate__() , __str__()  

2.4.2 Properties

Inherited from exceptions.BaseException: args, message

2.5 Class _Sentinel

A unique value that won’t compare equal to any other value. This class is used to create UNKNOWN.

2.5.1 Methods

__init__(self, name)  

__repr__(self)  

__nonzero__(self)
2.6 Class APIDoc

API documentation information for a single element of a Python program. APIDoc itself is an abstract base class; subclasses are used to specify what information should be recorded about each type of program element. In particular, APIDoc has two direct subclasses, `VariableDoc` for documenting variables and `ValueDoc` for documenting values; and the `ValueDoc` class is subclassed further for different value types.

Each APIDoc subclass specifies the set of attributes that should be used to record information about the corresponding program element type. The default value for each attribute is stored in the class; these default values can then be overridden with instance variables. Most attributes use the special value `UNKNOWN` as their default value, to indicate that the correct value for that attribute has not yet been determined. This makes it easier to merge two APIDoc objects that are documenting the same element (in particular, to merge information about an element that was derived from parsing with information that was derived from introspection).

For all attributes with boolean values, use only the constants `True` and `False` to designate true and false. In particular, do not use other values that evaluate as true or false, such as 2 or 0. This restriction makes it easier to handle `UNKNOWN` values. For example, to test if a boolean attribute is `True` or `UNKNOWN`, use `attrib in (True, UNKNOWN)` or `attrib is not False`.

Two APIDoc objects describing the same object can be merged, using the method `merge_and_overwrite(other)`. After two APIDocs are merged, any changes to one will be reflected in the other. This is accomplished by setting the two APIDoc objects to use a shared instance dictionary. See the documentation for `merge_and_overwrite` for more information, and some important caveats about hashing.

2.6.1 Methods

```
__init__(self, **kwargs)
```

Construct a new APIDoc object. Keyword arguments may be used to initialize the new APIDoc’s attributes.

**Raises**

- `TypeError` If a keyword argument is specified that does not correspond to a valid attribute for this (sub)class of APIDoc.

**Overrides**: `object.__init__`
### `_debug_setattr(self, attr, val)`
Modify an APIDoc's attribute. This is used when `epydoc.DEBUG` is true, to make sure we don’t accidentally set any inappropriate attributes on APIDoc objects.

**Raises**

`AttributeError` If `attr` is not a valid attribute for this (sub)class of APIDoc. (`attr` is considered a valid attribute iff `self.__class__` defines an attribute with that name.)

### `_setattr__(self, attr, val)`
Modify an APIDoc's attribute. This is used when `epydoc.DEBUG` is true, to make sure we don’t accidentally set any inappropriate attributes on APIDoc objects.

**Raises**

`AttributeError` If `attr` is not a valid attribute for this (sub)class of APIDoc. (`attr` is considered a valid attribute iff `self.__class__` defines an attribute with that name.)

**Overrides:** `object.__setattr__`

### `__repr__(self)`

**Overrides:** `object.__repr__` *(inherited documentation)*

### `pp(self, doublespace=0, depth=5, exclude=(), include=())`
Return a pretty-printed string representation for the information contained in this APIDoc.

### `_str__(self, doublespace=0, depth=5, exclude=(), include=())`
Return a pretty-printed string representation for the information contained in this APIDoc.

**Overrides:** `object.__str__`

### `specialize_to(self, cls)`
Change `self`'s class to `cls`. `cls` must be a subclass of `self`'s current class. For example, if a generic `ValueDoc` was created for a value, and it is determined that the value is a routine, you can update its class with:

```
>>> valdoc.specialize_to(RoutineDoc)
```

### `_hash__(self)`

**Overrides:** `object.__hash__` *(inherited documentation)*

### `_cmp__(self, other)`
Class APIDoc

Module epydoc.apidoc

**is_detailed** *(self)*

Does this object deserve a box with extra details?

**Return Value**

True if the object needs extra details, else False. *(type=bool)*

**merge_and_overwrite** *(self, other, ignore_hash_conflict=False)*

Combine `self` and `other` into a *merged object*, such that any changes made to one will affect the other. Any attributes that `other` had before merging will be discarded. This is accomplished by copying `self.__dict__` over `other.__dict__` and `self.__class__` over `other.__class__`.

Care must be taken with this method, since it modifies the hash value of `other`. To help avoid the problems that this can cause, `merge_and_overwrite` will raise an exception if `other` has ever been hashed, unless `ignore_hash_conflict` is True. Note that adding `other` to a dictionary, set, or similar data structure will implicitly cause it to be hashed. If you do set `ignore_hash_conflict` to True, then any existing data structures that rely on `other`'s hash staying constant may become corrupted.

**Return Value**

`self`

**Raises**

* ValueError If `other` has ever been hashed.

**apidoc_links** *(self, **filters)*

Return a list of all APIDocs that are directly linked from this APIDoc (i.e., are contained or pointed to by one or more of this APIDoc’s attributes.)

Keyword argument `filters` can be used to selectively exclude certain categories of attribute value. For example, using `includes=False` will exclude variables that were imported from other modules; and `subclasses=False` will exclude subclasses. The filter categories currently supported by epydoc are:

- **imports**: Imported variables.
- **packages**: Containing packages for modules.
- **submodules**: Contained submodules for packages.
- **bases**: Bases for classes.
- **subclasses**: Subclasses for classes.
- **variables**: All variables.
- **private**: Private variables.
- **overrides**: Points from class variables to the variables they override. This filter is False by default.

### 2.6.2 Class Variables

**__has BEEN HASHED**

True iff `self.__hash__() has ever been called.

**Value**: False
Class APIDoc

__mergeset

The set of all APIDoc objects that have been merged with this APIDoc (using merge_and_overwrite()). Each APIDoc in this set shares a common instance dictionary (__dict__).

Value: None

2.6.3 Instance Variables

Docstrings

docstring

The documented item's docstring.

Type: string or None

Value: _Sentinel('UNKNOWN')

docstring_lineno

The line number on which the documented item's docstring begins.

Type: int

Value: _Sentinel('UNKNOWN')

Information Extracted from Docstrings

descr

A description of the documented item, extracted from its docstring.

Type: ParsedDocstring

Value: _Sentinel('UNKNOWN')

summary

A summary description of the documented item, extracted from its docstring.

Type: ParsedDocstring

Value: _Sentinel('UNKNOWN')
other_docs
A flag indicating if the entire docstring body (except tags if any) is entirely included in the summary.

Type: bool
Value: _Sentinel('UNKNOWN')

metadata
Metadata about the documented item, extracted from fields in its docstring. Currently this is encoded as a list of tuples (field, arg, descr). But that may change.

Type: (str, str, ParsedDocstring)
Value: _Sentinel('UNKNOWN')

extra_docstring_fields
A list of new docstring fields tags that are defined by the documented item’s docstring. These new field tags can be used by this item or by any item it contains.

Type: DocstringField
Value: _Sentinel('UNKNOWN')

Source Information

docs_extracted_by
Information about where the information contained by this APIDoc came from. Can be one of ‘parser’, ‘introspector’, or ‘both’.

Type: str
Value: _Sentinel('UNKNOWN')

2.7 Class VariableDoc

API documentation information about a single Python variable.

Note: The only time a VariableDoc will have its own docstring is if that variable was created using an assignment statement, and that assignment statement had a docstring-comment or was followed by a pseudo-docstring.
2.7.1 Methods

__init__(self, **kwargs)
Construct a new APIDoc object. Keyword arguments may be used to initialize the new APIDoc’s attributes.

Raises
TypeError If a keyword argument is specified that does not correspond to a valid attribute for this (sub)class of APIDoc.

Overrides: epydoc.apidoc.APIDoc.__init__ (inherited documentation)

__repr__(self)

Overrides: epydoc.apidoc.APIDoc.__repr__

_get_defining_module(self)

apidoc_links(self, **filters)
Return a list of all APIDocs that are directly linked from this APIDoc (i.e., are contained or pointed to by one or more of this APIDoc’s attributes.)

Keyword argument filters can be used to selectivly exclude certain categories of attribute value. For example, using includes=False will exclude variables that were imported from other modules; and subclasses=False will exclude subclasses. The filter categories currently supported by epydoc are:

- imports: Imported variables.
- packages: Containing packages for modules.
- submodules: Contained submodules for packages.
- bases: Bases for classes.
- subclasses: Subclasses for classes.
- variables: All variables.
- private: Private variables.
- overrides: Points from class variables to the variables they override. This filter is False by default.

Overrides: epydoc.apidoc.APIDoc.apidoc_links (inherited documentation)

is_detailed(self)

Does this object deserve a box with extra details?

Return Value
True if the object needs extra details, else False.(type=bool)

Overrides: epydoc.apidoc.APIDoc.is_detailed (inherited documentation)
2.7.2 Properties

**defining_module**
A read-only property that can be used to get the variable’s defining module. This is defined as the defining module of the variable’s container.

Get: epydoc.apidoc.VariableDoc._get_defining_module()

2.7.3 Instance Variables

**Basic Variable Information**

**name**
The name of this variable in its containing namespace.

Type: str
Value: _Sentinel('UNKNOWN')

**container**
API documentation for the namespace that contains this variable.

Type: ValueDoc
Value: _Sentinel('UNKNOWN')

**canonical_name**
A dotted name that serves as a unique identifier for this VariableDoc. It should be formed by concatenating the VariableDoc’s container with its name.

Type: DottedName
Value: _Sentinel('UNKNOWN')

**value**
The API documentation for this variable’s value.

Type: ValueDoc
Value: _Sentinel('UNKNOWN')

**Information Extracted from Docstrings**
**type_descr**

A description of the variable’s expected type, extracted from its docstring.

**Type**: ParsedDocstring  
**Value**: `_Sentinel('UNKNOWN')`

Inherited from epydoc.apidoc.APIDoc (Section 2.6, p. 19): descr, extra_docstring_fields, metadata, other_docs, summary

### Information about Imported Variables

**imported_from**

The fully qualified dotted name of the variable that this variable’s value was imported from. This attribute should only be defined if `is_instvar` is true.

**Type**: DottedName  
**Value**: `_Sentinel('UNKNOWN')`

**is_imported**

Was this variable’s value imported from another module? (Exception: variables that are explicitly included in `__all__` have `is_imported` set to `False`, even if they are in fact imported.)

**Type**: bool  
**Value**: `_Sentinel('UNKNOWN')`

### Information about Variables in Classes

**is_instvar**

If true, then this variable is an instance variable; if false, then this variable is a class variable. This attribute should only be defined if the containing namespace is a class.

**Type**: bool  
**Value**: `_Sentinel('UNKNOWN')`

**overrides**

The API documentation for the variable that is overridden by this variable. This attribute should only be defined if the containing namespace is a class.

**Type**: VariableDoc  
**Value**: `_Sentinel('UNKNOWN')`

### Flags
**is_alias**

Is this variable an alias for another variable with the same value? If so, then this variable will be dispreferred when assigning canonical names.

**Type:** bool

**Value:** `_Sentinel('UNKNOWN')`

**is_public**

Is this variable part of its container’s public API?

**Type:** bool

**Value:** `_Sentinel('UNKNOWN')`

---

**Docstrings**

Inherited from epydoc.apidoc.APIDoc (Section 2.6, p. 19): docstring, docstring_lineno

---

**Source Information**

Inherited from epydoc.apidoc.APIDoc (Section 2.6, p. 19): docs_extracted_by

---

## 2.8 Class ValueDoc

[Diagram of the class hierarchy]


API documentation information about a single Python value.
2.8.1 Methods

__repr__(self)

Overrides: epydoc.apidoc.APIDoc.__repr__

__setstate__(self, state)

__getstate__(self)

State serializer for the pickle module. This is necessary because sometimes the pyval attribute contains an un-pickleable value.

apidoc_links(self, **filters)

Return a list of all APIDocs that are directly linked from this APIDoc (i.e., are contained or pointed to by one or more of this APIDoc’s attributes.)

Keyword argument filters can be used to selectively exclude certain categories of attribute value. For example, using includes=False will exclude variables that were imported from other modules; and subclasses=False will exclude subclasses. The filter categories currently supported by epydoc are:

- imports: Imported variables.
- packages: Containing packages for modules.
- submodules: Contained submodules for packages.
- bases: Bases for classes.
- subclasses: Subclasses for classes.
- variables: All variables.
- private: Private variables.
- overrides: Points from class variables to the variables they override. This filter is False by default.

Overrides: epydoc.apidoc.APIDoc.apidoc_links (inherited documentation)

Inherited from epydoc.apidoc.APIDoc(Section 2.6, p. 19): __cmp__(), __hash__(), __init__(),
__setattr__() , __str__(), __debug_setattr__(), is_detailed(), merge_and_overwrite(), pp(), specialize_to()

Value Representation

pyval_repr(self)

Return a formatted representation of the Python object described by this ValueDoc. This representation may include data from introspection or parsing, and is authoritative as 'the best way to represent a Python value.' Any lines that go beyond REPR_LINELEN characters will be wrapped; and if the representation as a whole takes more than REPR_MAXLINES lines, then it will be truncated (with an ellipsis marker). This function will never return UNKNOWN or None.

Return Value

ColorizedPyvalRepr
**summary_pyval_repr**(self, *max_len*=None)

Return a single-line formatted representation of the Python object described by this `ValueDoc`. This representation may include data from introspection or parsing, and is authoritative as ‘the best way to summarize a Python value.’ If the representation takes more then `SUMMARY_REPR_LINELEN` characters, then it will be truncated (with an ellipsis marker). This function will never return `UNKNOWN` or `None`.

**Return Value**

`ColorizedPyvalRepr`

## 2.8.2 Class Variables

### Value Representation

#### REPR_MAXLINES

The maximum number of lines of text that should be generated by `pyval_repr()`. If the string representation does not fit in this number of lines, an ellipsis marker (...) will be placed at the end of the formatted representation.

**Value:** 5

#### REPR_LINELEN

The maximum number of characters for lines of text that should be generated by `pyval_repr()`. Any lines that exceed this number of characters will be line-wrapped; The ← symbol will be used to indicate that the line was wrapped.

**Value:** 75

#### SUMMARY_REPR_LINELEN

The maximum number of characters for the single-line text representation generated by `summary_pyval_repr()`. If the value’s representation does not fit in this number of characters, an ellipsis marker (...) will be placed at the end of the formatted representation.

**Value:** 75

#### REPR_MIN_SCORE

The minimum score that a value representation based on `pyval` should have in order to be used instead of `parse_repr` as the canonical representation for this `ValueDoc`’s value.

**Value:** 0
2.8.3 Instance Variables

**canonical_name**
A dotted name that serves as a unique identifier for this ValueDoc’s value. If the value can be reached using a single sequence of identifiers (given the appropriate imports), then that sequence of identifiers is used as its canonical name. If the value can be reached by multiple sequences of identifiers (i.e., if it has multiple aliases), then one of those sequences of identifiers is used. If the value cannot be reached by any sequence of identifiers (e.g., if it was used as a base class but then its variable was deleted), then its canonical name will start with ‘??’. If necessary, a dash followed by a number will be appended to the end of a non-reachable identifier to make its canonical name unique.

When possible, canonical names are chosen when new ValueDocs are created. However, this is sometimes not possible. If a canonical name cannot be chosen when the ValueDoc is created, then one will be assigned by `assign_canonical_names()`.

Type: DottedName
Value: `_Sentinel('UNKNOWN')`

**toktree**
This is currently used to extract values from `_all_` etc, in the docparser module; maybe I should specialize process_assignment and extract it there? Although, for `_all_` it’s not clear where I’d put the value, since I just use it to set private/public/imported attrs on other vars (that might not exist yet at the time.)

Value: `_Sentinel('UNKNOWN')`

### Value Representation

**pyval**
A pointer to the actual Python object described by this ValueDoc. This is used to display the value (e.g., when describing a variable.) Use `pyval_repr()` to generate a plaintext string representation of this value.

Type: Python object
Value: `_Sentinel('UNKNOWN')`

**parse_repr**
A text representation of this value, extracted from parsing its source code. This representation may not accurately reflect the actual value (e.g., if the value was modified after the initial assignment).

Type: unicode
Value: `_Sentinel('UNKNOWN')`

### Context

30
### defining_module

The documentation for the module that defines this value. This is used, e.g., to lookup the appropriate markup language for docstrings. For a ModuleDoc, defining_module should be self.

**Type:** ModuleDoc  
**Value:** `_Sentinel('UNKNOWN')`

### Information about Imported Variables

#### proxy_for

If proxy_for is not None, then this value was imported from another file. proxy_for is the dotted name of the variable that this value was imported from. If that variable is documented, then its value may contain more complete API documentation about this value. The proxy_for attribute is used by the source code parser to link imported values to their source values (in particular, for base classes). When possible, these proxy ValueDocs are replaced by the imported value's ValueDoc by link_imports().

**Type:** DottedName  
**Value:** None

### Docstrings

Inherited from epydoc.apidoc.APIDoc (Section 2.6, p. 19): docstring, docstring_lineno

### Information Extracted from Docstrings

Inherited from epydoc.apidoc.APIDoc (Section 2.6, p. 19): descr, extra_docstring_fields, metadata, other_docs, summary

### Source Information

Inherited from epydoc.apidoc.APIDoc (Section 2.6, p. 19): docs_extracted_by

---

31
2.9 Class GenericValueDoc

API documentation about a 'generic' value, i.e., one that does not have its own docstring or any information other than its value and parse representation. GenericValueDocs do not get assigned canonical names.

2.9.1 Methods

**is_detailed**(self)

Does this object deserve a box with extra details?

**Return Value**

True if the object needs extra details, else False. *(type=bool)*

**Overrides:** epydoc.apidoc.APIDoc.is_detailed *(inherited documentation)*

Inherited from epydoc.apidoc.ValueDoc *(Section 2.8, p. 27)*: __getstate__(), __repr__(), __setstate__(), apidoc_links()

Inherited from epydoc.apidoc.APIDoc *(Section 2.6, p. 19)*: __cmp__(), __hash__(), __init__(), __setattr__(), __str__(), __debug_setattr__(), merge_and_overwrite(), pp(), specialize_to()

**Value Representation**

Inherited from epydoc.apidoc.ValueDoc *(Section 2.8, p. 27)*: pyval_repr(), summary_pyval_repr()
2.9.2 Class Variables

**canonical_name**
A dotted name that serves as a unique identifier for this `ValueDoc`'s value. If the value can be reached using a single sequence of identifiers (given the appropriate imports), then that sequence of identifiers is used as its canonical name. If the value can be reached by multiple sequences of identifiers (i.e., if it has multiple aliases), then one of those sequences of identifiers is used. If the value cannot be reached by any sequence of identifiers (e.g., if it was used as a base class but then its variable was deleted), then its canonical name will start with ‘??’. If necessary, a dash followed by a number will be appended to the end of a non-reachable identifier to make its canonical name unique.

When possible, canonical names are chosen when new `ValueDoc`s are created. However, this is sometimes not possible. If a canonical name can not be chosen when the `ValueDoc` is created, then one will be assigned by `assign_canonical_names()`.

Type: DottedName
Value: None

### Value Representation
Inherited from `epydoc.apidoc.ValueDoc` (Section 2.8, p. 27): `REPR_LINELEN`, `REPR_MAXLINES`, `REPR_MIN_SCORE`, `SUMMARY_REPR_LINELEN`

2.9.3 Instance Variables

Inherited from `epydoc.apidoc.ValueDoc` (Section 2.8, p. 27): `toktree`

### Value Representation
Inherited from `epydoc.apidoc.ValueDoc` (Section 2.8, p. 27): `parse_repr`, `pyval`

### Context
Inherited from `epydoc.apidoc.ValueDoc` (Section 2.8, p. 27): `defining_module`

### Information about Imported Variables
Inherited from `epydoc.apidoc.ValueDoc` (Section 2.8, p. 27): `proxy_for`

### Docstrings
Inherited from `epydoc.apidoc.APIDoc` (Section 2.6, p. 19): `docstring`, `docstring_lineno`
2.10 Class NamespaceDoc

Known Subclasses: epydoc.apidoc.ClassDoc, epydoc.apidoc.ModuleDoc

API documentation information about a single Python namespace value. (I.e., a module or a class).

2.10.1 Methods

\_\_init\_\_(\textit{self} , **\textit{kwargs} )

Construct a new APIDoc object. Keyword arguments may be used to initialize the new APIDoc’s attributes.

\textbf{Raises}

\texttt{TypeError} If a keyword argument is specified that does not correspond to a valid attribute for this (sub)class of APIDoc.

\textbf{Overrides:} epydoc.apidoc.APIDoc.\_\_init\_\_ (inherited documentation)

\textbf{is\_detailed}(\textit{self})

Does this object deserve a box with extra details?

\textbf{Return Value}

True if the object needs extra details, else False. (\texttt{type=bool})

\textbf{Overrides:} epydoc.apidoc.APIDoc.is\_detailed (inherited documentation)
apidoc_links(self, **filters)

Return a list of all APIDocs that are directly linked from this APIDoc (i.e., are contained or pointed to by one or more of this APIDoc’s attributes.)

Keyword argument filters can be used to selectively exclude certain categories of attribute value. For example, using includes=False will exclude variables that were imported from other modules; and subclasses=False will exclude subclasses. The filter categories currently supported by epydoc are:

- imports: Imported variables.
- packages: Containing packages for modules.
- submodules: Contained submodules for packages.
- bases: Bases for classes.
- subclasses: Subclasses for classes.
- variables: All variables.
- private: Private variables.
- overrides: Points from class variables to the variables they override. This filter is False by default.

Overrides: epydoc.apidoc.APIDoc.apidoc_links (inherited documentation)

init_sorted_variables(self)

Initialize the sorted_variables attribute, based on the variables and sort_spec attributes. This should usually be called after all variables have been added to variables (including any inherited variables for classes).

init_variable_groups(self)

Initialize the variable_groups attribute, based on the sorted_variables and group_specs attributes.

group_names(self)

Return a list of the group names defined by this namespace, in the order in which they should be listed, with no duplicates.

_init_grouping(self, elts)

Divide a given a list of APIDoc objects into groups, as specified by self.group_specs.

Parameters

- elts: A list of tuples (name, apidoc).

Return Value

A list of tuples (groupname, elts), where groupname is the name of a group and elts is a list of APIDocs in that group. The first tuple has name m, and is used for ungrouped elements. The remaining tuples are listed in the order that they appear in self.group_specs. Within each tuple, the elements are listed in the order that they appear in api_docs.
Class NamespaceDoc

Module epydoc.apidoc

**report_unused_groups**(*self*)

Issue a warning for any @group items that were not used by _init_grouping()_.

Inherited from epydoc.apidoc.ValueDoc (Section 2.8, p. 27): __getstate__(), __repr__(), __setstate__()

Inherited from epydoc.apidoc.APIDoc (Section 2.6, p. 19): __cmp__(), __hash__(), __setattr__(), __str__(), __debug_setattr__(), merge_and_overwrite(), pp(), specialize_to()

## Value Representation

Inherited from epydoc.apidoc.ValueDoc (Section 2.8, p. 27): pyval_repr(), summary_pyval_repr()

### 2.10.2 Class Variables

## Value Representation

Inherited from epydoc.apidoc.ValueDoc (Section 2.8, p. 27): REPR_LINELEN, REPR_MAXLINES, REPR_MIN_SCORE, SUMMARY_REPR_LINELEN

### 2.10.3 Instance Variables

Inherited from epydoc.apidoc.ValueDoc (Section 2.8, p. 27): canonical_name, toktree

## Information about Variables

**variables**

The contents of the namespace, encoded as a dictionary mapping from identifiers to VariableDocs. This dictionary contains all names defined by the namespace, including imported variables, aliased variables, and variables inherited from base classes (once inherit_docs() has added them).

**Type:** dict from string to VariableDoc

**Value:** _Sentinel(‘UNKNOWN’)
**sorted_variables**

A list of all variables defined by this namespace, in sorted order. The elements of this list should exactly match the values of `variables`. The sort order for this list is defined as follows:

- Any variables listed in a `@sort` docstring field are listed in the order given by that field.
- These are followed by any variables that were found while parsing the source code, in the order in which they were defined in the source file.
- Finally, any remaining variables are listed in alphabetical order.

**Type:** list of `VariableDoc`

**Value:** `_Sentinel('UNKNOWN')`

**sort_spec**

The order in which variables should be listed, encoded as a list of names. Any variables whose names are not included in this list should be listed alphabetically, following the variables that are included.

**Type:** list of str

**Value:** `_Sentinel('UNKNOWN')`

**group_specs**

The groups that are defined by this namespace’s docstrings. `group_specs` is encoded as an ordered list of tuples `(group_name, elt_names)`, where `group_name` is the name of a group and `elt_names` is a list of element names in that group. (An element can be a variable or a submodule.) A '*' in an element name will match any string of characters.

**Type:** list of (str,list)

**Value:** `_Sentinel('UNKNOWN')`

**variable_groups**

A dictionary specifying what group each variable belongs to. The keys of the dictionary are group names, and the values are lists of `VariableDoc`s. The order that groups should be listed in should be taken from `group_specs`.

**Type:** dict from str to list of `VariableDoc`

**Value:** `_Sentinel('UNKNOWN')`

---

**Value Representation**

Inherited from `epydoc.api.doc.ValueDoc(Section 2.8, p. 27): parse_repr, pyval`

---

**Context**
## Information about Imported Variables

Inherited from `epydoc.apidoc.ValueDoc` (*Section 2.8, p. 27*): `defining_module`

## Docstrings

Inherited from `epydoc.apidoc.APIDoc` (*Section 2.6, p. 19*): `docstring, docstring_lineno`

## Information Extracted from Docstrings

Inherited from `epydoc.apidoc.APIDoc` (*Section 2.6, p. 19*): `descr, extra_docstring_fields, metadata, other_docs, summary`

## Source Information

Inherited from `epydoc.apidoc.APIDoc` (*Section 2.6, p. 19*): `docs_extracted_by`

### 2.11 Class ModuleDoc

![Diagram of ModuleDoc inheritance]

API documentation information about a single module.
2.11.1 Methods

**apidoc_links**(self, **filters**)

Return a list of all APIDocs that are directly linked from this APIDoc (i.e., are contained or pointed to by one or more of this APIDoc’s attributes.)

Keyword argument filters can be used to selectively exclude certain categories of attribute value. For example, using includes=False will exclude variables that were imported from other modules; and subclasses=False will exclude subclasses. The filter categories currently supported by epydoc are:

- **imports**: Imported variables.
- **packages**: Containing packages for modules.
- **submodules**: Contained submodules for packages.
- **bases**: Bases for classes.
- **subclasses**: Subclasses for classes.
- **variables**: All variables.
- **private**: Private variables.
- **overrides**: Points from class variables to the variables they override. This filter is False by default.

**Overrides**: epydoc.apidoc.APIDoc.apidoc_links *(inherited documentation)*

**init_submodule_groups**(self)

Initialize the submodule_groups attribute, based on the submodules and group_specs attributes.
select_variables(self, group=None, value_type=None, public=None, imported=None, detailed=None)

Return a specified subset of this module’s sorted_variables list. If value_type is given, then only return variables whose values have the specified type. If group is given, then only return variables that belong to the specified group.

Parameters

value_type: A string specifying the value type for which variables should be returned. Valid values are:

- 'class' - variables whose values are classes or types.
- 'function' - variables whose values are functions.
- 'other' - variables whose values are not classes, exceptions, types, or functions.

(type=string)

group: The name of the group for which variables should be returned. A complete list of the groups defined by this ModuleDoc is available in the group_names instance variable. The first element of this list is always the special group name , which is used for variables that do not belong to any group. (type=string)

detailed: If True (False), return only the variables deserving (not deserving) a detailed informative box. If None, don’t care. (type=bool)

Requires: The sorted_variables, variable_groups, and submodule_groups attributes must be initialized before this method can be used. See init_sorted_variables(), init_variable_groups(), and init_submodule_groups().

Inherited from epydoc.apidoc.NamespaceDoc(Section 2.10, p. 34): __init__(), __init_grouping(),
group_names(), init_sorted_variables(), init_variable_groups(), is_detailed(), report_unused_groups()

Inherited from epydoc.apidoc.ValueDoc(Section 2.8, p. 27): __getstate__(), __repr__(), __setstate__()

Inherited from epydoc.apidoc.APIDoc(Section 2.6, p. 19): __cmp__(), __hash__(), __setattr__(),
__str__(), __debug_setattr__(), merge_and_overwrite(), pp(), specialize_to()

Value Representation

Inherited from epydoc.apidoc.ValueDoc(Section 2.8, p. 27): pyval_repr(), summary_pyval_repr()

2.11.2 Class Variables

Value Representation

Inherited from epydoc.apidoc.ValueDoc(Section 2.8, p. 27): REPR_LINELEN, REPR_MAXLINES,
REPR_MIN_SCORE, SUMMARY_REPR_LINELEN
2.11.3 Instance Variables

Inherited from epydoc.apidoc.ValueDoc (Section 2.8, p. 27): canonical_name, toktree

<table>
<thead>
<tr>
<th>Information about the Module</th>
</tr>
</thead>
</table>

**filename**
The name of the file that defines the module.

<table>
<thead>
<tr>
<th>Type: string</th>
</tr>
</thead>
</table>

Value: `_Sentinel('UNKNOWN')`

**docformat**
The markup language used by docstrings in this module.

<table>
<thead>
<tr>
<th>Type: string</th>
</tr>
</thead>
</table>

Value: `_Sentinel('UNKNOWN')`

<table>
<thead>
<tr>
<th>Information about Submodules</th>
</tr>
</thead>
</table>

**submodules**
Modules contained by this module (if this module is a package). (Note: on rare occasions, a module may have a submodule that is shadowed by a variable with the same name.)

<table>
<thead>
<tr>
<th>Type: list of ModuleDoc</th>
</tr>
</thead>
</table>

Value: `_Sentinel('UNKNOWN')`

**submodule_groups**
A dictionary specifying what group each submodule belongs to. The keys of the dictionary are group names, and the values are lists of ModuleDocs. The order that groups should be listed in should be taken from group_specs.

<table>
<thead>
<tr>
<th>Type: dict from str to list of ModuleDoc</th>
</tr>
</thead>
</table>

Value: `_Sentinel('UNKNOWN')`

<table>
<thead>
<tr>
<th>Information about Packages</th>
</tr>
</thead>
</table>

**package**
API documentation for the module’s containing package.

<table>
<thead>
<tr>
<th>Type: ModuleDoc</th>
</tr>
</thead>
</table>

Value: `_Sentinel('UNKNOWN')`
**is_package**

True if this `ModuleDoc` describes a package.

*Type: bool*

*Value: `_Sentinel('UNKNOWN')`*

**path**

If this `ModuleDoc` describes a package, then `path` contains a list of directories that constitute its path (i.e., the value of its `__path__` variable).

*Type: list of str*

*Value: `_Sentinel('UNKNOWN')`*

### Information about Imported Variables

**imports**

A list of the source names of variables imported into this module. This is used to construct import graphs.

*Type: list of DottedName*

*Value: `_Sentinel('UNKNOWN')`*

Inherited from `epydoc.apidoc.ValueDoc (Section 2.8, p. 27)`: proxy_for

### Information about Variables

Inherited from `epydoc.apidoc.NamespaceDoc (Section 2.10, p. 34)`: group_specs, sort_spec, sorted_variables, variable_groups, variables

### Value Representation

Inherited from `epydoc.apidoc.ValueDoc (Section 2.8, p. 27)`: parse_repr, pyval

### Context

Inherited from `epydoc.apidoc.ValueDoc (Section 2.8, p. 27)`: defining_module

### Docstrings

Inherited from `epydoc.apidoc.APIDoc (Section 2.6, p. 19)`: docstring, docstring_lineno

### Information Extracted from Docstrings
2.12 Class ClassDoc

API documentation information about a single class.

2.12.1 Methods

apidoc_links(self, **filters)

Return a list of all APIDocs that are directly linked from this APIDoc (i.e., are contained or pointed to by one or more of this APIDoc’s attributes.)

Keyword argument filters can be used to selectively exclude certain categories of attribute value. For example, using includes=False will exclude variables that were imported from other modules; and subclasses=False will exclude subclasses. The filter categories currently supported by epydoc are:

- imports: Imported variables.
- packages: Containing packages for modules.
- submodules: Contained submodules for packages.
- bases: Bases for classes.
- subclasses: Subclasses for classes.
- variables: All variables.
- private: Private variables.
- overrides: Points from class variables to the variables they override. This filter is False by default.

Overrides: epydoc.apidoc.APIDoc.apidoc_links (inherited documentation)
is_type(self)

is_exception(self)

is_newstyle_class(self)

mro(self, warn_about_bad_bases=False)

_dfs_bases(self, mro, seen, warn_about_bad_bases)

_c3_mro(self, warn_about_bad_bases)

Compute the class precedence list (mro) according to C3.

See Also: http://www.python.org/2.3/mro.html

_report_bad_base(self, base)

_c3_merge(self, seqs)

Helper function for _c3_mro.
**select_variables**(*self*, *group=None*, *value_type=None*,
*inherited=None*, *public=None*, *imported=None*, *
detailed=None*)

Return a specified subset of this class’s `sorted_variables` list. If `value_type` is given, then only return variables whose values have the specified type. If `group` is given, then only return variables that belong to the specified group. If `inherited` is `True`, then only return inherited variables; if `inherited` is `False`, then only return local variables.

**Parameters**

- **value_type**: A string specifying the value type for which variables should be returned. Valid values are:
  - `'instancemethod'` - variables whose values are instance methods.
  - `'classmethod'` - variables whose values are class methods.
  - `'staticmethod'` - variables whose values are static methods.
  - `'properties'` - variables whose values are properties.
  - `'class'` - variables whose values are nested classes (including exceptions and types).
  - `'instancevariable'` - instance variables. This includes any variables that are explicitly marked as instance variables with docstring fields; and variables with docstrings that are initialized in the constructor.
  - `'classvariable'` - class variables. This includes any variables that are not included in any of the above categories.

  *(type=string)*

- **group**: The name of the group for which variables should be returned. A complete list of the groups defined by this `ClassDoc` is available in the `group_names` instance variable. The first element of this list is always the special group name `■`, which is used for variables that do not belong to any group.

  *(type=string)*

- **inherited**: If `None`, then return both inherited and local variables; if `True`, then return only inherited variables; if `False`, then return only local variables.

- **detailed**: If `True` (False), return only the variables deserving (not deserving) a detailed informative box. If `None`, don’t care.

  *(type=bool)*

**Requires**: The `sorted_variables` and `variable_groups` attributes must be initialized before this method can be used. See `init_sorted_variables()` and `init_variable_groups()`.

Inherited from `epydoc.apidoc.NamespaceDoc`*(Section 2.10, p. 34)*: `__init__`, `__init_grouping()`, `group_names()`, `init_sorted_variables()`, `init_variable_groups()`, `is_detailed()`, `report_unused_groups()`

Inherited from `epydoc.apidoc.ValueDoc`*(Section 2.8, p. 27)*: `__getstate__`, `__repr__`, `__setstate__`

Inherited from `epydoc.apidoc.APIDoc`*(Section 2.6, p. 19)*: `__cmp__`, `__hash__`, `__setattr__`, `__str__`, `__debug_setattr()`, `merge_and_overwrite()`, `pp()`, `specialize_to()`

**Value Representation**

45
2.12.2 Class Variables

Information about Metaclasses

**metaclass**

Value: `_Sentinel('UNKNOWN')`

Value Representation

Inherited from epydoc.apidoc.ValueDoc(Section 2.8, p. 27): pyval_repr(), summary_pyval_repr()

2.12.3 Instance Variables

Information about Base Classes

**bases**

API documentation for the class’s base classes.

**Type:** list of ClassDoc

**Value:** `_Sentinel('UNKNOWN')`

Information about Subclasses

**subclasses**

API documentation for the class’s known subclasses.

**Type:** list of ClassDoc

**Value:** `_Sentinel('UNKNOWN')`

Information about Variables

Inherited from epydoc.apidoc.NamespaceDoc(Section 2.10, p. 34): group_specs, sort_spec, sorted_variables, variable_groups, variables

Value Representation
2.13 Class RoutineDoc


API documentation information about a single routine.
2.13.1 Methods

**is_detailed(self)**

Does this object deserve a box with extra details?

**Return Value**

True if the object needs extra details, else False. *(type=bool)*

**Overrides:** epydoc.apidoc.APIDoc.is_detailed *(inherited documentation)*

**all_args(self)**

**Return Value**

A list of the names of all arguments (positional, vararg, and keyword), in order. If a positional argument consists of a tuple of names, then that tuple will be flattened.

Inherited from epydoc.apidoc.ValueDoc *(Section 2.8, p. 27): _getstate_(), _repr_(), _setstate_(), apidoc_links()*

Inherited from epydoc.apidoc.APIDoc *(Section 2.6, p. 19): _cmp_(), _hash_(), _init_(), _setattr_(), _str_(), __debug_setattr(), merge_and_overwrite(), pp(), specialize_to()*

**Value Representation**

Inherited from epydoc.apidoc.ValueDoc *(Section 2.8, p. 27): pyval_repr(), summary_pyval_repr()*

2.13.2 Class Variables

**Value Representation**

Inherited from epydoc.apidoc.ValueDoc *(Section 2.8, p. 27): REPR_LINELEN, REPR_MAXLINES, REPR_MIN_SCORE, SUMMARY_REPR_LINELEN*

2.13.3 Instance Variables

**callgraph_uid**

*DotGraph.uid of the call graph for the function.*

**Type:** str

**Value:** None

Inherited from epydoc.apidoc.ValueDoc *(Section 2.8, p. 27): canonical_name, toktree*

**Signature**

48
posargs
The names of the routine’s positional arguments. If an argument list contains "unpacking" arguments, then their names will be specified using nested lists. E.g., if a function’s argument list is ((x1,y1), (x2,y2)), then posargs will be [['x1', 'y1'], ['x2', 'y2']].
Type: list
Value: _Sentinel('UNKNOWN')

posarg_defaults
API documentation for the positional arguments’ default values. This list has the same length as posargs, and each element of posarg_defaults describes the corresponding argument in posargs. For positional arguments with no default, posargs Defaults will contain None.
Type: list of ValueDoc or None
Value: _Sentinel('UNKNOWN')

vararg
The name of the routine’s vararg argument, or None if it has no vararg argument.
Type: string or None
Value: _Sentinel('UNKNOWN')

kwarg
The name of the routine’s keyword argument, or None if it has no keyword argument.
Type: string or None
Value: _Sentinel('UNKNOWN')

lineno
The line number of the first line of the function’s signature. For Python functions, this is equal to func.func_code.co_firstlineno. The first line of a file is considered line 1.
Type: int
Value: _Sentinel('UNKNOWN')

Decorators
decorators
A list of names of decorators that were applied to this routine, in the order that they are listed in the source code. (I.e., in the reverse of the order that they were applied in.)
Type: list of string
Value: `_Sentinel('UNKNOWN')`

arg_descrs
A list of descriptions of the routine’s arguments. Each element of this list is a tuple (args, descr), where args is a list of argument names; and descr is a ParsedDocstring describing the argument(s) specified by arg.
Type: list
Value: `_Sentinel('UNKNOWN')`

arg_types
Descriptions of the expected types for the routine’s arguments, encoded as a dictionary mapping from argument names to type descriptions.
Type: dict from string to ParsedDocstring
Value: `_Sentinel('UNKNOWN')`

return_descr
A description of the value returned by this routine.
Type: ParsedDocstring
Value: `_Sentinel('UNKNOWN')`

return_type
A description of expected type for the value returned by this routine.
Type: ParsedDocstring
Value: `_Sentinel('UNKNOWN')`
exception_descrs
A list of descriptions of exceptions that the routine might raise. Each element of this list is a tuple \((\text{exc}, \text{descr})\), where \text{exc} is a string containing the exception name; and \text{descr} is a \texttt{ParsedDocstring} describing the circumstances under which the exception specified by \text{exc} is raised.

\textbf{Type:} list
\textbf{Value:} \_Sentinel(‘UNKNOWN’)

Inherited from epydoc.apidoc.APIDoc (Section 2.6, p. 19): descr, extra_docstring_fields, metadata, other_docs, summary

\textbf{Value Representation}
Inherited from epydoc.apidoc.ValueDoc (Section 2.8, p. 27): parse_repr, pyval

\textbf{Context}
Inherited from epydoc.apidoc.ValueDoc (Section 2.8, p. 27): defining_module

\textbf{Information about Imported Variables}
Inherited from epydoc.apidoc.ValueDoc (Section 2.8, p. 27): proxy_for

\textbf{Docstrings}
Inherited from epydoc.apidoc.APIDoc (Section 2.6, p. 19): docstring, docstring_lineno

\textbf{Source Information}
Inherited from epydoc.apidoc.APIDoc (Section 2.6, p. 19): docs_extracted_by
2.14 Class ClassMethodDoc

2.14.1 Methods

Inherited from epydoc.apidoc.RoutineDoc(Section 2.13, p. 47): all_args(), is_detailed()

Inherited from epydoc.apidoc.ValueDoc(Section 2.8, p. 27): __getstate__(), __repr__(), __setstate__(), apidoc_links()

Inherited from epydoc.apidoc.APIDoc(Section 2.6, p. 19): __cmp__(), __hash__(), __init__(),
__setattr__(), __str__(), __debug_setattr__(), merge_and_overwrite(), pp(), specialize_to()

Value Representation

Inherited from epydoc.apidoc.ValueDoc(Section 2.8, p. 27): pyval_repr(), summary_pyval_repr()

2.14.2 Class Variables

Value Representation

Inherited from epydoc.apidoc.ValueDoc(Section 2.8, p. 27): REPR_LINELEN, REPR_MAXLINES,
REPR_MIN_SCORE, SUMMARY_REPR_LINELEN

2.14.3 Instance Variables

Inherited from epydoc.apidoc.RoutineDoc(Section 2.13, p. 47): callgraph_uid

Inherited from epydoc.apidoc.ValueDoc(Section 2.8, p. 27): canonical_name, toktree

Signature
Inherited from epydoc.apidoc.RoutineDoc (Section 2.13, p. 47): kwarg, lineno, posarg_defaults, posargs, vararg

### Decorators

Inherited from epydoc.apidoc.RoutineDoc (Section 2.13, p. 47): decorators

### Information Extracted from Docstrings

Inherited from epydoc.apidoc.RoutineDoc (Section 2.13, p. 47): arg_descrs, arg_types, exception_descrs, return_descr, return_type

Inherited from epydoc.apidoc.APIDoc (Section 2.6, p. 19): descr, extra_docstring_fields, metadata, other_docs, summary

### Value Representation

Inherited from epydoc.apidoc.ValueDoc (Section 2.8, p. 27): parse_repr, pyval

### Context

Inherited from epydoc.apidoc.ValueDoc (Section 2.8, p. 27): defining_module

### Information about Imported Variables

Inherited from epydoc.apidoc.ValueDoc (Section 2.8, p. 27): proxy_for

### Docstrings

Inherited from epydoc.apidoc.APIDoc (Section 2.6, p. 19): docstring, docstring_lineno

### Source Information

Inherited from epydoc.apidoc.APIDoc (Section 2.6, p. 19): docs_extracted_by
2.15 Class StaticMethodDoc

![Diagram of StaticMethodDoc class hierarchy]

2.15.1 Methods

Inherited from epydoc.apidoc.RoutineDoc (Section 2.13, p. 47): all_args(), is_detailed()

Inherited from epydoc.apidoc.ValueDoc (Section 2.8, p. 27): __getstate__(), __repr__(), __setstate__(), apidoc_links()

Inherited from epydoc.apidoc.APIDoc (Section 2.6, p. 19): __cmp__(), __hash__(), __init__(), __setattr__(), __str__(), debug_setattr(), merge_and_overwrite(), pp(), specialize_to() (Section 2.13, p. 47)

### Value Representation

Inherited from epydoc.apidoc.ValueDoc (Section 2.8, p. 27): pyval_repr(), summary_pyval_repr()

2.15.2 Class Variables

### Value Representation

Inherited from epydoc.apidoc.ValueDoc (Section 2.8, p. 27): REPR_LINELEN, REPR_MAXLINES, REPR_MIN_SCORE, SUMMARY_REPR_LINELEN

2.15.3 Instance Variables

Inherited from epydoc.apidoc.RoutineDoc (Section 2.13, p. 47): callgraph_uid

Inherited from epydoc.apidoc.ValueDoc (Section 2.8, p. 27): canonical_name, toktree

### Signature

54
<table>
<thead>
<tr>
<th><strong>Class</strong> StaticMethodDoc</th>
<th><strong>Module</strong> epydoc.apidoc</th>
</tr>
</thead>
</table>

Inherited from epydoc.apidoc.RoutineDoc *(Section 2.13, p. 47)*: kwarg, lineno, posarg_defaults, posargs, vararg

### Decorators

Inherited from epydoc.apidoc.RoutineDoc *(Section 2.13, p. 47)*: decorators

### Information Extracted from Docstrings

Inherited from epydoc.apidoc.RoutineDoc *(Section 2.13, p. 47)*: arg_descrs, arg_types, exception_descrs, return_descr, return_type

Inherited from epydoc.apidoc.APIDoc *(Section 2.6, p. 19)*: descr, extra_docstring_fields, metadata, other_docs, summary

### Value Representation

Inherited from epydoc.apidoc.ValueDoc *(Section 2.8, p. 27)*: parse_repr, pyval

### Context

Inherited from epydoc.apidoc.ValueDoc *(Section 2.8, p. 27)*: defining_module

### Information about Imported Variables

Inherited from epydoc.apidoc.ValueDoc *(Section 2.8, p. 27)*: proxy_for

### Docstrings

Inherited from epydoc.apidoc.APIDoc *(Section 2.6, p. 19)*: docstring, docstring_lineno

### Source Information

Inherited from epydoc.apidoc.APIDoc *(Section 2.6, p. 19)*: docs_extracted_by
Class PropertyDoc

API documentation information about a single property.

2.16.1 Methods

**apidoc_links**(self, **filters**)  
Return a list of all APIDocs that are directly linked from this APIDoc (i.e., are contained or pointed to by one or more of this APIDoc’s attributes.)

Keyword argument filters can be used to selectively exclude certain categories of attribute value. For example, using includes=False will exclude variables that were imported from other modules; and subclasses=False will exclude subclasses. The filter categories currently supported by epydoc are:

- **imports**: Imported variables.
- **packages**: Containing packages for modules.
- **submodules**: Contained submodules for packages.
- **bases**: Bases for classes.
- **subclasses**: Subclasses for classes.
- **variables**: All variables.
- **private**: Private variables.
- **overrides**: Points from class variables to the variables they override. This filter is False by default.

Overrides: epydoc.apidoc.APIDoc.apidoc_links *(inherited documentation)*

**is_detailed**(self)  
Does this object deserve a box with extra details?

**Return Value**  
True if the object needs extra details, else False. *(type=bool)*

Overrides: epydoc.apidoc.APIDoc.is_detailed *(inherited documentation)*

Inherited from epydoc.apidoc.ValueDoc*(Section 2.8, p. 27): _getstate(), _repr(), _setstate()*

Inherited from epydoc.apidoc.APIDoc*(Section 2.6, p. 19): cmp(), hash(), init(), 
_setattr(), str(), debug_setattr(), merge_and_overwrite(), pp(), specialize_to()*
**Value Representation**

Inherited from epydoc.apidoc.ValueDoc (Section 2.8, p. 27): `pyval_repr()`, `summary_pyval_repr()`

### 2.16.2 Class Variables

**Value Representation**

Inherited from epydoc.apidoc.ValueDoc (Section 2.8, p. 27): `REPR_LINELEN`, `REPR_MAXLINES`, `REPR_MIN_SCORE`, `SUMMARY_REPR_LINELEN`

### 2.16.3 Instance Variables

Inherited from epydoc.apidoc.ValueDoc (Section 2.8, p. 27): `canonical_name`, `toktree`

**Property Access Functions**

**fget**

API documentation for the property’s get function.

**Type:** RoutineDoc

**Value:** `_Sentinel('UNKNOWN')`

**fset**

API documentation for the property’s set function.

**Type:** RoutineDoc

**Value:** `_Sentinel('UNKNOWN')`

**fdel**

API documentation for the property’s delete function.

**Type:** RoutineDoc

**Value:** `_Sentinel('UNKNOWN')`

**Information Extracted from Docstrings**

57
2.17 Class DocIndex

[x] out of date.

An index that .. hmm... it *can’t* be used to access some things, cuz they’re not at the root level. Do I want to add them or what? And if so, then I have a sort of a new top level. hmm.. so basically the question is what to do with a name that’s not in the root var’s name space. 2 types:

- entirely outside (eg os.path)
- inside but not known (eg a submodule that we didn’t look at?)
- container of current thing not examined?

An index of all the APIDoc objects that can be reached from a root set of ValueDocs.

The members of this index can be accessed by dotted name. In particular, DocIndex defines two mappings, accessed via the get_vardoc() and get_valdoc() methods, which can be used to access VariableDocs or ValueDocs respectively by name. (Two separate mappings are necessary because a
single name can be used to refer to both a variable and to the value contained by that variable.

Additionally, the index defines two sets of ValueDocs: "reachable ValueDocs" and "contained ValueDocs". The reachable ValueDocs are defined as the set of all ValueDocs that can be reached from the root set by following any sequence of pointers to ValueDocs or VariableDocs. The contained ValueDocs are defined as the set of all ValueDocs that can be reached from the root set by following only the ValueDoc pointers defined by non-imported VariableDocs. For example, if the root set contains a module \( m \), then the contained ValueDocs includes the ValueDocs for any functions, variables, or classes defined in that module, as well as methods and variables defined in classes defined in the module. The reachable ValueDocs includes all of those ValueDocs, as well as ValueDocs for any values imported into the module, and base classes for classes defined in the module.

### 2.17.1 Methods

```python
__init__(self, root)
```
Create a new documentation index, based on the given root set of ValueDocs. If any APIDocs reachable from the root set does not have a canonical name, then it will be assigned one. etc.

**Parameters**
- root: A list of ValueDocs.

```python
get_vardoc(self, name)
```
Return the VariableDoc with the given name, or None if this index does not contain a VariableDoc with the given name.

```python
get_valdoc(self, name)
```
Return the ValueDoc with the given name, or None if this index does not contain a ValueDoc with the given name.

```python
_get(self, name)
```
A helper function that's used to implement get_vardoc() and get_valdoc().

```python
_get_from(self, val, doc, identifier)
```
find(self, name, context)

Look for an APIDoc named name, relative to context. Return the APIDoc if one is found; otherwise, return None. find looks in the following places, in order:

- Function parameters (if one matches, return None)
- All enclosing namespaces, from closest to furthest.
- If name starts with 'self', then strip it off and look for the remaining part of the name using find
- Builtins
- Parameter attributes
- Classes at module level (if the name is not ambiguous)

Parameters

name: (type=str or DottedName)
context: (type=APIDoc)

_get_module_classes(self, docs)

Gather all the classes defined in a list of modules.

Very often people refers to classes only by class name, even if they are not imported in the namespace. Linking to such classes will fail if we look for them only in nested namespaces. Allow them to retrieve only by name.

Parameters

docs: containers of the objects to collect (type=list of APIDoc)

Return Value

mapping from objects name to the object(s) with that name (type=dict from str to ClassDoc or list)

reachable_valdocs(self, **filters)

Return a list of all ValueDocs that can be reached, directly or indirectly from this DocIndex's root set.

Parameters

filters: A set of filters that can be used to prevent reachable_valdocs from following specific link types when looking for ValueDocs that can be reached from the root set. See APIDoc.apidoc_links for a more complete description.

container(self, api_doc)

Return the ValueDoc that contains the given APIDoc, or None if its container is not in the index.

read_profiling_info(self, profile_stats)

Initialize the callers and callees variables, given a Stat object from the pstats module.

Warning: This method uses undocumented data structures inside of profile_stats.
_update_funcid_to_doc(self, profile_stats)
Update the dictionary mapping from pstat.Stat function ids to RoutineDocs. pstat.Stat function ids are tuples of (filename, lineno, funcname).

2.17.2 Instance Variables

root
The list of ValueDocs to document.
Type: list

mlclasses
A mapping from class names to ClassDoc. Contains classes defined at module level for modules in root and which can be used as fallback by find() if looking in containing namespaces fails.
Type: dict from str to ClassDoc or list

callers
A dictionary mapping from RoutineDocs in this index to lists of RoutineDocs for the routine’s callers. This dictionary is initialized by calling read_profiling_info().
Type: list of RoutineDoc

callees
A dictionary mapping from RoutineDocs in this index to lists of RoutineDocs for the routine’s callees. This dictionary is initialized by calling read_profiling_info().
Type: list of RoutineDoc

_funcid_to_doc
A mapping from profile function ids to corresponding APIDoc objects. A function id is a tuple of the form (filename, lineno, funcname). This is used to update the callers and callees variables.

_container_cache
A cache for the container() method, to increase speed.

_get_cache
A cache for the get_vardoc() and get_valdoc() methods, to increase speed.
3 Module epydoc.checker

Documentation completeness checker. This module defines a single class, DocChecker, which can be used to check the that specified classes of objects are documented.

3.1 Variables

_NO_DOCS
Value: ['__hash__', '__repr__', '__str__', '__cmp__']

_NO_BASIC
Value: ['__hash__', '__repr__', '__str__', '__cmp__']

_NO_RETURN
Value: ['__init__', '__hash__', '__repr__', '__str__', '__cmp__']

_NO_PARAM
Value: ['__cmp__']

3.2 Class DocChecker

Documentation completeness checker. DocChecker can be used to check that specified classes of objects are documented. To check the documentation for a group of objects, you should create a DocChecker from a DocIndex that documents those objects; and then use the check method to run specified checks on the objects’ documentation.

What checks are run, and what objects they are run on, are specified by the constants defined by DocChecker. These constants are divided into three groups.

- Type specifiers indicate what type of objects should be checked: MODULE; CLASS; FUNC; VAR; IVAR; CVAR; and RETURN.
- Public/private specifiers indicate whether public or private objects should be checked: PRIVATE.
- Check specifiers indicate what checks should be run on the objects: TYPE; DESCR; AUTHOR; and VERSION.

The check method is used to perform a check on the documentation. Its parameter is formed by or-ing together at least one value from each specifier group:

```python
>>> checker.check(DocChecker.MODULE | DocChecker.DESCR)
```

To specify multiple values from a single group, simply or their values together:

```python
>>> checker.check(DocChecker.MODULE | DocChecker.CLASS |
...                DocCheckerFUNC)
```
3.2.1 Methods

```python
__init__(self, docindex)
Create a new DocChecker that can be used to run checks on the documentation of the objects documented by docindex

Parameters
    docindex: A documentation map containing the documentation for the objects to
             be checked. (type=Docindex)
```

```python
check(self, *check_sets)
Run the specified checks on the documentation of the objects contained by this DocChecker's
DocIndex. Any errors found are printed to standard out.

Parameters
    check_sets: The checks that should be run on the documentation. This value is
                constructed by or-ing together the specifiers that indicate which objects
                should be checked, and which checks should be run. See the module
description for more information. If no checks are specified, then a
default set of checks will be run. (type=int)

Return Value
    True if no problems were found. (type=boolean)
```

```python
_check(self, checks)
```

```python
_name(self, doc)
```

```python
_check_basic(self, doc)
Check the description, author, version, and see-also fields of doc. This is used as a helper function
by _check_module, _check_class, and _check_func.

Parameters
    doc: The documentation that should be checked. (type=APIDoc)

Return Value
    None
```

```python
_check_module(self, doc)
Run checks on the module whose APIDoc is doc.

Parameters
    doc: The APIDoc of the module to check. (type=APIDoc)

Return Value
    None
```
_check_class(self, doc)

Run checks on the class whose APIDoc is doc.

Parameters
- doc: The APIDoc of the class to check. (type=APIDoc)

Return Value
- None

_check_property(self, doc)

_check_var(self, doc)

Run checks on the variable whose documentation is var and whose name is name.

Parameters
- doc: The documentation for the variable to check. (type=APIDoc)

Return Value
- None

_check_func(self, doc)

Run checks on the function whose APIDoc is doc.

Parameters
- doc: The APIDoc of the function to check. (type=APIDoc)

Return Value
- None

warning(self, msg, doc)

3.2.2 Class Variables

PROPERTY

Value: 256

ALL

Value: 24831

Types
<table>
<thead>
<tr>
<th><strong>MODULE</strong></th>
<th>Type specifier that indicates that the documentation of modules should be checked.</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Type:</strong> int</td>
<td><strong>Value:</strong> 1</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th><strong>CLASS</strong></th>
<th>Type specifier that indicates that the documentation of classes should be checked.</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Type:</strong> int</td>
<td><strong>Value:</strong> 2</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th><strong>FUNC</strong></th>
<th>Type specifier that indicates that the documentation of functions should be checked.</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Type:</strong> int</td>
<td><strong>Value:</strong> 4</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th><strong>VAR</strong></th>
<th>Type specifier that indicates that the documentation of module variables should be checked.</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Type:</strong> int</td>
<td><strong>Value:</strong> 8</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th><strong>PARAM</strong></th>
<th>Type specifier that indicates that the documentation of function and method parameters should be checked.</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Type:</strong> int</td>
<td><strong>Value:</strong> 64</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th><strong>RETURN</strong></th>
<th>Type specifier that indicates that the documentation of return values should be checked.</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Type:</strong> int</td>
<td><strong>Value:</strong> 128</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th><strong>ALL_T</strong></th>
<th>Type specifier that indicates that the documentation of all objects should be checked.</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Type:</strong> int</td>
<td><strong>Value:</strong> 511</td>
</tr>
<tr>
<td>Specifier</td>
<td>Description</td>
</tr>
<tr>
<td>-----------</td>
<td>-------------</td>
</tr>
<tr>
<td>CVAR</td>
<td>Type specifier that indicates that the documentation of class variables should be checked.</td>
</tr>
<tr>
<td>IVAR</td>
<td>Type specifier that indicates that the documentation of instance variables should be checked.</td>
</tr>
<tr>
<td>TYPE</td>
<td>Check specifier that indicates that every variable and parameter should have a @type field.</td>
</tr>
<tr>
<td>AUTHOR</td>
<td>Check specifier that indicates that every object should have an author field.</td>
</tr>
<tr>
<td>VERSION</td>
<td>Check specifier that indicates that every object should have a version field.</td>
</tr>
<tr>
<td>DESCR</td>
<td>Check specifier that indicates that every object should have a description.</td>
</tr>
<tr>
<td>ALL_C</td>
<td>Check specifier that indicates that all checks should be run.</td>
</tr>
</tbody>
</table>
PRIVATE
Specifier that indicates that private objects should be checked.

Type: int
Value: 16384
4 Module epydoc.cli

Command-line interface for epydoc. Abbreviated Usage:

```plaintext
epydoc [options] NAMES...
```

- **NAMES...** The Python modules to document.
- **--html** Generate HTML output (default).
- **--latex** Generate LaTeX output.
- **--pdf** Generate pdf output, via LaTeX.
- **--o DIR, --output DIR** The output directory.
- **--inheritance STYLE** The format for showing inherited objects.
- **-V, --version** Print the version of epydoc.
- **-h, --help** Display a usage message.

Run "epydoc –help" for a complete option list. See the epydoc(1) man page for more information.

**Config Files**

Configuration files can be specified with the *-config* option. These files are read using ConfigParser¹. Configuration files may set options or add names of modules to document. Option names are (usually) identical to the long names of command line options. To specify names to document, use any of the following option names:

```plaintext
module modules value values object objects
```

A simple example of a config file is:

```plaintext
[epydoc]
modules: sys, os, os.path, re, %(MYSANDBOXPATH)/utilities.py
name: Example
graph: classtree
introspect: no
```

All ConfigParser interpolations are done using local values and the environment variables.

**Verbosity Levels**

The *-v* and *-q* options increase and decrease verbosity, respectively. The default verbosity level is zero. The verbosity levels are currently defined as follows:

<table>
<thead>
<tr>
<th>Level</th>
<th>Progress</th>
<th>Markup warnings</th>
<th>Warnings</th>
<th>Errors</th>
</tr>
</thead>
<tbody>
<tr>
<td>-3</td>
<td>none</td>
<td>no</td>
<td>no</td>
<td>no</td>
</tr>
<tr>
<td>-2</td>
<td>none</td>
<td>no</td>
<td>no</td>
<td>yes</td>
</tr>
<tr>
<td>-1</td>
<td>none</td>
<td>no</td>
<td>yes</td>
<td>yes</td>
</tr>
<tr>
<td>0 (default)</td>
<td>bar</td>
<td>no</td>
<td>yes</td>
<td>yes</td>
</tr>
<tr>
<td>1</td>
<td>bar</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
</tr>
<tr>
<td>2</td>
<td>list</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
</tr>
</tbody>
</table>

¹http://docs.python.org/lib/module-ConfigParser.html
## 4.1 Functions

### Argument & Config File Parsing

- `option_defaults()`

- `add_action(option, opt, value, optparser)`

- `add_target(option, opt, value, optparser)`

- `parse_arguments()`

- `parse_configfiles(configfiles, options, names)`

- `_str_to_bool(val, optname)`

- `_str_to_int(val, optname)`

- `_str_to_list(val)`

### Interface

- `main(options)`

  Perform all actions indicated by the given set of options.

  **Return Value**

  - the `epydoc.apidoc.DocIndex` object created while running epydoc (or None).

- `write_html(docindex, options)`

- `write_pickle(docindex, options)`

  Helper for writing output to a pickle file, which can then be read in at a later time. But loading the pickle is only marginally faster than building the docs from scratch, so this has pretty limited application.

- `pickle.persistent_id(obj)`

  Helper for pickling, which allows us to save and restore UNKNOWN, which is required to be identical to apidoc.UNKNOWN.
pickle_persistent_load(\textit{identifier})

Helper for pickling, which allows us to save and restore \texttt{UNKNOWN}, which is required to be identical to \texttt{apidoc.UNKNOWN}.

\textbf{write\_latex}(\textit{docindex, options})

\textbf{write\_text}(\textit{docindex, options})

\textbf{check\_docs}(\textit{docindex, options})

\textbf{cli}()

Perform all actions indicated by the options in \texttt{sys.argv}.

\textbf{Return Value}

\texttt{the \texttt{epydoc.apidoc.DocIndex} object created while running epydoc (or None)}.

\textbf{\_profile}()

4.2 Variables

\textbf{INHERITANCE\_STYLES}

Value: (\texttt{\textquote{grouped}}, \texttt{\textquote{listed}}, \texttt{\textquote{included}}, \texttt{\textquote{hidden}})

\textbf{GRAPH\_TYPES}

Value: (\texttt{\textquote{classtree}}, \texttt{\textquote{callgraph}}, \texttt{\textquote{umlclasstree}})

\textbf{ACTIONS}

Value: (\texttt{\textquote{html}}, \texttt{\textquote{text}}, \texttt{\textquote{latex}}, \texttt{\textquote{dvi}}, \texttt{\textquote{ps}}, \texttt{\textquote{pdf}}, \texttt{\textquote{check}})

\textbf{DEFAULT\_DOCFORMAT}

Value: \texttt{\textquote{epytext}}

\textbf{PROFILER}

Which profiler to use: \texttt{\textquote{hotshot}} or \texttt{\textquote{profile}}

Value: \texttt{\textquote{profile}}
### Variables

#### epydoc.cli

<table>
<thead>
<tr>
<th><strong>TARGET_ACTIONS</strong></th>
<th>Value: <code>('html', 'latex', 'dvi', 'ps', 'pdf')</code></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>DEFAULT_ACTIONS</strong></td>
<td>Value: <code>('html')</code></td>
</tr>
<tr>
<td><strong>PDFDRIVERS</strong></td>
<td>Value: <code>('pdflatex', 'latex', 'auto')</code></td>
</tr>
<tr>
<td><strong>descr</strong></td>
<td>Value: <code>'Black on white, with blue highlights'</code></td>
</tr>
<tr>
<td><strong>key</strong></td>
<td>Value: <code>'white'</code></td>
</tr>
</tbody>
</table>
| **sheet**          | Value: `'

/* Epydoc CSS Stylesheet
 *
 * This stylesheet can...` |
| **topic**          | Value: `'inheritance'` |

#### Help Topics

##### DOCFORMATS

Value: `('epytext', 'plaintext', 'restructuredtext', 'javadoc')`

##### HELP_TOPICS

Value: `{ 'css': 'The following built-in CSS stylesheets are avail...`
## Interface

### `_RERUN_LATEX_RE`
Value: `re.compile(r'(?i)^LaTeX\s+Warning:\s+Label\(s\)\s+may\s+...')`

### 4.3 Class ConsoleLogger

```python
class ConsoleLogger:
    def __init__(self, verbosity, progress_mode=None):
        pass

    def start_block(self, header):
        pass

    def end_block(self):
        pass

    def _format(self, prefix, message, color):
        pass
```

**Known Subclasses:** epydoc.cli.UnifiedProgressConsoleLogger

### 4.3.1 Methods

- **`__init__`**

  ```python
  def __init__(self, verbosity, progress_mode=None):
      pass
  ```

- **`start_block`**

  ```python
  def start_block(self, header):
      pass
  ```

  - Start a new message block. Any calls to `info()`, `warning()`, or `error()` that occur between a call to `start_block` and a corresponding call to `end_block` will be grouped together, and displayed with a common header. `start_block` can be called multiple times (to form nested blocks), but every call to `start_block` must be balanced by a call to `end_block`.
  - **Overrides:** epydoc.log.Logger.start_block *(inherited documentation)*

- **`end_block`**

  ```python
  def end_block(self):
      pass
  ```

  - **Overrides:** epydoc.log.Logger.end_block *(inherited documentation)*

- **`_format`**

  ```python
  def _format(self, prefix, message, color):
      pass
  ```

  - Rewrap the message; but preserve newlines, and don’t touch any lines that begin with spaces.
**log**(self, level, message)

Display a message.

**Parameters**
- **message**: The message string to display. message may contain newlines, but does not need to end in a newline.
- **level**: An integer value indicating the severity of the message.

**Overrides**: epydoc.log.Logger.log *(inherited documentation)*

**_report**(self, message)

**progress**(self, percent, message='')

Update the progress display.

**Parameters**
- **percent**: A float from 0.0 to 1.0, indicating how much progress has been made.
- **message**: A message indicating the most recent action that contributed towards that progress.

**Overrides**: epydoc.log.Logger.progress *(inherited documentation)*

**_timestr**(self, dt)

**start_progress**(self, header=None)

Begin displaying progress for a new task. header is a description of the task for which progress is being reported. Each call to **start_progress** must be followed by a call to **end_progress** (with no intervening calls to **start_progress**).

**Overrides**: epydoc.log.Logger.start_progress *(inherited documentation)*

**end_progress**(self)

Finish off the display of progress for the current task. See **start_progress** for more information.

**Overrides**: epydoc.log.Logger.end_progress *(inherited documentation)*

**print_times**(self)

Inherited from epydoc.log.Logger *(Section 21.3, p. 194)*: close()

### 4.3.2 Instance Variables

**reported_message_levels**

This set contains all the message levels (WARNING, ERROR, etc) that have been reported. It is used by the options –fail-on-warning etc to determine the return value.
suppressed_docstring_warning
This variable will be incremented once every time a docstring warning is reported to the logger, but the verbosity level is too low for it to be displayed.

4.4 Class UnifiedProgressConsoleLogger

4.4.1 Methods

__init__(self, verbosity, stages, progress_mode=None)
Overrides: epydoc.cli.ConsoleLogger.__init__

progress(self, percent, message='')
Update the progress display.

Parameters
percent: A float from 0.0 to 1.0, indicating how much progress has been made.
message: A message indicating the most recent action that contributed towards that progress.

Overrides: epydoc.log.Logger.progress (inherited documentation)

start_progress(self, header=None)
Begin displaying progress for a new task. header is a description of the task for which progress is being reported. Each call to start_progress must be followed by a call to end_progress (with no intervening calls to start_progress).

Overrides: epydoc.log.Logger.start_progress (inherited documentation)

end_progress(self)
Finish off the display of progress for the current task. See start_progress for more information.

Overrides: epydoc.log.Logger.end_progress (inherited documentation)

print_times(self)

Overrides: epydoc.cli.ConsoleLogger.print_times
4.4.2 Instance Variables

Inherited from epydoc.cli.ConsoleLogger (Section 4.3, p. 72): reported_message_levels, suppressed_docstring_warning

4.5 Class HTMLLogger

A logger used to generate a log of all warnings and messages to an HTML file.

4.5.1 Methods

```python
__init__(self, directory, options)
```

```python
write_options(self, options)
```

```python
start_block(self, header)
```

Start a new message block. Any calls to `info()`, `warning()`, or `error()` that occur between a call to `start_block` and a corresponding call to `end_block` will be grouped together, and displayed with a common header. `start_block` can be called multiple times (to form nested blocks), but every call to `start_block` must be balanced by a call to `end_block`.

**Overrides:** epydoc.log.Logger.start_block (inherited documentation)

```python
end_block(self)
```

End a warning block. See `start_block` for details.

**Overrides:** epydoc.log.Logger.end_block (inherited documentation)
**log**(*self*, *level*, *message*)

Display a message.

**Parameters**

- **message**: The message string to display. *message* may contain newlines, but does not need to end in a newline.
- **level**: An integer value indicating the severity of the message.

**Overrides**: epydoc.log.Logger.log *(inherited documentation)*

**_message**(self, level, message)

**close**(self)

Perform any tasks needed to close this logger. This should be safe to call multiple times.

**Overrides**: epydoc.log.Logger.close *(inherited documentation)*

**_elapsed_time**(self)

Inherited from epydoc.log.Logger *(Section 21.3, p. 194): end_progress(), progress(), start_progress()*

### 4.5.2 Class Variables

**FILENAME**

Value: `epydoc-log.html`

**HEADER**

Value: `<?xml version="1.0" encoding="ascii"?>
<!DOCTYPE html P...`

**START_BLOCK**

Value: `'<div class="log-block"><h2 class="log-hdr">%s</h2>'`

**MESSAGE**

Value: `'<div class="log-%s"><b>%s</b>: 
%s</div>
'`

**END_BLOCK**

Value: `'</div>'`
<table>
<thead>
<tr>
<th>Class</th>
<th>HTMLLogger</th>
<th>Module</th>
<th>epydoc.cli</th>
</tr>
</thead>
<tbody>
<tr>
<td>FOOTER</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Value:</td>
<td>&quot;&lt;/body&gt;\n&lt;/html&gt;\n&quot;</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
5 Module epydoc.compat

Backwards compatibility with previous versions of Python.

This module provides backwards compatibility by defining several functions and classes that were not available in earlier versions of Python. Intended usage:

```python
>>> from epydoc.compat import *
```

Currently, epydoc requires Python 2.3+.

5.1 Functions

```python
sorted(iterable, cmp=None, key=None, reverse=False)
```

```python
reversed(iterable)
```
6 Module epydoc.docbuilder

Construct data structures that encode the API documentation for Python objects. These data structures are created using a series of steps:

1. **Building docs**: Extract basic information about the objects, and objects that are related to them. This can be done by introspecting the objects’ values (with `epydoc.docintrospecter`; or by parsing their source code (with `epydoc.docparser`).

2. **Merging**: Combine the information obtained from introspection & parsing each object into a single structure.

3. **Linking**: Replace any ‘pointers’ that were created for imported variables by their target (if it’s available).

4. **Naming**: Chose a unique ‘canonical name’ for each object.

5. **Docstring Parsing**: Parse the docstring of each object, and extract any pertinent information.

6. **Inheritance**: Add information about variables that classes inherit from their base classes.

The documentation information for each individual object is represented using an `APIDoc`; and the documentation for a collection of objects is represented using a `DocIndex`.

The main interface to `epydoc.docbuilder` consists of two functions:

- `build_doc()` – Builds documentation for a single item, and returns it as an `APIDoc` object.
- `build_doc_index()` – Builds documentation for a collection of items, and returns it as a `DocIndex` object.

The remaining functions are used by these two main functions to perform individual steps in the creation of the documentation.

6.1 Functions

```python
_import_docs_from_items(items, options)

_import_docs_from_package(pkg, options)

_do_import(filename, options, parent=None)

_report_errors(name, introspect_doc, parse_doc, introspect_error, parse_error)

_find_overrides(class_doc)
```

Set the `overrides` attribute for all variables in `class_doc`. This needs to be done early (before docstring parsing), so we can know which docstrings to suppress warnings for.

Documentation Construction
**build_doc**(*item*, *introspect=True*, *parse=True*,
*add_submodules=True*, *exclude_introspect=None*, *exclude_parse=None*,
*inherit_from_object=False*)

Build API documentation for a given item, and return it as an APIDoc object.

**Parameters**
- **item**: The item to document, specified using any of the following:
  - A string, naming a python package directory (e.g., 'epydoc/markup')
  - A string, naming a python file (e.g., 'epydoc/docparser.py')
  - A string, naming a python object (e.g., 'epydoc.docparser.DocParser')
  - Any (non-string) python object (e.g., list.append)
- **introspect**: If true, then use introspection to examine the specified items. Otherwise, just use parsing.
- **parse**: If true, then use parsing to examine the specified items. Otherwise, just use introspection.

**Return Value**
- APIDoc

**build_doc_index**(*items*, *introspect=True*, *parse=True*,
*add_submodules=True*, *exclude_introspect=None*, *exclude_parse=None*,
*inherit_from_object=False*)

Build API documentation for the given list of items, and return it in the form of a DocIndex.

**Parameters**
- **items**: The items to document, specified using any of the following:
  - A string, naming a python package directory (e.g., 'epydoc/markup')
  - A string, naming a python file (e.g., 'epydoc/docparser.py')
  - A string, naming a python object (e.g., 'epydoc.docparser.DocParser')
  - Any (non-string) python object (e.g., list.append)
- **introspect**: If true, then use introspection to examine the specified items. Otherwise, just use parsing.
- **parse**: If true, then use parsing to examine the specified items. Otherwise, just use introspection.

**Return Value**
- DocIndex

**report_valdoc_progress**(i, val_doc, val_docs)
<table>
<thead>
<tr>
<th>Function Name</th>
<th>Documentation</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>_get_docs_from_items</code></td>
<td>get docs from items(items, options)</td>
</tr>
<tr>
<td><code>_get_docs_from_pyobject</code></td>
<td>get docs from PyObject(obj, options, progress_estimator)</td>
</tr>
<tr>
<td><code>_get_docs_from_pynamename</code></td>
<td>get docs from pynamename(name, options, progress_estimator, suppress_warnings=False)</td>
</tr>
<tr>
<td><code>_get_docs_from_pyscript</code></td>
<td>get docs from pyscript(filename, options, progress_estimator)</td>
</tr>
<tr>
<td><code>_get_docs_from_module_file</code></td>
<td>get docs from module file(filename, options, progress_estimator, parent_docs=(None, None))</td>
</tr>
</tbody>
</table>

Construct and return the API documentation for the python module with the given filename.

### Parameters

- **parent_docs**: The ModuleDoc of the containing package. If `parent_docs` is not provided, then this method will check if the given filename is contained in a package; and if so, it will construct a stub ModuleDoc for the containing package(s). `parent_docs` is a tuple, where the first element is the parent from introspection, and the second element is the parent from parsing.

<table>
<thead>
<tr>
<th>Function Name</th>
<th>Documentation</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>_get_docs_from_submodules</code></td>
<td>get docs from submodules(item, pkg_docs, options, progress_estimator)</td>
</tr>
</tbody>
</table>

Merging
**register_attribute_mergefunc**(attrib, mergefunc)

Register an attribute merge function. This function will be called by `merge_docs()` when it needs to merge the attribute values of two APIDocs.

**Parameters**

attrib: The name of the attribute whose values are merged by mergefunc.
mergefunc: The merge function, whose signature is:

```python
>>> def mergefunc(introspect_val, parse_val, precedence, cyclecheck, path):
    ...
    return calculate_merged_value(introspect_val, parse_val)
```

Where `introspect_val` and `parse_val` are the two values to combine; `precedence` is a string indicating which value takes precedence for this attribute ('introspect' or 'parse'); `cyclecheck` is a value used by `merge_docs()` to make sure that it only visits each pair of docs once; and `path` is a string describing the path that was taken from the root to this attribute (used to generate log messages).

If the merge function needs to call `merge_docs()`, then it should pass `cyclecheck` and `path` back in. (When appropriate, a suffix should be added to `path` to describe the path taken to the merged values.)

**merge_docs**(introspect_doc, parse_doc, cyclecheck=None, path=None)

Merge the API documentation information that was obtained from introspection with information that was obtained from parsing. `introspect_doc` and `parse_doc` should be two APIDoc instances that describe the same object. `merge_docs()` combines the information from these two instances, and returns the merged APIDoc.

If `introspect_doc` and `parse_doc` are compatible, then they will be merged – i.e., they will be coerced to a common class, and their state will be stored in a shared dictionary. Once they have been merged, any change made to the attributes of one will affect the other. The value for each of the merged APIDoc's attributes is formed by combining the values of the source APIDocs' attributes, as follows:

- If either of the source attributes' value is `UNKNOWN`, then use the other source attribute's value.
- Otherwise, if an attribute merge function has been registered for the attribute, then use that function to calculate the merged value from the two source attribute values.
- Otherwise, if `MERGE_PRECEDENCE` is defined for the attribute, then use the attribute value from the source that it indicates.
- Otherwise, use the attribute value from the source indicated by `DEFAULT_MERGE_PRECEDENCE`.

If `introspect_doc` and `parse_doc` are not compatible (e.g., if their values have incompatible types), then `merge_docs()` will simply return either `introspect_doc` or `parse_doc`, depending on the value of `DEFAULT_MERGE_PRECEDENCE`. The two input APIDocs will not be merged or modified in any way.

**Parameters**

cyclecheck, path: These arguments should only be provided when `merge_docs()` is called by an attribute merge function. See `register_attribute_mergefunc()` for more details.
<table>
<thead>
<tr>
<th>Function</th>
<th>Parameters</th>
</tr>
</thead>
<tbody>
<tr>
<td>_merge_posargs_and_defaults</td>
<td>(introspect_doc, parse_doc, path)</td>
</tr>
<tr>
<td>merge_attribute</td>
<td>(attrib, introspect_doc, parse_doc, cyclecheck, path)</td>
</tr>
<tr>
<td>merge_variables</td>
<td>(varlist1, varlist2, precedence, cyclecheck, path)</td>
</tr>
<tr>
<td>merge_value</td>
<td>(value1, value2, precedence, cyclecheck, path)</td>
</tr>
<tr>
<td>merge_overrides</td>
<td>(v1, v2, precedence, cyclecheck, path)</td>
</tr>
<tr>
<td>merge_fget</td>
<td>(v1, v2, precedence, cyclecheck, path)</td>
</tr>
<tr>
<td>merge_fset</td>
<td>(v1, v2, precedence, cyclecheck, path)</td>
</tr>
<tr>
<td>merge_fdel</td>
<td>(v1, v2, precedence, cyclecheck, path)</td>
</tr>
<tr>
<td>merge_proxy_for</td>
<td>(v1, v2, precedence, cyclecheck, path)</td>
</tr>
<tr>
<td>merge_bases</td>
<td>(baselist1, baselist2, precedence, cyclecheck, path)</td>
</tr>
<tr>
<td>merge_posarg_defaults</td>
<td>(defaults1, defaults2, precedence, cyclecheck, path)</td>
</tr>
<tr>
<td>merge_docstring</td>
<td>(docstring1, docstring2, precedence, cyclecheck, path)</td>
</tr>
<tr>
<td>merge_docs_extracted_by</td>
<td>(v1, v2, precedence, cyclecheck, path)</td>
</tr>
<tr>
<td>merge_submodules</td>
<td>(v1, v2, precedence, cyclecheck, path)</td>
</tr>
</tbody>
</table>

**Linking**

<table>
<thead>
<tr>
<th>Function</th>
<th>Parameters</th>
</tr>
</thead>
<tbody>
<tr>
<td>link_imports</td>
<td>(val_doc, docindex)</td>
</tr>
</tbody>
</table>

**Naming**
assign_canonical_names(val_doc, name, docindex, score=0)

Assign a canonical name to val_doc (if it doesn’t have one already), and (recursively) to each variable in val_doc. In particular, val_doc will be assigned the canonical name name iff either:

- val_doc’s canonical name is UNKNOWN; or
- val_doc’s current canonical name was assigned by this method; but the score of the new name (score) is higher than the score of the current name (score_dict[val_doc]).

Note that canonical names will even be assigned to values like integers and None; but these should be harmless.

_var_shadows_self(var_doc, varname)

.fix_self_shadowing_var(var_doc, varname, docindex)

_unreachable_name_for(var_doc, docindex)

Inheritance

inherit_docs(class_doc, inherit_from_object)

_inherit_info(var_doc)

Copy any relevant documentation information from the variable that var_doc overrides into var_doc itself.

6.2 Variables

_INHERITED_ATTRIBS

Value: ['descr', 'summary', 'metadata', 'extra_docstring_fields'...]

Merging

MERGE_PRECEDENCE

Indicates whether information from introspection or parsing should be given precedence, for specific attributes. This dictionary maps from attribute names to either ‘introspect’ or ‘parse’.

Value: {'canonical_name': 'introspect', 'docformat': 'parse', 'd...
6.3 Class BuildOptions

Holds the parameters for a documentation building process.

6.3.1 Methods

```python
__init__(self, introspect=True, parse=True, exclude_introspect=None, exclude_parse=None, add_submodules=True)
```

```python
must_introspect(self, name)
```

Return True if a module is to be introspected with the current settings.

Parameters

- name: The name of the module to test (type=DottedName or str)
must_parse(self, name)

Return True if a module is to be parsed with the current settings.

Parameters:
name: The name of the module to test (type=DottedName or str)

_matches_filter(self, name, regexp)

Test if a module name matches a pattern.

Parameters:
name: The name of the module to test (type=DottedName or str)
regexp: The pattern object to match name against. If None, return False (type=pattern)

Return Value:
True if name in dotted format matches regexp, else False (type=bool)

6.4 Class _ProgressEstimator

Used to keep track of progress when generating the initial docs for the given items. (It is not known in advance how many items a package directory will contain, since it might depend on those packages’ _path_ values.)

6.4.1 Methods

_init_(self, items)

progress(self)

revise_estimate(self, pkg_item, modules, subpackages)

_est_pkg_modules(self, package_dir)
7 Module `epydoc.docintrospecter`

Extract API documentation about python objects by directly introspecting their values.

The function `introspect_docs()` function, which provides the main interface of this module, examines a Python objects via introspection, and uses the information it finds to create an API objects containing the API documentation for that objects.

The `register_introspecter()` method can be used to extend the functionality of `docintrospector`, by providing methods that handle special value types.

7.1 Functions

```python
def clear_cache()
    Discard any cached API objects values that have been computed for introspected values.
```

```python
def introspect_docs(value=None, name=None, filename=None, context=None, is_script=False, module_name=None)
    Generate the API documentation for a specified object by introspecting Python values, and return it as a `ValueDoc`. The object to generate documentation for may be specified using the `value` parameter, the `filename` parameter, or the `name` parameter. (It is an error to specify more than one of these three parameters, or to not specify any of them.)

**Parameters**

- **value**: The python object that should be documented.
- **filename**: The name of the file that contains the python source code for a package, module, or script. If `filename` is specified, then `introspect` will return a `ModuleDoc` describing its contents.
- **name**: The fully-qualified python dotted name of any value (including packages, modules, classes, and functions). `DocParser` will automatically figure out which module(s) it needs to import in order to find the documentation for the specified object.
- **context**: The API documentation for the class of module that contains `value` (if available).
- **module_name**: The name of the module where the value is defined. Useful to retrieve the docstring encoding if there is no way to detect the module by introspection (such as in properties)

```python
def get_valuedoc(value)
    If a `ValueDoc` for the given value exists in the valuedoc cache, then return it; otherwise, create a new `ValueDoc`, add it to the cache, and return it. When possible, the new `ValueDoc`'s `pyval`, `repr`, and `canonical_name` attributes will be set appropriately.
```
Classes

- `introspect_module(module, module_doc, module_name=None, preliminary=False)`
  Add API documentation information about the module `module` to `module_doc`.

- `introspect_class(cls, class_doc, module_name=None)`
  Add API documentation information about the class `cls` to `class_doc`.

- `introspect_routine(routine, routine_doc, module_name=None)`
  Add API documentation information about the function `routine` to `routine_doc` (specializing it to `Routine_doc`).

- `introspect_property(prop, prop_doc, module_name=None)`
  Add API documentation information about the property `prop` to `prop_doc` (specializing it to `PropertyDoc`).

- `introspect_other(val, val_doc, module_name=None)`
  Specialize `val_doc` to a `GenericValueDoc` and return it.

- `isclass(object)`
  Return true if the given object is a class. In particular, return true if object is an instance of `types.TypeType` or of `types.ClassType`. This is used instead of `inspect.isclass()`, because the latter returns true for objects that are not classes (in particular, it returns true for any object that has a `__bases__` attribute, including objects that define `__getattr__` to always return a value).

- `register_class_type(typ)`
  Add a type to the lists of types that should be treated as classes. By default, this list contains `TypeType` and `ClassType`.

- `is_future_feature(object)`
  Return True if `object` results from a `from __future__ import feature` statement.

- `get_docstring(value, module_name=None)`
  Return the docstring for the given value; or `None` if it does not have a docstring.

Return Value: Unicode
**get_canonical_name**(value, strict=False)

Return Value
the canonical name for value, or UNKNOWN if no canonical name can be found. Currently, `get_canonical_name` can find canonical names for: modules; functions; non-nested classes; methods of non-nested classes; and some class methods of non-nested classes. *(type=DottedName or UNKNOWN)*

**verify_name**(value, dotted_name)
Verify the name. E.g., if it’s a nested class, then we won’t be able to find it with the name we constructed.

**value_repr**(value)

**get_containing_module**(value)
Return the name of the module containing the given value, or None if the module name can’t be determined.

Return Value
DottedName

**_find_function_module**(func)

Parameters
- func: The function whose module should be found. *(type=function)*

Return Value
The module that defines the given function. *(type=module)*

**register_introspector**(applicability_test, introspector, priority=10)
Register an introspector function. Introspector functions take two arguments, a python value and a ValueDoc object, and should add information about the given value to the the ValueDoc. Usually, the first line of an inspector function will specialize it to a subclass of ValueDoc, using ValueDoc.specialize_to():

```python
>>> def typical_introspector(value, value_doc):
...    value_doc.specialize_to(SomeSubclassOfValueDoc)
...    <add info to value_doc>
```

Parameters
- priority: The priority of this introspector, which determines the order in which introspecters are tried – introspecters with lower numbers are tried first. The standard introspecters have priorities ranging from 20 to 30. The default priority (10) will place new introspecters before standard introspecters.

**_get_introspector**(value)

**is_classmethod**(v)
is_staticmethod(v)

is_property(v)

is_getset(v)

is_member(v)

get_value_from_filename(filename, context=None)

get_value_from_scriptname(filename)

get_value_from_name(name, globs=None)

Parameters
globs: A namespace to check for the value, if there is no module containing the named value. Defaults to __builtin__.

lookup(module, name)

import(name, filename=None)

Run the given callable in a 'sandboxed' environment. Currently, this includes saving and restoring the contents of sys and __builtins__; and suppressing stdin, stdout, and stderr.

introspect_docstring_lineno(api_doc)

Try to determine the line number on which the given item's docstring begins. Return the line number, or None if the line number can't be determined. The line number of the first line in the file is 1.

is_zope_type(val)

is_zope_method(val)
7.2 Variables

_{valuedoc_cache}_
A cache containing the API documentation for values that we've already seen. This cache is implemented as a dictionary that maps a value’s pyid to its ValueDoc.

Note that if we encounter a value but decide not to introspect it (because it’s imported from another module), then _valuedoc_cache_ will contain an entry for the value, but the value will not be listed in _introspected_values_.

Value: `{135509120: <ClassDoc bool>, 135509312: <GenericValueDoc ...

_{introspected_values}_
A record which values we've introspected, encoded as a dictionary from pyid to bool.

Value: `{135509120: True, 135509312: True, 135509324: True, 13550...

UNDOCUMENTED_MODULE_VARS
A list of module variables that should not be included in a module's API documentation.

Value: `('__builtins__', '__doc__', '__all__', '__file__', '__path...

UNDOCUMENTED_CLASS_VARS
A list of class variables that should not be included in a class’s API documentation.

Value: `('__doc__', '__module__', '__dict__', '__weakref__', '__s...

_CLASS_TYPES
A list of types that should be treated as classes.

Value: `set([<type 'classobj'>, <type 'type'>])

_future_check_works
Value: `True`

_introspector_registry
Value: `[]`

_dev_null
Value: `_DevNull()`
7.3 Class _DevNull

A "file-like" object that discards anything that is written and always reports end-of-file when read. _DevNull is used by _import() to discard output when importing modules; and to ensure that stdin appears closed.

7.3.1 Methods

```python
__init__(self)

close(self)

flush(self)

read(self, size=0)

readline(self, size=0)

readlines(self, sizehint=0)

seek(self, offset, whence=0)

tell(self)

truncate(self, size=0)

write(self, str)

writelines(self, sequence)

xreadlines(self, sizehint=0)
```
8 Module  epydoc.docparser

Extract API documentation about python objects by parsing their source code.

The function `parse_docs()` provides the main interface of this module, reads and parses the Python source code for a module, and uses it to create an `APIDoc` object containing the API documentation for the variables and values defined in that module.

Currently, `parse_docs()` extracts documentation from the following source code constructions:

- module docstring
- import statements
- class definition blocks
- function definition blocks
- assignment statements
  - simple assignment statements
  - assignment statements with multiple '='s
  - assignment statements with unpacked left-hand sides
  - assignment statements that wrap a function in classmethod or staticmethod.
  - assignment to special variables `__path__`, `__all__`, and `__docformat__`
- delete statements

`parse_docs()` does not yet support the following source code constructions:

- assignment statements that create properties

By default, `parse_docs()` will explore the contents of top-level `try` and `if` blocks. If desired, `parse_docs()` can also be configured to explore the contents of `while` and `for` blocks. (See the configuration constants, below.)

**To Do:** Make it possible to extend the functionality of `parse_docs()`, by replacing `process_line` with a dispatch table that can be customized (similarly to `docintrospector.register_introspector()`).

8.1 Functions

Module parser
parse_docs(filename=None, name=None, context=None, is_script=False)

Generate the API documentation for a specified object by parsing Python source files, and return it as a ValueDoc. The object to generate documentation for may be specified using the filename parameter or the name parameter. (It is an error to specify both a filename and a name; or to specify neither a filename nor a name).

Parameters
- filename: The name of the file that contains the python source code for a package, module, or script. If filename is specified, then parse will return a ModuleDoc describing its contents.
- name: The fully-qualified python dotted name of any value (including packages, modules, classes, and functions). parse_docs() will automatically figure out which module(s) it needs to parse in order to find the documentation for the specified object.
- context: The API documentation for the package that contains filename. If no context is given, then filename is assumed to contain a top-level module or package. It is an error to specify a context if the name argument is used.

Return Value
- ValueDoc

_parse_package(package_dir)

If the given directory is a package directory, then parse its __init__.py file (and the __init__.py files of all ancestor packages); and return its ModuleDoc.

handle_special_module_vars(module_doc)

_module_var_toktree(module_doc, name)

Module Lookup

_find(name, package_doc=None)

Return the API documentation for the object whose name is name. package_doc, if specified, is the API documentation for the package containing the named object.

_is_submodule_import_var(module_doc, var_name)

Return true if var_name is the name of a variable in module_doc that just contains an imported_from link to a submodule of the same name. (I.e., is a variable created when a package imports one of its own submodules.)

_find_in_namespace(name, namespace_doc)
_get_filename(identifier, path=None)

File tokenization loop

process_file(module_doc)

Read the given ModuleDoc's file, and add variables corresponding to any objects defined in that file. In particular, read and tokenize module_doc.filename, and process each logical line using process_line().

add_to_group(container, api_doc, group_name)

script_guard(line)

Detect the idiomatic trick if __name__ == "__main__":

Shallow parser

shallow_parse(line_toks)

Given a flat list of tokens, return a nested tree structure (called a token tree), whose leaves are identical to the original list, but whose structure reflects the structure implied by the grouping tokens (i.e., parentheses, braces, and brackets). If the parentheses, braces, and brackets do not match, or are not balanced, then raise a ParseError.

Assign some structure to a sequence of structure (group parens).

Line processing

process_line(line, parent_docs, prev_line_doc, lineno, comments, decorators, encoding)

Return Value
new-doc, decorator...

process_control_flow_line(line, parent_docs, prev_line_doc, lineno, comments, decorators, encoding)

process_import(line, parent_docs, prev_line_doc, lineno, comments, decorators, encoding)

process_from_import(line, parent_docs, prev_line_doc, lineno, comments, decorators, encoding)
Handle a statement of the form:

```plaintext
>>> from <src> import *
```

If IMPORT_HANDLING is 'parse', then first try to parse the module `<src>`, and copy all of its exported variables to `parent_docs[-1]`.

Otherwise, try to determine the names of the variables exported by `<src>`, and create a new variable for each export. If IMPORT_STAR_HANDLING is 'parse', then the list of exports if found by parsing `<src>`; if it is 'introspect', then the list of exports is found by importing and introspecting `<src>`.

Handle a statement of the form:

```plaintext
>>> import <name>
```

If IMPORT_HANDLING is 'parse', then first try to find the value by parsing; and create an appropriate variable in parentdoc.

Otherwise, add a variable for the imported variable. (More than one variable may be created for cases like `import a.b`, where we need to create a variable 'a' in parentdoc containing a proxy module; and a variable 'b' in the proxy module.

Handle a statement of the form:

```plaintext
>>> import src as name
```

If IMPORT_HANDLING is 'parse', then first try to find the value by parsing; and create an appropriate variable in parentdoc.

Otherwise, create a variables with its imported_from attribute pointing to the imported object.

Add a new imported variable named `name` to `container`, with imported_from=src.

If the given name is package-local (relative to the current context, as determined by `parent_docs`), then convert it to a global name.

Add a new imported variable named `name` to `container`, with imported_from=src.

If the given name is package-local (relative to the current context, as determined by `parent_docs`), then convert it to a global name.

Handle a statement of the form:

```plaintext
>>> import src as name
```

If IMPORT_HANDLING is 'parse', then first try to find the value by parsing; and create an appropriate variable in parentdoc.

Otherwise, create a variables with its imported_from attribute pointing to the imported object.

Add a new imported variable named `name` to `container`, with imported_from=src.

If the given name is package-local (relative to the current context, as determined by `parent_docs`), then convert it to a global name.

Handle a statement of the form:

```plaintext
>>> from <src> import *
```

If IMPORT_HANDLING is 'parse', then first try to parse the module `<src>`, and copy all of its exported variables to `parent_docs[-1]`.

Otherwise, try to determine the names of the variables exported by `<src>`, and create a new variable for each export. If IMPORT_STAR_HANDLING is 'parse', then the list of exports if found by parsing `<src>`; if it is 'introspect', then the list of exports is found by importing and introspecting `<src>`.
get_lhs_parent(lhs_name, parent_docs)

process_one_line_block(line, parent_docs, prev_line_doc, lineno, comments, decorators, encoding)

- The line handler for single-line blocks, such as:
  >>> def f(x): return x*2

  This handler calls process_line twice: once for the tokens up to and including the colon, and once for the remaining tokens. The comment docstring is applied to the first line only.

  Return Value
  None

process_multi_stmt(line, parent_docs, prev_line_doc, lineno, comments, decorators, encoding)

- The line handler for semicolon-separated statements, such as:
  >>> x=1; y=2; z=3

  This handler calls process_line once for each statement. The comment docstring is not passed on to any of the sub-statements.

  Return Value
  None

process_del(line, parent_docs, prev_line_doc, lineno, comments, decorators, encoding)

- The line handler for delete statements, such as:
  >>> del x, y.z

  This handler calls del_variable for each dotted variable in the variable list. The variable list may be nested. Complex expressions in the variable list (such as x[3]) are ignored.

  Return Value
  None

process_docstring(line, parent_docs, prev_line_doc, lineno, comments, decorators, encoding)

- The line handler for bare string literals. If prev_line_doc is not None, then the string literal is added to that APIDoc as a docstring. If it already has a docstring (from comment docstrings), then the new docstring will be appended to the old one.
**process_funcdef**(*line*, *parent_docs*, *prev_line_doc*, *lineno*, *comments*, *decorators*, *encoding*)

The line handler for function declaration lines, such as:

```python
>>> def f(a, b=22, (c,d)):
```

This handler creates and initializes a new `VariableDoc` containing a `RoutineDoc`, adds the `VariableDoc` to the containing namespace, and returns the `RoutineDoc`.

**apply_decorator**(*decorator_name*, *func_doc*, *parent_docs*, *lineno*)

**init_arglist**(*func_doc*, *arglist*)

**process_classdef**(*line*, *parent_docs*, *prev_line_doc*, *lineno*, *comments*, *decorators*, *encoding*)

The line handler for class declaration lines, such as:

```python
>>> class Foo(Bar, Baz):
```

This handler creates and initializes a new `VariableDoc` containing a `ClassDoc`, adds the `VariableDoc` to the containing namespace, and returns the `ClassDoc`.

**_proxy_base**(**attrs**)  

**find_base**(*name*, *parent_docs*)

**process_append_to_all**(*line*, *parent_docs*, *prev_line_doc*, *lineno*, *comments*, *decorators*, *encoding*)

The line handler for `__all__.append()` lines; either of:

```python
>>> __all__.append('name')
>>> __all__ += ['name']
```

This handler looks up the value of the variable `__all__` in `parent_docs`; and if it is found, and has a list-of-strings value, the handler appends the new name.

**append_to_all**(*name*, *parent_docs*, *lineno*)

**is_append_to_all**(*line*)

Check if a line is an `__all__.append` line()

See Also: `process_append_to_all`

**Parsing**
**dotted_names_in(elt_list)**

Return a list of all simple dotted names in the given expression.

**parse_name(elt, strip_parens=False)**

If the given token tree element is a name token, then return that name as a string. Otherwise, raise ParseError.

Parameters

- **strip_parens**: If true, then if elt is a single name enclosed in parentheses, then return that name.

**parse_dotted_name(elt_list, strip_parens=True, parent_name=None)**

Parameters

- **parent_name**: canonical name of referring module, to resolve relative imports.
  
  *(type=DottedName)*

Bug: does not handle 'x.(y).z'

**split_on(elt_list, split_tok)**

**parse_funcdef_arg(elt)**

If the given tree token element contains a valid function definition argument (i.e., an identifier token or nested list of identifiers), then return a corresponding string identifier or nested list of string identifiers. Otherwise, raise a ParseError.

**parse_classdef_bases(elt)**

If the given tree token element contains a valid base list (that contains only dotted names), then return a corresponding list of DottedName objects. Otherwise, raise a ParseError.

Bug: Does not handle either of:

- class A((base.in.parens)): pass
- class B((lambda:calculated.base)()): pass

**parse_dotted_name_list(elt_list)**

If the given list of tree token elements contains a comma-separated list of dotted names, then return a corresponding list of DottedName objects. Otherwise, raise ParseError.

**parse_string(elt_list)**

**parse_string_list(elt_list, require_sequence=False)**

---

**Variable Manipulation**
**set_variable** *(namespace, var_doc, preserve_docstring=False)*

Add var_doc to namespace. If namespace already contains a variable with the same name, then discard the old variable. If **preserve_docstring** is true, then keep the old variable’s docstring when overwriting a variable.

**del_variable** *(namespace, name)*

**Name Lookup**

**lookup_name** *(identifier, parent/docs)*

Find and return the documentation for the variable named by the given identifier.

**Return Value**

VariableDoc or None

**lookup_variable** *(dotted_name, parent/docs)*

**lookup_value** *(dotted_name, parent/docs)*

Find and return the documentation for the value contained in the variable with the given name in the current namespace.

**Docstring Comments**

**add_docstring_from_comments** *(api_doc, comments)*

**Tree tokens**

**_join_toktree** *(s1, s2)*

**_pp_toktree_add_piece** *(spacing, pieces, piece)*

**pp_toktree** *(elts, spacing='normal', indent=0)*

**_pp_toktree** *(elts, spacing, indent, pieces)*

**Helper Functions**

**get_module_encoding** *(filename)*

See Also: PEP 263

*http://www.python.org/peps/pep-0263.html*
Variables

_module_name(filename, package_doc)
Return (dotted_name, is_package)

flatten(lst, out=None)
Parameters
lst: The nested list that should be flattened.
Return Value
a flat list containing the leaves of the given nested list.

8.2 Variables

_moduledoc_cache
A cache of ModuleDocs that we’ve already created. _moduledoc_cache is a dictionary mapping from filenames to ValueDoc objects.
Type: dict
Value: {'/home/edloper/newdata/projects/docutils/docutils/__init...'

Configuration Constants: Control Flow

PARSE_TRY_BLOCKS
Should the contents of try blocks be examined?
Value: True

PARSE_EXCEPT_BLOCKS
Should the contents of except blocks be examined?
Value: True

PARSE_FINALLY_BLOCKS
Should the contents of finally blocks be examined?
Value: True

PARSE_IF_BLOCKS
Should the contents of if blocks be examined?
Value: True
**PARSE_ELSE_BLOCKS**
Should the contents of *else* and *elif* blocks be examined?
Value: **True**

**PARSE_WHILE_BLOCKS**
Should the contents of *while* blocks be examined?
Value: **False**

**PARSE_FOR_BLOCKS**
Should the contents of *for* blocks be examined?
Value: **False**

---

**Configuration Constants: Imports**

**IMPORT_HANDLING**
What should *docparser* do when it encounters an import statement?
- *'link'*: Create variabledoc objects with imported *from* pointers to the source object.
- *'parse'*: Parse the imported file, to find the actual documentation for the imported object.
  (This will fall back to the 'link' behavior if the imported file can't be parsed, e.g., if it's a builtin.)
Value: **'link'**

**IMPORT_STAR_HANDLING**
When *docparser* encounters a *'from m import *'* statement, and is unable to parse *m* (either because *IMPORT_HANDLING='link'* or because parsing failed), how should it determine the list of identifiers exposed by *m*?
- *'ignore'*: ignore the import statement, and don’t create any new variables.
- *'parse'*: parse it to find a list of the identifiers that it exports. (This will fall back to the 'ignore' behavior if the imported file can't be parsed, e.g., if it's a builtin.)
- *'introspect'*: import the module and introspect it (using *dir*) to find a list of the identifiers that it exports. (This will fall back to the 'ignore' behavior if the imported file can't be parsed, e.g., if it's a builtin.)
Value: **'parse'**
**DEFAULT_DECORATOR_BEHAVIOR**

When DocParse encounters an unknown decorator, what should it do to the documentation of the decorated function?

- **'transparent'**: leave the function’s documentation as-is.
- **'opaque'**: replace the function’s documentation with an empty ValueDoc object, reflecting the fact that we have no knowledge about what value the decorator returns.

Value: **'transparent'**

**PUBLIC_DECORATOR APPENDS TO ALL**

If true, then the @public decorator will append the function’s name to the module’s __all__ variable.

Value: **True**

**BASE_HANDLING**

What should docparser do when it encounters a base class that was imported from another module?

- **'link'**: Create a valuedoc with a proxy_for pointer to the base class.
- **'parse'**: Parse the file containing the base class, to find the actual documentation for it. (This will fall back to the 'link' behavior if the imported file can’t be parsed, e.g., if it’s a builtin.)

Value: **'parse'**

**Configuration Constants: Comment docstrings**

**COMMENT_DOCSTRING_MARKER**

The prefix used to mark comments that contain attribute docstrings for variables.

Value: **'#: '**

**Configuration Constants: Grouping**

**START_GROUP_MARKER**

The prefix used to mark a comment that starts a group. This marker should be followed (on the same line) by the name of the group. Following a start-group comment, all variables defined at the same indentation level will be assigned to this group name, until the parser reaches the end of the file, a matching end-group comment, or another start-group comment at the same indentation level.

Value: **'#{ '**
8.3 Class ParseError

An exception that is used to signify that `docparser` encountered syntactically invalid Python code while processing a Python source file.

8.3.1 Methods

Inherited from exceptions.Exception: `_init_`, `_new_`


8.3.2 Properties

Inherited from exceptions.BaseException: `args`, `message`
9 Module epydoc.docstringparser

Parse docstrings and handle any fields it defines, such as @type and @author. Fields are used to describe specific information about an object. There are two classes of fields: simple fields and special fields.

Simple fields are fields that get stored directly in an APIDoc’s metadata dictionary, without any special processing. The set of simple fields is defined by the list STANDARD_FIELDS, whose elements are DocstringFields.

Special fields are fields that perform some sort of processing on the APIDoc, or add information to attributes other than the metadata dictionary. Special fields are are handled by field handler functions, which are registered using register_field_handler.

9.1 Functions

### Docstring Parsing

**parse_docstring**(*api_doc, docindex, suppress_warnings=[])*)

Process the given APIDoc’s docstring. In particular, populate the APIDoc’s descr and summary attributes, and add any information provided by fields in the docstring.

**Parameters**

- **docindex:** A DocIndex, used to find the containing module (to look up the docformat); and to find any user docfields defined by containing objects.
- **suppress_warnings:** A set of objects for which docstring warnings should be suppressed.

**add_metadata_from_var**(*api_doc, field*)

**initialize_api_doc**(*api_doc*)

A helper function for parse_docstring() that initializes the attributes that parse_docstring() will write to.

**split_init_fields**(*fields, warnings*)

Remove the fields related to the constructor from a class docstring fields list.

**Parameters**

- **fields:** The fields to process. The list will be modified in place (type=list of markup.Field)
- **warnings:** A list to emit processing warnings (type=list)

**Return Value**

The fields items to be applied to the \_\_init\_\_ method(type=list of markup.Field)
Functions

Module epydoc.docstringparser

report_errors(api_doc, docindex, parse_errors, field_warnings)

A helper function for parse_docstring() that reports any markup warnings and field warnings that we encountered while processing api_doc’s docstring.

Field Processing

process_field(api_doc, docindex, tag, arg, descr)

Process a single field, and use it to update api_doc. If tag is the name of a special field, then call its handler function. If tag is the name of a simple field, then use process_simple_field to process it. Otherwise, check if it’s a user-defined field, defined in this docstring or the docstring of a containing object; and if so, process it with process_simple_field.

Parameters

tag: The field’s tag, such as 'author'

arg: The field’s optional argument

descr: The description following the field tag and argument.

Raises

ValueError If a problem was encountered while processing the field. The ValueError’s string argument is an explanation of the problem, which should be displayed as a warning message.

user_docfields(api_doc, docindex)

Return a list of user defined fields that can be used for the given object. This list is taken from the given api_doc, and any of its containing NamespaceDocs.

Note: We assume here that a parent’s docstring will always be parsed before its childrens’. This is indeed the case when we are called via docbuilder.build_doc_index(). If a child’s docstring is parsed before its parents, then its parent won’t yet have had its extra_docstring_fields attribute initialized.

register_field_handler(handler, *field_tags)

Register the given field handler function for processing any of the given field tags. Field handler functions should have the following signature:

```python
>>> def field_handler(api_doc, docindex, tag, arg, descr):
...   # update api_doc in response to the field.
```

Where api_doc is the documentation object to update; docindex is a DocIndex that can be used to look up the documentation for related objects; tag is the field tag that was used; arg is the optional argument; and descr is the description following the field tag and argument.

Field Handler Functions

process_summary_field(api_doc, docindex, tag, arg, descr)

Store descr in api_doc.summary
**process_include_field**(api_doc, docindex, tag, arg, descr)

Copy the docstring contents from the object named in descr.

**process_undocumented_field**(api_doc, docindex, tag, arg, descr)

Remove any documentation for the variables named in descr.

**process_group_field**(api_doc, docindex, tag, arg, descr)

Define a group named arg containing the variables whose names are listed in descr.

**process_deffield_field**(api_doc, docindex, tag, arg, descr)

Define a new custom field.

**process_raise_field**(api_doc, docindex, tag, arg, descr)

Record the fact that api_doc can raise the exception named tag in api_doc.exception_descrs.

**process_sort_field**(api_doc, docindex, tag, arg, descr)

**process_type_field**(api_doc, docindex, tag, arg, descr)

**process_var_field**(api_doc, docindex, tag, arg, descr)

**process_cvar_field**(api_doc, docindex, tag, arg, descr)

**process_ivar_field**(api_doc, docindex, tag, arg, descr)

**process_return_field**(api_doc, docindex, tag, arg, descr)

**process_rtype_field**(api_doc, docindex, tag, arg, descr)

**process_arg_field**(api_doc, docindex, tag, arg, descr)

**process_kwarg_field**(api_doc, docindex, tag, arg, descr)

### Helper Functions

**check_type_fields**(api_doc, field_warnings)

Check to make sure that all type fields correspond to some documented parameter; if not, append a warning to field_warnings.
set_var_desc\(r\(api\_doc, ident, descr\)\)

set_var_type\(\(api\_doc, ident, descr\)\)

_check\(\(api\_doc, tag, arg, context=None, expect_arg=None\)\)

get_docformat\(\(api\_doc, docindex\)\)

Return the name of the markup language that should be used to parse the API documentation for the given object.

unindent_docstring\(\(docstring\)\)

_descr_to_identifiers\(\(descr\)\)

Given a ParsedDocstring that contains a list of identifiers, return a list of those identifiers. This is used by fields such as @group and @sort, which expect lists of identifiers as their values. To extract the identifiers, the docstring is first converted to plaintext, and then split. The plaintext content of the docstring must be a a list of identifiers, separated by spaces, commas, colons, or semicolons.

Parameters

descr: A ParsedDocstring containing a list of identifiers. (type=markup.ParsedDocstring)

Return Value

A list of the identifier names contained in descr. (type=list of string)

Raises

GetValueError If descr does not contain a valid list of identifiers.

_descr_to_docstring_field\(\(arg, descr\)\)

Function Signature Extraction

108
parse_function_signature(func_doc, doc_source, docformat, parse_errors)

Construct the signature for a built-in function or method from its docstring. If the docstring uses the standard convention of including a signature in the first line of the docstring (and formats that signature according to standard conventions), then it will be used to extract a signature. Otherwise, the signature will be set to a single varargs variable named ". . . ."

Parameters

func_doc: The target object where to store parsed signature. Also container of the docstring to parse if doc_source is None (type=RoutineDoc)

doc_source: Contains the docstring to parse. If None, parse func_doc docstring instead (type=APIDoc)

Return Value

None

9.2 Variables

STANDARD_FIELDS

A list of the standard simple fields accepted by epydoc. This list can be augmented at run-time by a docstring with the special @deffield field. The order in which fields are listed here determines the order in which they will be displayed in the output.


Docstring Parsing

DEFAULT_DOCFORMAT

The name of the default markup language used to process docstrings.

Value: 'plaintext'

RETURN_PDS

A ParsedDocstring containing the text 'Returns'. This is used to construct summary descriptions for routines that have empty descr, but non-empty return_descr.

Value: markup.parse('Returns:', markup= 'epytext')

Field Processing Error Messages

UNEXPECTED_ARG

Value: '%r did not expect an argument'
EXPECTED_ARG
Value: '%r expected an argument'

EXPECTED_SINGLE_ARG
Value: '%r expected a single argument'

BAD_CONTEXT
Value: 'Invalid context for %r'

REDEFINED
Value: 'Redefinition of %s'

UNKNOWN_TAG
Value: 'Unknown field tag %r'

BAD_PARAM
Value: '@%s for unknown parameter %s'

Field Processing

_field_dispatch_table
Value: {}

Field Handler Functions

PARAMETER_TAGS
Value: ('arg', 'argument', 'parameter', 'param', 'kwarg', 'keywo...'

VARIABLE_TAGS
Value: ('cvar', 'cvariable', 'ivar', 'ivariable')

EXCEPTION_TAGS
Value: ('raise', 'raises', 'except', 'exception')
9.3 Class DocstringField

A simple docstring field, which can be used to describe specific information about an object, such as its author or its version. Simple docstring fields are fields that take no arguments, and are displayed as simple sections.

9.3.1 Methods

```python
_init_(self, tags, label, plural=None, short=0, multivalue=1, takes_arg=0, varnames=None)
```

```python
_cmp_(self, other)
```

```python
_hash_(self)
```

```python
_repr_(self)
```

9.3.2 Instance Variables

```python
multivalue
```

If true, then multiple values may be given for this field; if false, then this field can only take a single value, and a warning should be issued if it is redefined.

```python
plural
```

The label that should be used to identify this field in the output, if the field contains multiple values.
**short**
If true, then multiple values should be combined into a single comma-delimited list. If false, then multiple values should be listed separately in a bulleted list.

**singular**
The label that should be used to identify this field in the output, if the field contains one value.

**tags**
The set of tags that can be used to identify this field.

**takes_arg**
If true, then this field expects an argument; and a separate field section will be constructed for each argument value. The label (and plural label) should include a '%s' to mark where the argument's string rep should be added.
10 Package epydoc.docwriter

Output generation.
Module epydoc.docwriter.dotgraph

Render Graphviz directed graphs as images. Below are some examples.
See Also: The Graphviz Homepage

11.1 Functions

Graph Generation Functions
package_tree_graph(packages, linker, context=None, **options)

Return a DotGraph that graphically displays the package hierarchies for the given packages.

uml_package_tree_graph(packages, linker, context=None, **options)

Return a DotGraph that graphically displays the package hierarchies for the given packages as a nested set of UML symbols.

class_tree_graph(classes, linker, context=None, **options)

Return a DotGraph that graphically displays the class hierarchy for the given classes. Options:

- exclude: A list of classes that should be excluded
- dir: LR|RL|BT requests a left-to-right, right-to-left, or bottom-to-top, drawing. (corresponds to the dot option 'rankdir')
- max_subclass_depth: The maximum depth to which subclasses will be drawn.
- max_subclasses: A list of ints, specifying how many subclasses should be drawn per class at each level of the graph. E.g., [5,3,1] means draw up to 5 subclasses for the specified classes; up to 3 subsubclasses for each of those (up to) 5 subclasses; and up to 1 subclass for each of those.

_class_tree_graph(graph, classes, mknode, mkedge, linker, context, options, cls2node)

A helper function that is used by both class_tree_graph() and uml_class_tree_graph() to draw class trees. To abstract over the differences between the two, this function takes two callback functions that create graph nodes and edges:

- mknode(base, nodetype, linker, context, options): Returns a DotGraphNode. nodetype is one of: subclass, superclass, selected, undocumented.
- mkedge(begin, end, edgetype, options): Returns a DotGraphEdge. edgetype is one of: subclass, truncate-subclass.

_add_class_tree_superclasses(graph, classes, mknode, mkedge, linker, context, options, cls2node)

_add_class_tree_subclasses(graph, classes, mknode, mkedge, linker, context, options, cls2node, truncated)

_add_class_tree_inheritance(graph, classes, mknode, mkedge, linker, context, options, cls2node, truncated)

_get_subclass_depth_map(classes)
**uml_class_tree_graph**(classes, linker, context=None, **options)

Return a DotGraph that graphically displays the class hierarchy for the given class, using UML notation. Options:

- **exclude**: A list of classes that should be excluded.
- **dir**: LR|RL|BT requests a left-to-right, right-to-left, or bottom-to-top drawing. (corresponds to the dot option ‘rankdir’)
- **max_subclass_depth**: The maximum depth to which subclasses will be drawn.
- **max_subclasses**: A list of ints, specifying how many subclasses should be drawn per class at each level of the graph. E.g., [5,3,1] means draw up to 5 subclasses for the specified classes; up to 3 subsubclasses for each of those (up to) 5 subclasses; and up to 1 subclass for each of those.
- **max_attributes**
- **max_operations**
- **show_private_vars**
- **show_magic_vars**
- **link_attributes**
- **show_signature_defaults**
- **max_signature_width**

**_uml_mknode**(cls, nodetype, linker, context, options)

**_uml_mkedge**(start, end, edgetype, options)

**import_graph**(modules, docindex, linker, context=None, **options)

**call_graph**(api_docs, docindex, linker, context=None, **options)

**Parameters**

- **dir**: rankdir for the graph. (default=LR)
- **add_callers**: also include callers for any of the routines in api_docs. (default=False)
- **add_callees**: also include callees for any of the routines in api_docs. (default=False)

**To Do**: Add an exclude option?

**Dot Version**

**get_dot_version()**
### Helper Functions

**add_valdoc_nodes**\( (\text{graph, val\_docs, linker, context}) \)

| To Do: | Use different node styles for different subclasses of APIDoc |

**mk_valdoc_node**\( (\text{val\_doc, linker, context}) \)

**specialize_valdoc_node**\( (\text{node, val\_doc, context, linker}) \)

| Update the style attributes of \text{node} to reflect its type and context. |

**name_list**\( (\text{api\_docs, context=\text{None}}) \)

### 11.2 Variables

**USE_DOT2TEX**

Should the dot2tex module be used to render dot graphs to latex (if it’s available)? This is experimental, and not yet working, so it should be left False for now.

| Value: | False |

**COLOR**

Colors for graphs of APIDocs

| Value: | \{\'BASECLASS\_BG\': \'#e0b0a0\', \'CLASS\_BG\': \'#d8ffe8\', \'INH\_L... |

### Dot Graphs

**DOT\_COMMAND**

The command that should be used to spawn dot

| Value: | \'dot\' |

### Dot Version

**_dot\_version**

| Value: | None |

**_DOT\_VERSION\_RE**

| Value: | \text{re.compile}(r\'dot version ([\d.]+)\') |
A *dot* directed graph. The contents of the graph are constructed from the following instance variables:

- **nodes**: A list of `DotGraphNode`s, encoding the nodes that are present in the graph. Each node is characterized by a set of attributes, including an optional label.

- **edges**: A list of `DotGraphEdge`s, encoding the edges that are present in the graph. Each edge is characterized by a set of attributes, including an optional label.

- **node_defaults**: Default attributes for nodes.

- **edge_defaults**: Default attributes for edges.

- **body**: A string that is appended as-is in the body of the graph. This can be used to build more complex dot graphs.

The `link()` method can be used to resolve crossreference links within the graph. In particular, if the 'href' attribute of any node or edge is assigned a value of the form `<name>`, then it will be replaced by the URL of the object with that name. This applies to the `body` as well as the `nodes` and `edges`.

To render the graph, use the methods `write()` and `render()`. Usually, you should call `link()` before you render the graph.

### 11.3.1 Methods

```python
__init__(self, title, body='', node_defaults=None, edge_defaults=None, caption=None)
```

Create a new `DotGraph`.

**Overrides**: `object.__init__`
to_latex(self, directory, center=True, size=None)

Return the LaTeX code that should be used to display this graph. Two image files will be written: `image_file+.eps` and `image_file+.pdf`.

Parameters:
- `size`: The maximum size for the generated image, in inches. In particular, if `size` is "w,h", then this will add a line `size="w,h"` to the dot graph. Defaults to DEFAULT_LATEX_SIZE. (type=str)

_to_dot2tex(self, center=True, size=None)

_to_html(self, directory, center=True, size=None)

Return the HTML code that should be used to display this graph (including a client-side image map).

Parameters:
- `image_url`: The URL of the image file for this graph; this should be generated separately with the write() method.
- `size`: The maximum size for the generated image, in inches. In particular, if `size` is "w,h", then this will add a line `size="w,h"` to the dot graph. Defaults to DEFAULT_HTML_SIZE. (type=str)

link(self, docstring_linker)

Replace any href attributes whose value is `<name>` with the url of the object whose name is `<name>`.

_link_href(self, attribs, docstring_linker)

Helper for link()

write(self, filename, language=None, size=None)

Render the graph using the output format `language`, and write the result to `filename`.

Parameters:
- `size`: The maximum size for the generated image, in inches. In particular, if `size` is "w,h", then this will add a line `size="w,h"` to the dot graph. If not specified, no size line will be added. (type=str)

Return Value:
- True if rendering was successful.

_pick_language(self, filename)
render(self, language=None, size=None)

Use the dot command to render this graph, using the output format language. Return the result as a string, or None if the rendering failed.

Parameters:
size: The maximum size for the generated image, in inches. In particular, if size is "w,h", then this will add a line size="w,h" to the dot graph. If not specified, no size line will be added. (type=str)

_run_dot(self, *options, **kwparam)

to_dotfile(self, size=None)

Return the string contents of the dot file that should be used to render this graph.

Parameters:
size: The maximum size for the generated image, in inches. In particular, if size is "w,h", then this will add a line size="w,h" to the dot graph. If not specified, no size line will be added. (type=str)

11.3.2 Class Variables

_uids
A set of all uids that that have been generated, used to ensure that each new graph has a unique uid.
Value: set(['class_hierarchy_for_apidoc', 'class_hierarchy_for_a...)

DEFAULT_NODE_DEFAULTS
Value: {'fontname': 'Helvetica', 'fontsize': 10}

DEFAULT_EDGE_DEFAULTS
Value: {'fontname': 'Helvetica', 'fontsize': 10}

DEFAULT_LATEX_SIZE
The default minimum size in inches (width,height) for graphs when rendering with to_latex()
Value: '6.25,8'

DEFAULT_HTML_SIZE
The default minimum size in inches (width,height) for graphs when rendering with to_html()
Value: '10,20'
11.3.3 Instance Variables

**title**
- The title of the graph.

**caption**
- A caption for the graph.

**nodes**
- A list of the nodes that are present in the graph.
  - Type: list of DotGraphNode

**edges**
- A list of the edges that are present in the graph.
  - Type: list of DotGraphEdge

**body**
- A string that should be included as-is in the body of the graph.
  - Type: str

**node_defaults**
- Default attribute values for nodes.

**edge_defaults**
- Default attribute values for edges.

**uid**
- A unique identifier for this graph. This can be used as a filename when rendering the graph. No two DotGraphs will have the same uid.
11.4 Class DotGraphNode

Known Subclasses: epydoc.docwriter.dotgraph.DotGraphUmlClassNode,
                epydoc.docwriter.dotgraph.DotGraphUmlModuleNode

11.4.1 Methods

    __init__(self, label=None, html_label=None, **attrs)

    Overrides: object.__init__ (inherited documentation)

    __getitem__(self, attr)

    __setitem__(self, attr, val)

    to_dotfile(self)

    Return the dot commands that should be used to render this node.

11.4.2 Class Variables

    _next_id

    Value: 0

11.5 Class DotGraphEdge

11.5.1 Methods

    __init__(self, start, end, label=None, **attrs)

    Parameters
        start: (type=DotGraphNode)
        end: (type=DotGraphNode)

    Overrides: object.__init__
__getitem__(self, attr)

__setitem__(self, attr, val)

to_dotfile(self)

Return the dot commands that should be used to render this edge.

11.5.2 Instance Variables

start
  Type: DotGraphNode

dot

end
  Type: DotGraphNode

11.6 Class DotGraphUmlClassNode

A specialized dot graph node used to display ClassDocs using UML notation. The node is rendered as a table with three cells: the top cell contains the class name; the middle cell contains a list of attributes; and the bottom cell contains a list of operations:

<table>
<thead>
<tr>
<th>ClassDoc</th>
</tr>
</thead>
<tbody>
<tr>
<td>x: int</td>
</tr>
<tr>
<td>...</td>
</tr>
<tr>
<td>f(self, x)</td>
</tr>
<tr>
<td>...</td>
</tr>
</tbody>
</table>

DotGraphUmlClassNodes may be collapsed, in which case they are drawn as a simple box containing the class name:

<table>
<thead>
<tr>
<th>ClassDoc</th>
</tr>
</thead>
</table>

Attributes with types corresponding to documented classes can optionally be converted into edges, using link_attributes().
To Do: Add more options? - show/hide operation signature - show/hide operation signature types - show/hide operation signature return type - show/hide attribute types - use qualifiers
11.6.1 Methods

```python
__init__(self, class_doc, linker, context, collapsed=False, bgcolor='#d8ffe8', **options)
```

Create a new `DotGraphUmlClassNode` based on the class `class_doc`.

**Parameters**

- **linker**: Used to look up URLs for classes. *(type=markup.DocstringLinker)*
- **context**: The context in which this node will be drawn; dotted names will be contextualized to this context. *(type=APIDoc)*
- **collapsed**: If true, then display this node as a simple box. *(type=bool)*
- **bgcolor**: The background color for this node. *(type=str)*
- **options**: A set of options used to control how the node should be displayed. *(type=dict)*
- **show_private_vars**: If false, then private variables are filtered out of the attributes & operations lists. (Default: False)
- **show_magic_vars**: If false, then magic variables (such as `__init__` and `__add__`) are filtered out of the attributes & operations lists. (Default: True)
- **show_inherited_vars**: If false, then inherited variables are filtered out of the attributes & operations lists. (Default: False)
- **max_attributes**: The maximum number of attributes that should be listed in the attribute box. If the class has more than this number of attributes, some will be ellided. Ellipsis is marked with '...'. (Default: 10)
- **max_operations**: The maximum number of operations that should be listed in the operation box. (Default: 5)
- **add_nodes_for_linked_attributes**: If true, then `link_attributes()` will create new a collapsed node for the types of a linked attributes if no node yet exists for that type.
- **show_signature_defaults**: If true, then show default parameter values in method signatures; if false, then hide them. (Default: False)
- **max_signature_width**: The maximum width (in chars) for method signatures. If the signature is longer than this, then it will be truncated (with '...'). (Default: 60)

**Overrides**: `epydoc.docwriter.dotgraph.DotGraphNode.__init__`
Attribute Linking

link_attributes(self, graph, nodes)
Convert any attributes with type descriptions corresponding to documented classes to edges. The following type descriptions are currently handled:

- Dotted names: Create an attribute edge to the named type, labelled with the variable name.
- Collections: Create an attribute edge to the named type, labelled with the variable name, and marked with ‘*’ at the type end of the edge.
- Mappings: Create an attribute edge to the named type, labelled with the variable name, connected to the class by a qualifier box that contains the key type description.
- Optional: Create an attribute edge to the named type, labelled with the variable name, and marked with '0..1' at the type end of the edge.

The edges created by link_attributes() are handled internally by DotGraphUmlClassNode; they should not be added directly to the DotGraph.

Parameters
nodes: A dictionary mapping from ClassDocs to DotGraphUmlClassNodes, used to look up the nodes for attribute types. If the add_nodes_for_linked_attributes option is used, then new nodes will be added to this dictionary for any types that are not already listed. These added nodes must be added to the DotGraph.

_link_attribute(self, var, graph, nodes)
Helper for link_attributes(): try to convert the attribute variable var into an edge, and add that edge to self.edges. Return True iff the variable was successfully converted to an edge (in which case, it should be removed from the attributes list).

_add_attribute_edge(self, var, graph, nodes, type_str, **attribs)
Helper for link_attributes(): try to add an edge for the given attribute variable var. Return True if successful.

Helper Methods

_summary(self, api_doc)
Return a plaintext summary for api_doc

_type_descr(self, api_doc)
Return a plaintext type description for api_doc
_tooltip(self, var_doc)

Return a tooltip for var_doc.

Rendering

_attribute_cell(self, var_doc)

_operation_cell(self, var_doc)

To Do:
- do 'word wrapping' on the signature, by starting a new row in the table, if necessary. How
to indent the new line? Maybe use align=right? I don’t think dot has a &nbsp;.
- Optionally add return type info?

_operation_arg(self, name, default, func_doc)

To Do:
- Handle tuple args better
- Optionally add type info?

_qualifier_cell(self, key_label, port)

_get_html_label(self)

to_dotfile(self)

Return the dot commands that should be used to render this node.

Overrides: epydoc.docwriter.dotgraph.DotGraphNode.to_dotfile (inherited documentation)

11.6.2 Class Variables

Inherited from epydoc.docwriter.dotgraph.DotGraphNode(Section 11.4, p. 124): _next_id

Attribute Linking

SIMPLE_TYPE_RE

A regular expression that matches descriptions of simple types.

Value: re.compile(r"^\w\.+$/)
<table>
<thead>
<tr>
<th>Collection Type</th>
<th>Description</th>
<th>Value</th>
</tr>
</thead>
</table>
| **COLLECTION_TYPE_RE** | A regular expression that matches descriptions of collection types. | `re.compile(r'^list|set|sequence|tuple|collection) of (\[\...

<table>
<thead>
<tr>
<th>Mapping Type</th>
<th>Description</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>MAPPING_TYPE_RE</strong></td>
<td>A regular expression that matches descriptions of mapping types.</td>
<td>`re.compile(r'^dict</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Mapping To Collection Type</th>
<th>Description</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>MAPPING_TO_COLLECTION_TYPE_RE</strong></td>
<td>A regular expression that matches descriptions of mapping types whose value type is a collection.</td>
<td>`re.compile(r'^dict</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Optional Type</th>
<th>Description</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>OPTIONAL_TYPE_RE</strong></td>
<td>A regular expression that matches descriptions of optional types.</td>
<td>`re.compile(r'^None or</td>
</tr>
</tbody>
</table>

### Rendering

<table>
<thead>
<tr>
<th>Cell Type</th>
<th>Description</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>ATTRIBUTE_CELL</strong></td>
<td>args: (url, tooltip, label)</td>
<td>`\n &lt;TR&gt;&lt;TD ALIGN=&quot;LEFT&quot; HREF=&quot;%s&quot; TOOLTIP=&quot;%s&quot;&gt;%s&lt;/TD...</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Cell Type</th>
<th>Description</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>OPERATION_CELL</strong></td>
<td>args: (url, tooltip, label)</td>
<td>`\n &lt;TR&gt;&lt;TD ALIGN=&quot;LEFT&quot; HREF=&quot;%s&quot; TOOLTIP=&quot;%s&quot;&gt;%s&lt;/TD...</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Cell Type</th>
<th>Description</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>QUALIFIER_CELL</strong></td>
<td>args: (port, bgcolor, label)</td>
<td>`\n &lt;TR&gt;&lt;TD VALIGN=&quot;BOTTOM&quot; PORT=&quot;%s&quot; BGCOLOR=&quot;%s&quot; BOR...</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Cell Type</th>
<th>Description</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>QUALIFIER_DIV</strong></td>
<td></td>
<td>`\n &lt;TR&gt;&lt;TD VALIGN=&quot;BOTTOM&quot; HEIGHT=&quot;10&quot; WIDTH=&quot;10&quot; FIX...</td>
</tr>
</tbody>
</table>
11.6.3 Instance Variables

**class_doc**

The class represented by this node.

**linker**

Used to look up URLs for classes.

**context**

The context in which the node will be drawn.

**bgcolor**

The background color of the node.

**options**

Options used to control how the node is displayed.

**collapsed**

If true, then draw this node as a simple box.

**attributes**

The list of VariableDocs for attributes

**operations**

The list of VariableDocs for operations

**qualifiers**

List of (key, label, port) tuples.
edges
List of edges used to represent this node’s attributes. These should not be added to the DotGraph; this node will generate their dotfile code directly.

same_rank
List of nodes that should have the same rank as this one. (Used for nodes that are created by _link_attributes).

11.7 Class DotGraphUmlModuleNode

A specialized dot graph node used to display ModuleDocs using UML notation. Simple module nodes look like:

```
.----.
| modulename |
+------------+
```

Packages nodes are drawn with their modules & subpackages nested inside:

```
.----.
| packagename |
| .----. .----. .----. |
| +---------+ +---------+ +---------+ |
| | module1 | | module2 | | module3 | |
| +---------+ +---------+ +---------+ |
| +------------------------------------+
```

11.7.1 Methods

```
__init__(self, module_doc, linker, context, collapsed=False,
excluded_submodules=(), **options)
```

Overrides: epydoc.docwriter.dotgraph.DotGraphNode.__init__
_get_html_label(self, package)

Return Value

(label, depth, width) where:

• label is the HTML label
• depth is the depth of the package tree (for coloring)
• width is the max width of the HTML label, roughly in units of characters.

color(self, package, depth)

to_dotfile(self)

Return the dot commands that should be used to render this node.

Overrides: epydoc.docwriter.dotgraph.DotGraphNode.to_dotfile (inherited documentation)

Inherited from epydoc.docwriter.dotgraph.DotGraphNode(Section 11.4, p. 124): __getitem__(), __setitem__()
12 Module epydoc.docwriter.html

The HTML output generator for epydoc. The main interface provided by this module is the HTMLWriter class.

To Do: Add a cache to HTMLWriter.url()?

12.1 Functions

**compile_template**(docstring, template_string, output_function='out', debug=True)

Given a template string containing inline python source code, return a python function that will fill in the template, and output the result. The signature for this function is taken from the first line of docstring. Output is generated by making repeated calls to the output function with the given name (which is typically one of the function’s parameters).

The templating language used by this function passes through all text as-is, with three exceptions:

- If every line in the template string is indented by at least \( x \) spaces, then the first \( x \) spaces are stripped from each line.
- Any line that begins with ‘\>>>’ (with no indentation) should contain python code, and will be inserted as-is into the template-filling function. If the line begins a control block (such as ‘if’ or ‘for’), then the control block will be closed by the first ‘\>>>’-marked line whose indentation is less than or equal to the line’s own indentation (including lines that only contain comments.)
- In any other line, any expression between two ‘$’ signs will be evaluated and inserted into the line (using `str()` to convert the result to a string).

Here is a simple example:

```python
>>> TEMPLATE = r'
...
... <book>
... <title>$book.title$</title>
... <pages>$book.count_pages()$</pages>
... >>> for chapter in book.chapters:
... ... <chaptername>$chapter.name$</chaptername>
... ... #endfor
... </book>
>>> write_book = compile_template('write_book(out, book)', TEMPLATE)
```

**strip_indent**(s)

Given a multiline string \( s \), find the minimum indentation for all non-blank lines, and return a new string formed by stripping that amount of indentation from all lines in \( s \).
Class HTMLWriter

12.2.1 Methods

__init__(self, docindex, **kwargs)

Construct a new HTML writer, using the given documentation index.

Parameters:
- docindex: The documentation index.
- prj_name: The name of the project. Defaults to none. (type=string)
- prj_url: The target for the project homepage link on the navigation bar. If prj_url is not specified, then no hyperlink is created. (type=string)
- prj_link: The label for the project link on the navigation bar. This link can contain arbitrary HTML code (e.g. images). By default, a label is constructed from prj_name. (type=string)
- top_page: The top page for the documentation. This is the default page shown main frame, when frames are enabled. top can be a URL, the name of a module, the name of a class, or one of the special strings "trees.html", "indices.html", or "help.html". By default, the top-level package or module is used, if there is one; otherwise, "trees" is used. (type=string)
- css: The CSS stylesheet file. If css is a file name, then the specified file's contents will be used. Otherwise, if css is the name of a CSS stylesheet in epydoc.docwriter.html_css, then that stylesheet will be used. Otherwise, an error is reported. If no stylesheet is specified, then the default stylesheet is used. (type=string)
- help_file: The name of the help file. If no help file is specified, then the default help file will be used. (type=string)
- show_private: Whether to create documentation for private objects. By default, private objects are documented. (type=boolean)
- show_frames: Whether to create a frames-based table of contents. By default, it is produced. (type=boolean)
- show_imports: Whether or not to display lists of imported functions and classes. By default, they are not shown. (type=boolean)
- variable_maxlines: The maximum number of lines that should be displayed for the value of a variable in the variable details section. By default, 8 lines are displayed. (type=int)
- variable_linelength: The maximum line length used for displaying the values of variables in the variable details sections. If a line is longer than this length, then it will be wrapped to the next line. The default line length is 70 characters. (type=int)
- variable_summary_linelength: The maximum line length used for displaying the values of variables in the summary section. If a line is longer than this length, then it will be truncated. (type=int)
Class HTMLWriter Module epydoc.docwriter.html

_find_top_page(self, pagename)

Find the top page for the API documentation. This page is used as the default page shown in the
main frame, when frames are used. When frames are not used, this page is copied to index.html.

Parameters
pagename: The name of the page, as specified by the keyword argument top to the constructor. (type=str)

Return Value
The URL of the top page. (type=str)

1. Interface Methods

write(self, directory=None)

Write the documentation to the given directory.

Parameters
directory: The directory to which output should be written. If no directory is
specified, output will be written to the current directory. If the directory
does not exist, it will be created. (type=str)

Return Value
None

Raises
OSError If directory cannot be created.
OSError If any file cannot be created or written to.

_write(self, write_func, directory, filename, *args)

_mkdir(self, directory)

If the given directory does not exist, then attempt to create it.

Return Value
None

2.1. Module Pages

write_module(self, out, doc)

Write an HTML page containing the API documentation for the given module to out.

Parameters
doc: A ModuleDoc containing the API documentation for the module that should be
described.

2.??. Source Code Pages

write_sourcecode(self, out, doc, name_to_docs)
2.2. Class Pages

write_class(self, out, doc)

Write an HTML page containing the API documentation for the given class to out.

Parameters
- doc: A ClassDoc containing the API documentation for the class that should be described.

write_class_tree_graph(self, out, doc, graphmaker)

Write HTML code for a class tree graph of doc (a classdoc), using graphmaker to draw the actual graph. graphmaker should be class_tree_graph(), or uml_class_tree_graph(), or any other function with a compatible signature.

If the given class has any private subclasses (including recursive subclasses), then two graph images will be generated – one to display when private values are shown, and the other to display when private values are hidden.

2.3. Trees pages

write_module_tree(self, out)

write_class_tree(self, out)

Write HTML code for a nested list showing the base/subclass relationships between all documented classes. Each element of the top-level list is a class with no (documented) bases; and under each class is listed all of its subclasses. Note that in the case of multiple inheritance, a class may appear multiple times.

To Do: For multiple inheritance, don’t repeat subclasses the second time a class is mentioned; instead, link to the first mention.

write_treepage_header(self, out, title, url)

2.4. Index pages

write_link_index(self, out, indices, title, url, index_by_section, sections='ABCDEFGHIJKLMNOPQRSTUVWXYZ_ ', section_url='#%s')

write_metadata_index(self, out, indices, field, title, typ)

Write an HTML page containing a metadata index.

write_indexpage_header(self, out, indices, title, url)

A helper for the index page generation functions, which generates a header that can be used to navigate between the different indices.
write_index_section(self, out, items, add_blankline=False)

2.5. Help Page

write_help(self, out)
Write an HTML help file to the given stream. If self._helpfile contains a help file, then use it; otherwise, use the default helpfile from epydoc.docwriter.html_help.

2.6. Frames-based Table of Contents

write_frames_index(self, out)
Write the frames index file for the frames-based table of contents to the given streams.

write_toc(self, out)

write_toc_section(self, out, name, docs, fullname=True)

write_project_toc(self, out)

write_module_toc(self, out, doc)
Write an HTML page containing the table of contents page for the given module to the given streams. This page lists the modules, classes, exceptions, functions, and variables defined by the module.

2.7. Project homepage (index.html)

write_homepage(self, directory)
Write an index.html file in the given directory. The contents of this file are copied or linked from an existing page, so this method must be called after all pages have been written. The page used is determined by _frames_index and _top_page:

- If _frames_index is true, then frames.html is copied.
- Otherwise, the page specified by _top_page is copied.

write_redirect_index(self, out, top, name)

2.8. Stylesheet (epydoc.css)
write_css(self, directory, cssname)

Write the CSS stylesheet in the given directory. If cssname contains a stylesheet file or name (from epydoc.docwriter.html_css), then use that stylesheet; otherwise, use the default stylesheet.

Return Value
None

2.9. Javascript (epydoc.js)

write_javascript(self, directory)

2.10. Graphs

render_graph(self, graph)

render_callgraph(self, callgraph, token='')

Render the HTML chunk of a callgraph.

If callgraph is a string, use the _callgraph_cache to return a pre-rendered HTML chunk. This mostly avoids to run dot twice for the same callgraph. Else, run the graph and store its HTML output in the cache.

Parameters
- callgraph: The graph to render or its uid. (type=DotGraph or str)
- token: A string that can be used to make the <div> id unambiguous, if the callgraph is used more than once in a page. (type=str)

Return Value
The HTML representation of the graph.(type=str)

callgraph_link(self, callgraph, token='')

Render the HTML chunk of a callgraph link.

The link can toggles the visibility of the callgraph rendered using render_callgraph with matching parameters.

Parameters
- callgraph: The graph to render or its uid. (type=DotGraph or str)
- token: A string that can be used to make the <div> id unambiguous, if the callgraph is used more than once in a page. (type=str)

Return Value
The HTML representation of the graph link.(type=str)

2.11. Images

write_images(self, directory)
3.1. Page Header

`write_header(self, out, title)`
Generate HTML code for the standard page header, and write it to `out`. `title` is a string containing the page title. It should be appropriately escaped/encoded.

3.2. Page Footer

`write_footer(self, out, short=False)`
Generate HTML code for the standard page footer, and write it to `out`.

3.3. Navigation Bar

`write_navbar(self, out, context)`
Generate HTML code for the navigation bar, and write it to `out`. The navigation bar typically looks like:

```
[ Home Trees Index Help Project ]
```

**Parameters**

`context`: A value indicating what page we’re generating a navigation bar for. If we’re generating an API documentation page for an object, then `context` is a `ValueDoc` containing the documentation for that object; otherwise, `context` is a string name for the page. The following string names are recognized: `tree`, `index`, and `help`.

3.4. Breadcrumbs

`write_breadcrumbs(self, out, context, context_url)`
Generate HTML for the breadcrumbs line, and write it to `out`. The breadcrumbs line is an invisible table with a list of pointers to the current object’s ancestors on the left; and the show/hide private selector and the frames/noframes selector on the right.

**Parameters**

`context`: The API documentation for the object whose breadcrumbs we should generate. (`type=ValueDoc`)
**`write_summary_table`**

Generate HTML code for a summary table, and write it to `out`. A summary table is a table that includes a one-row description for each variable (of a given type) in a module or class.

**Parameters**

- **heading:** The heading for the summary table; typically, this indicates what kind of value the table describes (e.g., functions or classes).
- **doc:** A `ValueDoc` object containing the API documentation for the module or class whose variables we should summarize.
- **value_type:** A string indicating what type of value should be listed in this summary table. This value is passed on to `doc`'s `select_variables()` method.

**`write_summary_group`**

**`write_inheritance_list`**

**`write_var_list`**

**`write_summary_line`**

Generate HTML code for a single line of a summary table, and write it to `out`. See `write_summary_table` for more information.

**Parameters**

- **var_doc:** The API documentation for the variable that should be described by this line of the summary table.
- **container:** The API documentation for the class or module whose summary table we’re writing.

**`_write_summary_line`**

3.6. Details Lists

**`write_details_list`**

**`write_details_entry`**

**`labelled_list_item`**

**`property_accessor_to_html`**
arg_name_to_html(self, func_doc, arg_name)

A helper function used to format an argument name, for use in the argument description list under a routine's details entry. This just wraps strong & code tags around the arg name; and if the arg name is associated with a type, then adds it parenthetically after the name.

write_function_details_entry(self, out, var_doc, descr, callgraph, rtype, rdescr, arg_descrs, div_class)

write_property_details_entry(self, out, var_doc, descr, accessors, div_class)

write_variable_details_entry(self, out, var_doc, descr, div_class)

variable_tooltip(self, var_doc)

pprint_value(self, val_doc)

Base Tree

base_tree(self, doc, width=None, postfix='', context=None)

Return Value

The HTML code for a class's base tree. The tree is drawn 'upside-down' and right justified, to allow for multiple inheritance. (type=string)

find_tree_width(self, doc, context)

Helper function for base_tree.

Return Value

The width of a base tree, when drawn right-justified. This is used by base_tree to determine how far to indent lines of the base tree. (type=int)

contextual_label(self, doc, context)

Return the label for doc to be shown in context.
function_signature(self, api_doc, is_summary=False, link_name=False, anchor=False, context=None)

Render a function signature in HTML.

Parameters
api_doc: The object whose name is to be rendered. If a VariableDoc, its value should be a RoutineDoc (type=VariableDoc or RoutineDoc)

is_summary: True if the function is to be rendered in the summary. type css_class: bool

link_name: If True, the name is a link to the object anchor. (type=bool)

anchor: If True, the name is the object anchor. (type=bool)

context: If set, represent the function name from this context. Only useful when api_doc is a RoutineDoc. (type=DottedName)

Return Value
The HTML code for the object. (type=str)

summary_name(self, api_doc, css_class='summary-name', link_name=False, anchor=False)

Render an object name in HTML.

Parameters
api_doc: The object whose name is to be rendered (type=APIDoc)

css_class: The CSS class to assign to the rendered name type css_class: str

link_name: If True, the name is a link to the object anchor. (type=bool)

anchor: If True, the name is the object anchor. (type=bool)

Return Value
The HTML code for the object. (type=str)

func_arg(self, name, default, css_class)

_arg_name(self, arg)

Import Lists

write_imports(self, out, doc)

_import(self, var_doc, context)

Module Trees
write_module_list(self, out, doc)

write_module_tree_item(self, out, doc, package=None)

Class trees

write_class_tree_item(self, out, doc, class_set)

Standard Fields

write_standard_fields(self, out, doc)

Write HTML code containing descriptions of any standard markup fields that are defined by the given APIDoc object (such as @author and @todo fields).

Parameters

doc: The APIDoc object containing the API documentation for the object whose standard markup fields should be described.

write_standard_field(self, out, doc, field, descrs, arg=None)

Index generation

build_identifier_index(self)

_group_by_letter(self, items)

Preserves sort order of the input.

build_term_index(self)

_terms_from_docstring(self, base_url, container, parsed_docstring)

build_metadata_index(self, field_name)

_term_index_to_anchor(self, term)

Given the name of an inline index item, construct a URI anchor. These anchors are used to create links from the index page to each index item.

Redirect page
**write_redirect_page**\( (\textit{self}, \textit{out}) \)

Build the auto-redirect page, which translates dotted names to URLs using javascript. When the user visits \(<\text{redirect.html}\#\text{dotted.name}>\), they will automatically get redirected to the page for the object with the given fully-qualified dotted name. E.g., for epydoc, \(<\text{redirect.html}\#\text{epydoc.apidoc.UNKNOWN}>\) redirects the user to \(<\text{epydoc.apidoc-module.html}\#\text{UNKNOWN}>\).

**_write_redirect_page**\( (\textit{self}, \textit{out}, \textit{pages}) \)

**write_api_list**\( (\textit{self}, \textit{out}) \)

Write a list of mapping name-url for all the documented objects.

**write_url_record**\( (\textit{self}, \textit{out}, \textit{obj}) \)

**Helper functions**

**_val_is_public**\( (\textit{self}, \textit{valdoc}) \)

Make a best-guess as to whether the given class is public.

**write_table_header**\( (\textit{self}, \textit{out}, \textit{css_class}, \textit{heading}=\text{None}, \textit{private_link}=\text{True}, \textit{colspan}=2) \)

**write_group_header**\( (\textit{self}, \textit{out}, \textit{group}, \textit{tr_class}=\text{■}) \)

**url**\( (\textit{self}, \textit{obj}) \)

Return the URL for the given object, which can be a \texttt{VariableDoc}, a \texttt{ValueDoc}, or a \texttt{DottedName}.

**_url**\( (\textit{self}, \textit{obj}) \)

Internal helper for \texttt{url}.

**pysrc_link**\( (\textit{self}, \textit{api_doc}) \)

**pysrc_url**\( (\textit{self}, \textit{api_doc}) \)
href(self, target, label=None, css_class=None, context=None, tooltip=None)

Return the HTML code for an HREF link to the given target (which can be a VariableDoc, a ValueDoc, or a DottedName. If a NamespaceDoc context is specified, the target label is contextualized to it.

_attr_to_html(self, attr, api_doc, indent)

summary(self, api_doc, indent=0)

descr(self, api_doc, indent=0)

type_descr(self, api_doc, indent=0)

return_type(self, api_doc, indent=0)

return_descr(self, api_doc, indent=0)

docstring_to_html(self, parsed_docstring, where=None, indent=0)

description(self, parsed_docstring, where=None, indent=0)

doc_kind(self, doc)

_doc_or_ancestor_is_private(self, api_doc)

_private_subclasses(self, class_doc)

Return a list of all subclasses of the given class that are private, as determined by _val_is_public. Recursive subclasses are included in this list.

12.2.2 Class Variables

2.4. Index pages

SPLIT_IDENT_INDEX_SIZE

If the identifier index has more than this number of entries, then it will be split into separate pages, one for each alphabetical section.

Value: 3000
LETTERS
The alphabetical sections that are used for link index pages.
Value: 'ABCDEFGHIJKLMNOPQRSTUVWXYZ_'

2.9. Javascript (epydoc.js)

TOGGLE_PRIVATE_JS
A javascript that is used to show or hide the API documentation for private objects. In order for this to work correctly, all documentation for private objects should be enclosed in <div class="private">...</div> elements.
Value: 'function toggle_private() {
 // Search for any p...

GET_COOKIE_JS
A javascript that is used to read the value of a cookie. This is used to remember whether private variables should be shown or hidden.
Value: 'function getCookie(name) {
 var dc = document.co...

SET_FRAME_JS
A javascript that is used to set the contents of two frames at once. This is used by the project table-of-contents frame to set both the module table-of-contents frame and the main frame when the user clicks on a module.
Value: 'function setFrame(url1, url2) {
 parent.frames...

HIDE_PRIVATE_JS
A javascript that is used to hide private variables, unless either: (a) the cookie says not to; or (b) we appear to be linking to a private variable.
Value: 'function checkCookie() {
 var cmd=getCookie("Epy...

TOGGLE_CALLGRAPH_JS
Value: 'function toggleCallGraph(id) {
 var elt = docume...

SHOW_PRIVATE_JS
Value: 'function show_private() {
 var elts = document.g...

GET_ANCHOR_JS
Value: 'function get_anchor() {
 var href = location.h...
REDIRECT_URL_JS
A javascript that is used to implement the auto-redirect page. When the user visits `<redirect.html#dotted.name>`, they will automatically get redirected to the page for the object with the given fully-qualified dotted name. E.g., for epydoc, `<redirect.html#epydoc.apidoc.UNKNOWN>` redirects the user to `<epydoc.apidoc-module.html#UNKNOWN>`.

Value: `function redirect_url(dottedName) {\n  // Scan t...`

2.10. Graphs

RE_CALLGRAPH_ID
Value: `re.compile(r'\"[\"\'](.+-div)[\"\']\')`

2.11. Images

IMAGES
Value: `{'crarr.png': 'iVBORw0KGgoAAAANSUhEUgAAABEAAAAKCAMAAABlok...`

3.6. Details Lists

SPECIAL_METHODS
Value: `{'_add_': 'Addition operator', '_and_': 'And operator...`

Index generation

METADATA_INDICES
A list of metadata indices that should be generated. Each entry in this list is a tuple (tag, label, short_label), where tag is the cannonical tag of a metadata field; label is a label for the index page; and short_label is a shorter label, used in the index selector.

Value: `[('bug', 'Bug List', 'Bugs'), ('todo', 'To Do List', 'To ...`

Helper functions

TABLE FOOTER
Value: `</table>\n`
PRIVATE_LINK
Value: `<span class="options">[<a href="javascript:void(0);" cla...``

_url_cache
Value: `{}`

12.2.3 Instance Variables

_show_private
Should private docs be included?

_prj_name
The project's name (for the project link in the navbar)

_prj_url
URL for the project link in the navbar

_prj_link
HTML code for the project link in the navbar

_top_page
The 'main' page

_css
CSS stylesheet to use

_helpfile
Filename of file to extract help contents from

_frames_index
Should a frames index be created?

_show_imports
Should imports be listed?
<table>
<thead>
<tr>
<th>Variable/Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><em>propfunc_linelen</em></td>
<td>[XXX] Not used!</td>
</tr>
<tr>
<td><em>variable_maxlines</em></td>
<td>Max lines for variable values</td>
</tr>
<tr>
<td><em>variable_linelen</em></td>
<td>Max line length for variable values</td>
</tr>
<tr>
<td><em>variable_summary_linelen</em></td>
<td>Max length for variable value summaries</td>
</tr>
<tr>
<td><em>variable_tooltip_linelen</em></td>
<td>Max length for variable tooltips</td>
</tr>
<tr>
<td><em>inheritance</em></td>
<td>How should inheritance be displayed? 'listed', 'included', or 'grouped'</td>
</tr>
<tr>
<td><em>incl_sourcecode</em></td>
<td>Should pages be generated for source code of modules?</td>
</tr>
<tr>
<td><em>mark_docstrings</em></td>
<td>Wrap <code>&lt;span class='docstring'&gt;&lt;/span&gt;</code> around docstrings?</td>
</tr>
<tr>
<td><em>graph_types</em></td>
<td>Graphs that we should include in our output.</td>
</tr>
<tr>
<td><em>include_log</em></td>
<td>Are we generating an HTML log page?</td>
</tr>
<tr>
<td><em>include_timestamp</em></td>
<td>Include a timestamp on the generated docs?</td>
</tr>
<tr>
<td><em>src_code_tab_width</em></td>
<td>Number of spaces to replace each tab with in source code listings.</td>
</tr>
</tbody>
</table>
_callgraph_cache
Map the callgraph uid to their HTML representation.

_redundant_details
If true, then include objects in the details list even if all info about them is already provided by the summary table.

_show_submodule_list
If true, the include a list of submodules on the package documentation page.

_google_analytics
A tracker code for google analytics; or None

module_list
The list of ModuleDocs for the documented modules.

module_set
The set of ModuleDocs for the documented modules.

class_list
The list of ClassDocs for the documented classes.

class_set
The set of ClassDocs for the documented classes.

routine_list
The list of RoutineDocs for the documented routines.

indexed_docs
The list of APIDocs for variables and values that should be included in the index.

12.3 Class _HTMLDocstringLinker

```
markup.DocstringLinker

 class _HTMLDocstringLinker
```
12.3.1 Methods

```
__init__(self, htmlwriter, container)
```

```
translate_indexterm(self, indexterm)
```

Translate an index term to the appropriate output format. The output will typically include a crossreference anchor.

**Parameters**
- `indexterm`: The index term to translate.

**Return Value**
- The translated index term. *(type=string)*

*Overrides:* `epydoc.markup.DocstringLinker.translate_indexterm (inherited documentation)*

```
translate_identifier_xref(self, identifier, label=None)
```

Translate a crossreference link to a Python identifier to the appropriate output format. The output will typically include a reference or pointer to the crossreference target.

**Parameters**
- `identifier`: The name of the Python identifier that should be linked to.
- `label`: The label that should be used for the identifier, if it’s different from the name of the identifier. This should be expressed in the target markup language – e.g. for latex, "_"s should be escaped.

**Return Value**
- The translated crossreference link. *(type=string)*

*Overrides:* `epydoc.markup.DocstringLinker.translate_identifier_xref (inherited documentation)*

```
url_for(self, identifier)
```

Given an identifier, return a URL pointing at that identifier. This is used to create hyperlinks in dotgraphs. This method is *optional* – i.e., it may raise `NotImplementedError`

*Overrides:* `epydoc.markup.DocstringLinker.url_for (inherited documentation)*

```
_failed_xref(self, identifier)
```

Add an identifier to the htmlwriter’s failed crossreference list.
13 Module epydoc.docwriter.html_colorize

Functions to produce colorized HTML code for various objects. Currently, html_colorize defines functions to colorize Python source code.

13.1 Variables

<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>PYSRC_JAVASCRIPTS</td>
<td>Javascript code for the PythonSourceColorizer</td>
</tr>
<tr>
<td>Value: function expand(id)</td>
<td><code>var elt = document.getElementById(id); elt.style.color = color; return</code></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
</tr>
</thead>
</table>
| PYSRC_EXPANDTO_JAVASCRIPT    | Value: `<script type="text/javascript">!
     expandto(location...`                                                   |

<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
</tr>
</thead>
</table>
| _HDR                         | Value: `<?xml version="1.0" encoding="ascii"?>
     <!DOCTYPE...`                                                           |

<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>_FOOT</td>
<td>Value: <code>&lt;/body&gt;&lt;/html&gt;</code></td>
</tr>
</tbody>
</table>

13.2 Class PythonSourceColorizer

A class that renders a python module’s source code into HTML pages. These HTML pages are intended to be provided along with the API documentation for a module, in case a user wants to learn more about a particular object by examining its source code. Links are therefore generated from the API documentation to the source code pages, and from the source code pages back into the API documentation.

The HTML generated by PythonSourceColorizer has several notable features:

- CSS styles are used to color tokens according to their type. (See CSS_CLASSES for a list of the different token types that are identified).
- Line numbers are included to the left of each line.
- The first line of each class and function definition includes a link to the API source documentation for that object.
- The first line of each class and function definition includes an anchor that can be used to link directly to that class or function.
- If javascript is enabled, and the page is loaded using the anchor for a class or function (i.e., if the url ends in `#<name>`), then that class or function will automatically be highlighted; and all other classes and function definition blocks will be 'collapsed'. These collapsed blocks can be expanded by clicking on them.
- Unicode input is supported (including automatic detection of `coding:` declarations).
13.2.1 Methods

```python
__init__(self, module_filename, module_name, docindex=None, url_func=None, name_to_docs=None, tab_width=8)
```
Create a new HTML colorizer for the specified module.

Parameters

- **module_filename**: The name of the file containing the module; its text will be loaded from this file.
- **module_name**: The dotted name of the module; this will be used to create links back into the API source documentation.

```python
find_line_offsets(self)
```
Construct the line_offsets table from self.text.

```python
lineno_to_html(self)
```

```python
colorize(self)
```
Return an HTML string that renders the source code for the module that was specified in the constructor.

```python
tokeneater(self, toktype, toktext, (srow, scol), (erow, ecol), line)
```
A callback function used by tokenize.tokenize to handle each token in the module. tokeneater collects tokens into the self.cur_line list until a complete logical line has been formed; and then calls handle_line to process that line.

```python
handle_line(self, line)
```
Render a single logical line from the module, and write the generated HTML to self.out.

Parameters

- **line**: A single logical line, encoded as a list of (toktype, toktext) pairs corresponding to the tokens in the line.

```python
context_name(self, extra=None)
```

```python
doclink(self, name, docs)
```

```python
doc_descr(self, doc, context)
```

```python
doc_kind(self, doc)
```

```python
mark_def(self, s, name)
```
is_docstring(self, line, i)

add_line_numbers(self, s, css_class)

name2url(self, class_name, func_name=None)

13.2.2 Class Variables

**CSS_CLASSES**
A look-up table that is used to determine which CSS class should be used to colorize a given token. The following keys may be used:

- Any token name (e.g., 'STRING')
- Any operator token (e.g., '=' or '@').
- 'KEYWORD' – Python keywords such as 'for' and 'if'
- 'DEFNAME' – the name of a class or function at the top of its definition statement.
- 'BASECLASS' – names of base classes at the top of a class definition statement.
- 'PARAM' – function parameters
- 'DOCSTRING' – docstrings
- 'DECORATOR' – decorator names

If no CSS class can be found for a given token, then it won’t be marked with any CSS class.

Value: ```{ '@': 'py-decorator', 'BASECLASS': 'py-base-class', 'COMM...```
UNICODE_CODING_RE
A regular expression used to pick out the unicode encoding for the source file.
Value: `re.compile(r'.*?\n.*?coding[:=]\s*([-\w.]+)\')`

ADD_DEF_BLOCKS
A configuration constant, used to determine whether or not to add collapsable `<div>` elements for definition blocks.
Value: True

ADD_LINE_NUMBERS
A configuration constant, used to determine whether or not to add line numbers.
Value: True

ADD_TOOLTIPs
A configuration constant, used to determine whether or not to add tooltips for linked names.
Value: True

GUESS_LINK_TARGETS
If true, then try to guess which target is appropriate for linked names; if false, then always open a div asking the user which one they want.
Value: False

_next_uid
Value: 0

_FIX_DECORATOR_RE
A regexp used to move the `<div>` that marks the beginning of a function or method to just before the decorators.
Value: `re.compile(r'(?:<a name="L\d+"></a><tt class="py-li...`
<table>
<thead>
<tr>
<th><strong>module_name</strong></th>
<th>The dotted name of the module we're colorizing.</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>docindex</strong></td>
<td>A docindex, used to create href links from identifiers to the API documentation for their values.</td>
</tr>
<tr>
<td><strong>name_to_docs</strong></td>
<td>A mapping from short names to lists of ValueDoc, used to decide which values an identifier might map to when creating href links from identifiers to the API docs for their values.</td>
</tr>
<tr>
<td><strong>url_func</strong></td>
<td>A function that maps APIDoc -&gt; URL, used to create href links from identifiers to the API documentation for their values.</td>
</tr>
<tr>
<td><strong>pos</strong></td>
<td>The index in text of the last character of the last token we’ve processed.</td>
</tr>
<tr>
<td><strong>line_offsets</strong></td>
<td>A list that maps line numbers to character offsets in text. In particular, line i begins at character line_offset[i] in text. Since line numbers begin at 1, the first element of line_offsets is None.</td>
</tr>
<tr>
<td><strong>cur_line</strong></td>
<td>A list of (toktype, toktext) for all tokens on the logical line that we are currently processing. Once a complete line of tokens has been collected in cur_line, it is sent to handle_line for processing.</td>
</tr>
<tr>
<td><strong>context</strong></td>
<td>A list of the names of the class or functions that include the current block. context has one element for each level of indentation; context[i] is the name of the class or function defined by the i-th level of indentation, or None if that level of indentation doesn’t correspond to a class or function definition.</td>
</tr>
<tr>
<td><strong>context_types</strong></td>
<td>A list, corresponding one-to-one with self.context, indicating the type of each entry. Each element of context_types is one of: ’func’, ’class’, None.</td>
</tr>
<tr>
<td><strong>indents</strong></td>
<td>A list of indentation strings for each of the current block’s indents. I.e., the current total indentation can be found by taking \text{.join}(self.indents).</td>
</tr>
<tr>
<td><strong>lineno</strong></td>
<td>The line number of the line we're currently processing.</td>
</tr>
<tr>
<td>--------------------</td>
<td>----------------------------------------------------------</td>
</tr>
<tr>
<td><strong>def_name</strong></td>
<td>The name of the class or function whose definition started on the previous logical line, or <em>None</em> if the previous logical line was not a class or function definition.</td>
</tr>
<tr>
<td><strong>def_type</strong></td>
<td>The type of the class or function whose definition started on the previous logical line, or <em>None</em> if the previous logical line was not a class or function definition. Can be 'func', 'class', <em>None</em>.</td>
</tr>
<tr>
<td><strong>tab_width</strong></td>
<td>The number of spaces to replace each tab in source code with</td>
</tr>
</tbody>
</table>
14 Module epydoc.docwriter.html_css

Predefined CSS stylesheets for the HTML outputter (epydoc.docwriter.html).

14.1 Functions

_set_colors(template, *dicts)

_rv(match)
Given a regexp match for a color, return the reverse-video version of that color.

Parameters
match: A regular expression match. (type=Match)

Return Value
The reverse-video color. (type=string)

_darken_darks(match)

14.2 Variables

_TEMPLATE
Value: 

/* Epydoc CSS Stylesheet
 *
 * This stylesheet can...

_COLOR_RE
Value: re.compile(r'#(..)(..)(..)')

_WHITE_COLORS
Value: {'body_bg': '#ffffff', 'body_fg': '#000000', 'body_link':...}

_BLUE_COLORS
Value: {'body_bg': '#000070', 'body_fg': '#ffffff', 'body_link':...}

_WHITE
Value: 

/* Epydoc CSS Stylesheet
 *
 * This stylesheet can...

_BLUE
Value: 

/* Epydoc CSS Stylesheet
 *
 * This stylesheet can...
Variables

_MODULE ESCAPE variable

_GREEN
Value: \n
/* Epydoc CSS Stylesheet
 * This stylesheet can...}

_BLACK
Value: \n
/* Epydoc CSS Stylesheet
 * This stylesheet can...}

_GRAYSCALE
Value: \n
/* Epydoc CSS Stylesheet
 * This stylesheet can...}

_STYLESHEETS_ A dictionary mapping from stylesheet names to CSS stylesheets and descriptions. A single stylesheet may have multiple names. Currently, the following stylesheets are defined:

- **default**: The default stylesheet (synonym for **white**).
- **white**: Black on white, with blue highlights (similar to javadoc).
- **blue**: Black on steel blue.
- **green**: Black on green.
- **black**: White on black, with blue highlights
- **grayscale**: Grayscale black on white.
- **none**: An empty stylesheet.

Type: dictionary from string to (string, string)

Value: {'black': \n
/* Epydoc CSS Stylesheet
 * This stylesheet can...}
Module epydoc.docwriter.html_help

Default help file for the HTML outputter (epydoc.docwriter.html).

15.1 Variables

<table>
<thead>
<tr>
<th>HTML_HELP</th>
<th>The contents of the HTML body for the default help page.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Type: string</td>
<td></td>
</tr>
</tbody>
</table>
| Value:                           | \n<h1 class="epydoc">
API Documentation
</h1>

Th...
16  Module  epydoc.docwriter.latex

The LaTeX output generator for epydoc. The main interface provided by this module is the `LatexWriter` class.

**To Do:** Inheritance=listed

### 16.1 Functions

- `_label(doc)`
- `_hyperlink(target, name)`
- `_hypertarget(uid, sig)`
- `_dotted(name)`
- `find_latex_error(s)`
- `show_latex_warnings(s)`

### 16.2 Variables

- **LATEX_WARNING_RE**
  Value: `re.compile(r'(\?im)(?P<file>[.a-zA-Z_-/\0-9]+)[.\n][a-...`  
- **OVERFULL_RE**
  Value: `re.compile(r'(\?P<typ>Underfull|Overfull)\s+\(\?P<boxtype>...`  
- **IGNORE_WARNING_REGEXPS**
  Value: `[re.compile(r'LaTeX\s+Font\s+Warning:\s+.+\(Font\)\s*us...`  

### 16.3 Class  LatexWriter

#### 16.3.1 Methods

- `__init__(self, docindex, **kwargs)`
**write**(self, directory=None)

Write the API documentation for the entire project to the given directory.

**Parameters**

directory: The directory to which output should be written. If no directory is specified, output will be written to the current directory. If the directory does not exist, it will be created. *(type=string)*

**Return Value**

None

**Raises**

OSError If directory cannot be created,

OSError If any file cannot be created or written to.

**write_sty**(self, directory, stylesheet)

Copy the requested LaTeX stylesheet to the target directory. The stylesheet can be specified as a name (i.e., a key from the STYLESHEETS directory); a filename; or None for the default stylesheet. If any stylesheet *other* than the default stylesheet is selected, then the default stylesheet will be copied to 'epydoc-default.sty', which makes it possible to reference it via \RequirePackage.

**write**(self, write_func, directory, filename, *args)

**num_files**(self)

**Return Value**

The number of files that this LatexFormatter will generate. *(type=int)*

**mkdir**(self, directory)

If the given directory does not exist, then attempt to create it.

**Return Value**

None

---

**Main Doc File**

**write_topfile**(self, out)

**write_preamble**(self, out)

**Chapters**

**write_module**(self, out, doc)

**render_graph**(self, graph)
write_class(self, out, doc)

Module hierarchy trees

write_module_tree(self, out)
write_module_list(self, out, doc)
write_module_tree_item(self, out, doc, depth=0)

Helper function for write_module_tree and write_module_list.

Return Value

string

Base class trees

base_tree(self, doc, width=None, linespec=None)
_base_name(self, doc)
_find_tree_width(self, doc)
_base_tree_line(self, doc, width, linespec)

Class List

write_class_list(self, out, doc)
write_class_list_line(self, out, var_doc)

Details Lists

write_list(self, out, heading, doc, list_type, value_type, seclevel=1)
write_list_group(self, out, doc, name, var_docs, grouped_inh_vars)
write_inheritance_list(self, out, doc, listed_inh_vars)
_parens_if_func(self, var_doc)
Function Details

replace_par(self, out)

write_function(self, out, var_doc)

write_function_parameters(self, out, var_doc)

function_signature(self, var_doc, indent=6)

func_arg(self, name, default)
_arg_name(self, arg)

Variable Details

write_var(self, out, var_doc)

Property Details

write_property(self, out, var_doc)

Standard Fields

metadata(self, doc, indent=0)

metadata_field(self, doc, field, descrs, indent, arg='')

_descrlist(self, items, singular, plural=None, short=0, indent=0)

Docstring -> LaTeX Conversion

docstring_to_latex(self, docstring, where, indent=0, breakany=0)

Return a latex string that renders the given docstring. This string expects to start at the beginning of a line; and ends with a newline.

Helpers

write_header(self, out, where)
start_of(self, section_name, doc=None)

section(self, title, depth=0, ref=None)

sectionstar(self, title, depth, ref=None)

doc_kind(self, doc)

indexterm(self, doc, pos='only', indent=0)
    Return a latex string that marks the given term or section for inclusion in the index. This string ends with a newline.

get_latex_encoding(self)
    Return Value
    The LaTeX representation of the selected encoding. (type=str)

crossref(self, doc, indent=0)

_filter_deprecated(self, docs)

_is_deprecated(self, doc)

16.3.2 Class Variables

PREAMBLE
    Expects (options, epydoc_sty_package)
    Value: ['\documentclass{article}', '\usepackage[%s]{%s}']

SECTIONS
    Value: ['\part{%s}', '\chapter{%s}', '\section{%s}', '\subse...]

STAR_SECTIONS
    Value: ['\part*{%s}', '\chapter*{%s}', '\section*{%s}', '\su...]

Docstring -> LaTeX Conversion
16.3.3 Instance Variables

_graph_types
Graphs that we should include in our output.

_encoding
The Python representation of the encoding. Update latex_encodings in case of mismatch between it and the inputenc LaTeX package.

class_list
The list of ClassDocs for the documented classes.

class_set
The set of ClassDocs for the documented classes.

module_list
The list of ModuleDocs for the documented modules.

module_set
The set of ModuleDocs for the documented modules.

16.4 Class LatexWriter._LatexDocstringLinker

```
_classDiagram
  markup.DocstringLinker
  _LatexDocstringLinker
  markup.DocstringLinker --> _LatexDocstringLinker
```

16.4.1 Methods

**translate_indexterm**(self, indexterm)

Translate an index term to the appropriate output format. The output will typically include a crossreference anchor.

**Parameters**

- **indexterm**: The index term to translate.

**Return Value**

The translated index term. *(type=*string*)

**Overrides**: epydoc.markup.DocstringLinker.translate_indexterm *(inherited documentation)*

**translate_identifier_xref**(self, identifier, label=None)

Translate a crossreference link to a Python identifier to the appropriate output format. The output will typically include a reference or pointer to the crossreference target.

**Parameters**

- **identifier**: The name of the Python identifier that should be linked to.
- **label**: The label that should be used for the identifier, if it’s different from the name of the identifier. This should be expressed in the target markup language – e.g. for latex, ”_”s should be escaped.

**Return Value**

The translated crossreference link. *(type=*string*)

**Overrides**: epydoc.markup.DocstringLinker.translate_identifier_xref *(inherited documentation)*

Inherited from epydoc.markup.DocstringLinker *(Section 22.5, p. 202): url_for()*
### Module epydoc.docwriter.latex_sty

LaTeX stylesheets (*.sty) for epydoc’s LaTeX writer.

#### 17.1 Variables

<table>
<thead>
<tr>
<th>Name</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>NIST_DISCLAIMER</strong></td>
<td>% This style file is a derivative work, based on a pub...</td>
</tr>
<tr>
<td><strong>BASE</strong></td>
<td>% epydoc-base.sty% Authors: Jonathan Guyer &lt;guyer...</td>
</tr>
<tr>
<td><strong>BOXES</strong></td>
<td>% epydoc-boxes.sty% Authors: Jonathan Guyer &lt;guyer...</td>
</tr>
<tr>
<td><strong>SHADEN</strong></td>
<td>% epydoc-shaded.sty% Author: Edward Loper &lt;edlope...</td>
</tr>
<tr>
<td><strong>BLUE</strong></td>
<td>% epydoc-blue.sty% A relatively minimal customize...</td>
</tr>
<tr>
<td><strong>TEMPLATE</strong></td>
<td>% epydoc-template.sty% This is a starting point f...</td>
</tr>
<tr>
<td><strong>STYLESHEETS</strong></td>
<td>{'base': % epydoc-base.sty% Authors: Jonathan Guy...</td>
</tr>
</tbody>
</table>
18 Module epydoc.docwriter.plaintext

Plaintext output generation.

18.1 Class PlaintextWriter

18.1.1 Methods

```python
__init__(self, term)

write(self, api_doc, **options)

write_module(self, out, mod_doc)

baselist(self, class_doc)

write_class(self, out, class_doc, name=None, prefix='', verbose=True)

write_variable(self, out, var_doc, name=None, prefix='', verbose=True)

write_property(self, out, prop_doc, name=None, prefix='', verbose=True)

write_function(self, out, func_doc, name=None, prefix='', verbose=True)

write_signature(self, out, func_doc, name, prefix, verbose)

write_list(self, out, heading, doc, value_type=None, imported=False, inherited=False, prefix='', noindent=False, verbose=True)

_descr(self, descr, prefix)

bold(self, text)
```

Write a string in bold by overstriking.
title(self, text, indent)

section(self, text, indent='')

color(self, text, style)

18.1.2 Class Variables

STYLE

Value: {'class_name': 'bold cyan', 'h1': 'bold', 'h2': 'bold', ...}
Module epydoc.docwriter.xlink

A Docutils interpreted text role for cross-API reference support.

This module allows a Docutils document to refer to elements defined in external API documentation. It is possible to refer to many external API from the same document.

Each API documentation is assigned a new interpreted text role: using such interpreted text, an user can specify an object name inside an API documentation. The system will convert such text into an url and generate a reference to it. For example, if the API `db` is defined, being a database package, then a certain method may be referred as:

```
:db:`Connection.cursor()`
```

To define a new API, an *index file* must be provided. This file contains a mapping from the object name to the URL part required to resolve such object.

Index file

Each line in the the index file describes an object.

Each line contains the fully qualified name of the object and the URL at which the documentation is located. The fields are separated by a `<tab>` character.

The URL’s in the file are relative from the documentation root: the system can be configured to add a prefix in front of each returned URL.

Allowed names

When a name is used in an API text role, it is split over any *separator*. The separators defined are `.`, `::`, `->`. All the text from the first noise char (neither a separator nor alphanumeric or `.`) is discarded. The same algorithm is applied when the index file is read.

First the sequence of name parts is looked for in the provided index file. If no matching name is found, a partial match against the trailing part of the names in the index is performed. If no object is found, or if the trailing part of the name may refer to many objects, a warning is issued and no reference is created.

Configuration

This module provides the class `ApiLinkReader` a replacement for the Docutils standalone reader. Such reader specifies the settings required for the API canonical roles configuration. The same command line options are exposed by Epydoc.

The script `apirst2html.py` is a frontend for the `ApiLinkReader` reader.

API Linking Options:

```
--external-api=NAME
Define a new API document. A new interpreted text
role NAME will be added.
--external-api-file=NAME:FILENAME
Use records in FILENAME to resolve objects in the API
named NAME.
```
19.1 Functions

### API register

#### register_api(name, generator=None)

Register the API `name` into the `api_register`.

A registered API will be available to the markup as the interpreted text role `name`.

If a `generator` is not provided, register a `VoidUrlGenerator` instance: in this case no warning will be issued for missing names, but no URL will be generated and all the dotted names will simply be rendered as literals.

**Parameters**

- `name`: the name of the generator to be registered (type=`str`)
- `generator`: the object to register to translate names into URLs. (type=`UrlGenerator`)

#### set_api_file(name, file)

Set an URL generator populated with data from `file`.

Use `file` to populate a new `DocUrlGenerator` instance and register it as `name`.

**Parameters**

- `name`: the name of the generator to be registered (type=`str`)
- `file`: the file to parse populate the URL generator (type=`str` or `file`)

#### set_api_root(name, prefix)

Set the root for the URLs returned by a registered URL generator.

**Parameters**

- `name`: the name of the generator to be updated (type=`str`)
- `prefix`: the prefix for the generated URL’s (type=`str`)

**Raises**

- `IndexError` `name` is not a registered generator
**create_api_role***(name, problematic)***

Create and register a new role to create links for an API documentation.

Create a role called name, which will use the URL resolver registered as name in api_register to create a link for an object.

**Parameters**

*name:* name of the role to create. *(type=str)*

*problematic:* if True, the registered role will create problematic nodes in case of failed references. If False, a warning will be raised anyway, but the output will appear as an ordinary literal. *(type=bool)*

---

**Command line parsing**

**split_name***(value)***

Split an option in form NAME:VALUE and check if NAME exists.

---

### 19.2 Variables

**API register**

**api_register**

Mapping from the API name to the UrlGenerator to be used.

Use register_api() to add new generators to the register.

**Value:** {}

**_TARGET_RE**

**Value:** `re.compile(r'^-(.+?)\s*(?:URI:|L:)?([^-<>]+)>$')`

---

### 19.3 Class UrlGenerator

![Class Diagram]

**Known Subclasses:** epydoc.docwriter.xlink.DocUrlGenerator,
epydoc.docwriter.xlink.VoidUrlGenerator

Generate URL from an object name.
19.3.1 Methods

**get_url**(self, name)

Look for a name and return the matching URL documentation.

First look for a fully qualified name. If not found, try with partial name.

If no url exists for the given object, return None.

**Parameters**

- name: the name to look for (type=str)

**Return Value**

the URL that can be used to reach the name documentation. None if no such URL exists. (type=str)

**Raises**

- IndexError: no object found with name
- DocUrlGenerator.IndexAmbiguous: more than one object found with a non-fully qualified name; notice that this is an IndexError subclass

**getCanonical_name**(self, name)

Convert an object name into a canonical name.

the canonical name of an object is a tuple of strings containing its name fragments, splitted on any allowed separator (',', '::', '->').

Noise such parenthesis to indicate a function is discarded.

**Parameters**

- name: an object name, such as os.path.prefix() or lib::foo::bar (type=str)

**Return Value**

the fully qualified name such ('os', 'path', 'prefix') and ('lib', 'foo', 'bar') (type=tuple of str)

19.3.2 Class Variables

**_SEP_RE**

Value: re.compile(r'( *)([a-zA-Z0-9_]+)|(\.|::|->)|(.)')
19.4 Class UrlGenerator.IndexAmbiguous

The name looked for is ambiguous

19.4.1 Methods

Inherited from exceptions.IndexError: _init_, __new__

Inherited from exceptions.BaseException: _delattr_, __getattr__, __getitem__, __getslice__, __reduce__, __repr__, __setattr__, __setstate__, __str__

19.4.2 Properties

Inherited from exceptions.BaseException: args, message

19.5 Class VoidUrlGenerator

Don’t actually know any url, but don’t report any error.

Useful if an index file is not available, but a document linking to it is to be generated, and warnings are to be avoided.
Don’t report any object as missing, Don’t return any url anyway.

19.5.1 Methods

```python
get_url(self, name)
```

Look for a name and return the matching URL documentation.

First look for a fully qualified name. If not found, try with partial name.
If no url exists for the given object, return None.

**Parameters**

- `name`: the name to look for

**Return Value**

- the URL that can be used to reach the `name` documentation. `None` if no such URL exists.(type=str)

**Raises**

- `IndexError` no object found with `name`
- `DocUrlGenerator.IndexAmbiguous` more than one object found with a non-fully qualified name; notice that this is an `IndexError` subclass

**Overrides**: epydoc.docwriter.xlink.UrlGenerator.get_url (inherited documentation)

Inherited from epydoc.docwriter.xlink.UrlGenerator(Section 19.3, p. 176): getCanonicalName()

19.5.2 Class Variables

Inherited from epydoc.docwriter.xlink.UrlGenerator(Section 19.3, p. 176): _SEP_RE

19.6 Class DocUrlGenerator

Read a documentation index and generate URL’s for it.

19.6.1 Methods

```python
__init__(self)
```
**get_url**(self, name)

Look for a name and return the matching URL documentation.

First look for a fully qualified name. If not found, try with partial name.

If no url exists for the given object, return None.

**Parameters**

name: the name to look for

**Return Value**

the URL that can be used to reach the name documentation. None if no such URL exists. *(type=str)*

**Raises**

`IndexError` no object found with name

DocUrlGenerator.IndexAmbiguous more than one object found with a non-fully qualified name; notice that this is an IndexError subclass

**Overrides:** epydoc.docwriter.xlink.UrlGenerator.get_url *(inherited documentation)*

Inherited from epydoc.docwriter.xlink.UrlGenerator *(Section 19.3, p. 176): get_canonical_name()*

---

**content loading**

**clear**(self)

Clear the current class content.

**load_index**(self, f)

Read the content of an index file.

Populate the internal maps with the file content using `load_records()`.

**Parameters**

f: a file name or file-like object from which read the index. *(type=str or file)*

**_iter_tuples**(self, f)

Iterate on a file returning 2-tuples.

**load_records**(self, records)

Read a sequence of pairs name -> url and populate the internal maps.

**Parameters**

records: the sequence of pairs (name, url) to add to the maps. *(type=iterable)*

---

**19.6.2 Class Variables**

Inherited from epydoc.docwriter.xlink.UrlGenerator *(Section 19.3, p. 176): _SEP_RE*
## 19.6.3 Instance Variables

<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>_exact_matches</strong></td>
<td>A map from an object fully qualified name to its URL. Values are both the name as tuple of fragments and as read from the records (see <code>load_records()</code>), mostly to help <code>_partial_names</code> to perform lookup for unambiguous names.</td>
</tr>
<tr>
<td><strong>_partial_names</strong></td>
<td>A map from partial names to the fully qualified names they may refer. The keys are the possible left sub-tuples of fully qualified names, the values are list of strings as provided by the index. If the list for a given tuple contains a single item, the partial match is not ambiguous. In this case the string can be looked up in <code>_exact_matches</code>. If the name fragment is ambiguous, a warning may be issued to the user. The items can be used to provide an informative message to the user, to help him qualifying the name in a unambiguous manner.</td>
</tr>
<tr>
<td><strong>prefix</strong></td>
<td>Prefix portion for the URL’s returned by <code>get_url()</code>.</td>
</tr>
<tr>
<td><strong>_filename</strong></td>
<td>Not very important: only for logging.</td>
</tr>
</tbody>
</table>
19.7 Class ApiLinkReader

Known Subclasses: epydoc.markup.restructuredtext._EpydocReader

A Docutils standalone reader allowing external documentation links.
The reader configures the URL resolvers at the time `read()` is invoked the first time.

19.7.1 Methods

```python
__init__(self, *args, **kwargs)
```
Initialize the Reader instance.
Several instance attributes are defined with dummy initial values. Subclasses may use these attributes as they wish.

**Overrides**: docutils.readers.Reader.__init__ (inherited documentation)

```python
read(self, source, parser, settings)
```

**Overrides**: docutils.readers.Reader.read
**read_configuration**(*self*, *settings*, *problematic=**True***)

Read the configuration for the configured URL resolver.

Register a new role for each configured API.

**Parameters**

- **settings**: the settings structure containing the options to read.
- **problematic**: if True, the registered role will create problematic nodes in case of failed references. If False, a warning will be raised anyway, but the output will appear as an ordinary literal. *(type=bool)*

Inherited from docutils.readers.standalone.Reader: `get_transforms`

Inherited from docutils.readers.Reader: `new_document`, `parse`, `set_parser`

Inherited from docutils.Component: `supports`

### 19.7.2 Class Variables

**settings_spec**

The option parser configuration.

Value: (*'API Linking Options', None, ('Define a new API document...*

**_conf**

Value: **True**

Inherited from docutils.readers.standalone.Reader: `config_section`, `config_section_dependencies`, `document`, `supported`

Inherited from docutils.readers.Reader: `component_type`

Inherited from docutils.SettingsSpec: `relative_path_settings`, `settings_default_overrides`, `settings_defaults`

Inherited from docutils.TransformSpec: `default_transforms`, `unknown_reference_resolvers`
Module epydoc.gui

Graphical interface to epydoc. This interface might be useful for systems where it’s inconvenient to use the command-line interface (such as Windows). It supports many (but not all) of the features that are supported by the command-line interface. It also supports loading and saving of project files, which store a set of related modules, and the options that should be used to generate the documentation for those modules.

Usage:

```bash
epydocgui [OPTIONS] [FILE.prj | MODULES...]
```

**FILE.prj**
An epydoc GUI project file.

**MODULES...**
A list of Python modules to document.

**-V, --version**
Print the version of epydoc.

**-h, -?, --help, --usage**
Display this usage message

**--debug**
Do not suppress error messages

**To Do:** Use ini-style project files, rather than pickles (using the same format as the CLI).

### 20.1 Functions

**document**(options, cancel, done)
Create the documentation for modules, using the options specified by options. document is designed to be started in its own thread by EpydocGUI._go.

**Parameters**

- **options**: The options to use for generating documentation. This includes keyword options that can be given to docwriter.html.HTMLWriter, as well as the option target, which controls where the output is written to. (type=dictionary)

**version()**
Display the version information, and exit.

**Return Value**
None

**usage()**

**_error(s)**

**gui()**
## 20.2 Variables

<table>
<thead>
<tr>
<th>Variable</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>DEBUG</td>
<td>0</td>
</tr>
<tr>
<td>BG_COLOR</td>
<td>'#e0e0e0'</td>
</tr>
<tr>
<td>ACTIVEBG_COLOR</td>
<td>'#e0e0e0'</td>
</tr>
<tr>
<td>TEXT_COLOR</td>
<td>'black'</td>
</tr>
<tr>
<td>ENTRYSELECT_COLOR</td>
<td>'#e0e0e0'</td>
</tr>
<tr>
<td>SELECT_COLOR</td>
<td>'#208070'</td>
</tr>
<tr>
<td>MESSAGE_COLOR</td>
<td>'#000060'</td>
</tr>
<tr>
<td>ERROR_COLOR</td>
<td>'#600000'</td>
</tr>
<tr>
<td>GUIERROR_COLOR</td>
<td>'#600000'</td>
</tr>
<tr>
<td>WARNING_COLOR</td>
<td>'#604000'</td>
</tr>
<tr>
<td>HEADER_COLOR</td>
<td>'#000000'</td>
</tr>
</tbody>
</table>
COLOR_CONFIG
Value: {'background': '#e0e0e0', 'foreground': 'black', 'highlight...' 

ENTRY_CONFIG
Value: {'background': '#e0e0e0', 'foreground': 'black', 'highlight...' 

SB_CONFIG
Value: {'activebackground': '#e0e0e0', 'background': '#e0e0e0', ... 

LISTBOX_CONFIG
Value: {'background': '#e0e0e0', 'foreground': 'black', 'highlight...' 

BUTTON_CONFIG
Value: {'activebackground': '#e0e0e0', 'activeforeground': 'black', ... 

CBUTTON_CONFIG
Value: {'activebackground': '#e0e0e0', 'activeforeground': 'black', ... 

SHOWMSG_CONFIG
Value: {'activebackground': '#e0e0e0', 'activeforeground': 'black', ... 

SHOWWRN_CONFIG
Value: {'activebackground': '#e0e0e0', 'activeforeground': 'black', ... 

SHOWERR_CONFIG
Value: {'activebackground': '#e0e0e0', 'activeforeground': 'black', ... 

PROGRESS_HEIGHT
Value: 16 

PROGRESS_WIDTH
Value: 200
<table>
<thead>
<tr>
<th>Variable</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>PROGRESS_BG</td>
<td>'#305060'</td>
</tr>
<tr>
<td>PROGRESS_COLOR1</td>
<td>'#30c070'</td>
</tr>
<tr>
<td>PROGRESS_COLOR2</td>
<td>'#60ff0a'</td>
</tr>
<tr>
<td>PROGRESS_COLOR3</td>
<td>'#106030'</td>
</tr>
<tr>
<td>DX</td>
<td>1</td>
</tr>
<tr>
<td>DY</td>
<td>1</td>
</tr>
<tr>
<td>DH</td>
<td>1</td>
</tr>
<tr>
<td>DW</td>
<td>3</td>
</tr>
<tr>
<td>IMPORT_PROGRESS</td>
<td>0.1</td>
</tr>
<tr>
<td>BUILD_PROGRESS</td>
<td>0.2</td>
</tr>
<tr>
<td>WRITE_PROGRESS</td>
<td>0.7</td>
</tr>
</tbody>
</table>
20.3 Class GUILogger

```python
log.Logger
```

```python
GUILogger
```

20.3.1 Methods

```python
__init__(self, progress, cancel)
```

```python
clear(self)
```

```python
log(self, level, message)
```

Display a message.

**Parameters**

message: The message string to display. message may contain newlines, but does not need to end in a newline.

level: An integer value indicating the severity of the message.

**Overrides:** epydoc.log.Logger.log *(inherited documentation)*
class GUILogger

start_block(self, header)

Start a new message block. Any calls to info(), warning(), or error() that occur between a call to start_block and a corresponding call to end_block will be grouped together, and displayed with a common header. start_block can be called multiple times (to form nested blocks), but every call to start_block must be balanced by a call to end_block.

Overrides: epydoc.log.Logger.start_block (inherited documentation)

end_block(self)

End a warning block. See start_block for details.

Overrides: epydoc.log.Logger.end_block (inherited documentation)

start_progress(self, header=None)

Begin displaying progress for a new task. header is a description of the task for which progress is being reported. Each call to start_progress must be followed by a call to end_progress (with no intervening calls to start_progress).

Overrides: epydoc.log.Logger.start_progress (inherited documentation)

end_progress(self)

Finish off the display of progress for the current task. See start_progress for more information.

Overrides: epydoc.log.Logger.end_progress (inherited documentation)

progress(self, percent, message=''

Update the progress display.

Parameters

percent: A float from 0.0 to 1.0, indicating how much progress has been made.

message: A message indicating the most recent action that contributed towards that progress.

Overrides: epydoc.log.Logger.progress (inherited documentation)

read(self)

Inherited from epydoc.log.Logger (Section 21.3, p. 194): close()

20.3.2 Class Variables

_STAGES

Value: [40, 7, 1, 3, 1, 30, 1, 2, 100]
20.4 Class EpydocGUI

A graphical user interface to epydoc.

20.4.1 Methods

```python
_init_(self)
_init_menu_bar(self)
_init_module_list(self, mainframe)
_init_progress_bar(self, mainframe)
_init_messages(self, msgsframe, ctrlframe)
_update_msg_tags(self, *e)
_init_options(self, optsframe, ctrlframe)
_init_bindings(self)
_options_toggle(self, *e)
_messages_toggle(self, *e)
_configure(self, event)
_delete_module(self, *e)
_entry_module(self, *e)
_browse_module(self, *e)
_browse_css(self, *e)
_browse_help(self, *e)
```

_browse_out(self, *e)

destroy(self, *e)

add_module(self, name, check=0)

mainloop(self, *args, **kwargs)

_getopts(self)

_go(self, *e)

_update_messages(self)

_update(self, dt, id)

_new(self, *e)

_open(self, *e)

open(self, prjfile)

_save(self, *e)

_saveas(self, *e)
Module epydoc.log

Functions used to report messages and progress updates to the user. These functions are delegated to zero or more registered Logger objects, which are responsible for actually presenting the information to the user. Different interfaces are free to create and register their own Loggers, allowing them to present this information in the manner that is best suited to each interface.

**Note:** I considered using the standard logging package to provide this functionality. However, I found that it would be too difficult to get that package to provide the behavior I want (esp. with respect to progress displays; but also with respect to message blocks).

### 21.1 Functions

**register_logger**(logger)

Register a logger. Each call to one of the logging functions defined by this module will be delegated to each registered logger.

**remove_logger**(logger, close_logger=True)

**fatal**(messages)

Display the given fatal message.

**error**(messages)

Display the given error message.

**warning**(messages)

Display the given warning message.

**docstring_warning**(messages)

Display the given docstring warning message.

**info**(messages)

Display the given informational message.

**debug**(messages)

Display the given debugging message.

**start_block**(header)

Start a new message block. Any calls to info(), warning(), or error() that occur between a call to start_block and a corresponding call to end_block will be grouped together, and displayed with a common header. start_block can be called multiple times (to form nested blocks), but every call to start_block must be balanced by a call to end_block.
end_block()

End a warning block. See start_block for details.

start_progress(header=None)

Begin displaying progress for a new task. header is a description of the task for which progress is being reported. Each call to start_progress must be followed by a call to end_progress (with no intervening calls to start_progress).

end_progress()

Finish off the display of progress for the current task. See start_progress for more information.

progress(percent, message='')

Update the progress display.

Parameters

percent: A float from 0.0 to 1.0, indicating how much progress has been made.

message: A message indicating the most recent action that contributed towards that progress.

close()

21.2 Variables

DOCSTRING_WARNING

Value: 25

_loggers

The list of registered logging functions.

Value: []

Message Severity Levels

DEBUG

Value: 10

INFO

Value: 20
**WARNING**

Value: 30

**ERROR**

Value: 40

**FATAL**

Value: 40

## 21.3 Class Logger

An abstract base class that defines the interface for *loggers*, which are used by epydoc to report information back to the user. Loggers are responsible for tracking two types of information:

- Messages, such as warnings and errors.
- Progress on the current task.

This abstract class allows the command-line interface and the graphical interface to each present this information to the user in the way that’s most natural for each interface. To set up a logger, create a subclass of *Logger* that overrides all methods, and register it using `register_logger`.

### 21.3.1 Methods

**log**(self, level, message)

Display a message.

**Parameters**

- `message`: The message string to display. `message` may contain newlines, but does not need to end in a newline.
- `level`: An integer value indicating the severity of the message.
close(self)

Perform any tasks needed to close this logger. This should be safe to call multiple times.

start_block(self, header)

Start a new message block. Any calls to info(), warning(), or error() that occur between a call to start_block and a corresponding call to end_block will be grouped together, and displayed with a common header. start_block can be called multiple times (to form nested blocks), but every call to start_block must be balanced by a call to end_block.

end_block(self)

End a warning block. See start_block for details.

start_progress(self, header=None)

Begin displaying progress for a new task. header is a description of the task for which progress is being reported. Each call to start_progress must be followed by a call to end_progress (with no intervening calls to start_progress).

end_progress(self)

Finish off the display of progress for the current task. See start_progress for more information.

progress(self, percent, message='')

Update the progress display.

Parameters
percent: A float from 0.0 to 1.0, indicating how much progress has been made.
message: A message indicating the most recent action that contributed towards that progress.

21.4 Class SimpleLogger

21.4.1 Methods

__init__(self, threshold=30)
**log**(self, level, message)

Display a message.

**Parameters**

- **message**: The message string to display. `message` may contain newlines, but does not need to end in a newline.
- **level**: An integer value indicating the severity of the message.

**Overrides**: `epydoc.log.Logger.log (inherited documentation)`

Inherited from `epydoc.log.Logger (Section 21.3, p. 194)`: close(), end_block(), end_progress(), progress(), start_block(), start_progress()
22 Package epydoc.markup

Markup language support for docstrings. Each submodule defines a parser for a single markup language. These parsers convert an object’s docstring to a ParsedDocstring, a standard intermediate representation that can be used to generate output. ParsedDocstrings support the following operations:

- output generation (to_plaintext(), to_html(), and to_latex()).
- Summarization (summary()).
- Field extraction (split_fields()).
- Index term extraction (index_terms()).

The parse() function provides a single interface to the epydoc.markup package: it takes a docstring and the name of a markup language; delegates to the appropriate parser; and returns the parsed docstring (along with any errors or warnings that were generated).

The ParsedDocstring output generation methods (to_format()) use a DocstringLinker to link the docstring output with the rest of the documentation that epydoc generates. DocstringLinkers are currently responsible for translating two kinds of crossreference:

- index terms (translate_indexterm()).
- identifier crossreferences (translate_identifier_xref()).

A parsed docstring’s fields can be extracted using the ParsedDocstring.split_fields() method. This method divides a docstring into its main body and a list of Fields, each of which encodes a single field. The field’s bodies are encoded as ParsedDocstrings.

Markup errors are represented using ParseErrors. These exception classes record information about the cause, location, and severity of each error.

22.1 Functions

```python
parse(docstring, markup='plaintext', errors=None, **options)
```

Parse the given docstring, and use it to construct a ParsedDocstring. If any fatal ParseErrors are encountered while parsing the docstring, then the docstring will be rendered as plaintext, instead.

**Parameters**

- docstring: The docstring to encode. *(type=string)*
- markup: The name of the markup language that is used by the docstring. If the markup language is not supported, then the docstring will be treated as plaintext. The markup name is case-insensitive. *(type=string)*
- errors: A list where any errors generated during parsing will be stored. If no list is specified, then fatal errors will generate exceptions, and non-fatal errors will be ignored. *(type=list of ParseError)*

**Return Value**

A ParsedDocstring that encodes the contents of docstring. *(type=ParsedDocstring)*

**Raises**

- ParseError If errors is None and an error is encountered while parsing.
**register_markup_language(name, parse_function)**

Register a new markup language named `name`, which can be parsed by the function `parse_function`.

**Parameters**

- **name**: The name of the markup language. `name` should be a simple identifier, such as 'epytext' or 'restructuredtext'. Markup language names are case insensitive.
- **parse_function**: A function which can be used to parse the markup language, and returns a `ParsedDocstring`. It should have the following signature:

  ```python
  >>> def parse(s, errors):
  ...     'returns a ParsedDocstring'
  ```

  Where:

  - `s` is the string to parse. (`s` will be a unicode string.)
  - `errors` is a list; any errors that are generated during docstring parsing should be appended to this list (as `ParseError` objects).

**Utility Functions**

**parse_type_of(obj)**

**Parameters**

- **obj**: The object whose type should be returned as DOM document. (`type=any`)

**Return Value**

A `ParsedDocstring` that encodes the type of the given object. (`type=ParsedDocstring`)

### 22.2 Variables

**_markup_language_registry**

| Value: | {'restructuredtext': 'epydoc.markup.restructuredtext', 'e... |

**MARKUP_LANGUAGES_USED**

| Value: | set(['epytext', 'plaintext', u'restructuredtext']) |
22.3 Class ParsedDocstring

A standard intermediate representation for parsed docstrings that can be used to generate output. Parsed docstrings are produced by markup parsers (such as `epytext.parse` or `javadoc.parse`). ParsedDocstrings support several kinds of operation:

- output generation (`to_plaintext()`, `to_html()`, and `to_latex()`).
- Summarization (`summary()`).
- Field extraction (`split_fields()`).
- Index term extraction (`index_terms()`).

The output generation methods (`to_format()`) use a `DocstringLinker` to link the docstring output with the rest of the documentation that epydoc generates.

Subclassing

The only method that a subclass is required to implement is `to_plaintext()`; but it is often useful to override the other methods. The default behavior of each method is described below:

- **to_format**: Calls `to_plaintext`, and uses the string it returns to generate verbatim output.
- **summary**: Returns `self` (i.e., the entire docstring).
- **split_fields**: Returns `()` (i.e., extracts no fields).
- **index_terms**: Returns `[]` (i.e., extracts no index terms).

If and when epydoc adds more output formats, new `to_format` methods will be added to this base class; but they will always be given a default implementation.
22.3.1 Methods

**split_fields** *(self, errors=None)*

Split this docstring into its body and its fields.

**Parameters**

- *errors*: A list where any errors generated during splitting will be stored. If no list is specified, then errors will be ignored. *(type=list of ParseError)*

**Return Value**

A tuple \((body, fields)\), where \(body\) is the main body of this docstring, and \(fields\) is a list of its fields. If the resulting body is empty, return \(None\) for the body. *(type=(ParsedDocstring, list of Field))*

**summary** *(self)*

**Return Value**

A pair consisting of a short summary of this docstring and a boolean value indicating whether there is further documentation in addition to the summary. Typically, the summary consists of the first sentence of the docstring. *(type=(ParsedDocstring, bool))*

**concatenate** *(self, other)*

**Return Value**

A new parsed docstring containing the concatenation of this docstring and \(other\).

**Raises**

*ValueError* If the two parsed docstrings are incompatible.

**__add__** *(self, other)*

**to_html** *(self, docstring_linker, **options)*

Translate this docstring to HTML.

**Parameters**

- *docstring_linker*: An HTML translator for crossreference links into and out of the docstring. *(type=DocstringLinker)*

- *options*: Any extra options for the output. Unknown options are ignored.

**Return Value**

An HTML fragment that encodes this docstring. *(type=string)*
to\_latex\(\)(\texttt{self}, \texttt{docstring\_linker}, **\texttt{options})\)

Translate this docstring to LaTeX.

**Parameters**
- \texttt{docstring\_linker}: A LaTeX translator for crossreference links into and out of the docstring. \((\texttt{type=DocstringLinker})\)
- \texttt{options}: Any extra options for the output. Unknown options are ignored.

**Return Value**
A LaTeX fragment that encodes this docstring, \((\texttt{type=string})\)

\hspace{1em}

\texttt{to\_plaintext}(\texttt{self}, \texttt{docstring\_linker}, **\texttt{options})\)

Translate this docstring to plaintext.

**Parameters**
- \texttt{docstring\_linker}: A plaintext translator for crossreference links into and out of the docstring. \((\texttt{type=DocstringLinker})\)
- \texttt{options}: Any extra options for the output. Unknown options are ignored.

**Return Value**
A plaintext fragment that encodes this docstring, \((\texttt{type=string})\)

\hspace{1em}

\texttt{index\_terms}(\texttt{self})

**Return Value**
The list of index terms that are defined in this docstring. Each of these items will be added to the index page of the documentation, \((\texttt{type=list\ of\ ParsedDocstring})\)

\section{22.4 Class Field}

The contents of a docstring’s field. Docstring fields are used to describe specific aspects of an object, such as a parameter of a function or the author of a module. Each field consists of a tag, an optional argument, and a body:

- The tag specifies the type of information that the field encodes.
- The argument specifies the object that the field describes. The argument may be \texttt{None} or a \texttt{string}.
- The body contains the field’s information.

Tags are automatically downcased and stripped; and arguments are automatically stripped.

\subsection{22.4.1 Methods}

\hspace{1em}

\texttt{\_init\_}(\texttt{self}, \texttt{tag}, \texttt{arg}, \texttt{body})

\hspace{1em}

\texttt{tag}(\texttt{self})

**Return Value**
This field’s \texttt{tag}, \((\texttt{type=string})\)
arg(self)  
Return Value  
This field’s argument, or None if this field has no argument. (type=string or None)

body(self)  
Return Value  
This field’s body. (type=ParsedDocstring)

__repr__(self)

22.5 Class DocstringLinker

A translator for crossreference links into and out of a ParsedDocstring. DocstringLinker is used by ParsedDocstring to convert these crossreference links into appropriate output formats. For example, DocstringLinker.to_html expects a DocstringLinker that converts crossreference links to HTML.

22.5.1 Methods

translate_identifier_xref(self, identifier, label=None)  
Translate a crossreference link to a Python identifier to the appropriate output format. The output will typically include a reference or pointer to the crossreference target.

Parameters  
identifier: The name of the Python identifier that should be linked to. (type=string)  
label: The label that should be used for the identifier, if it’s different from the name of the identifier. This should be expressed in the target markup language – e.g. for latex, ”_”s should be escaped. (type=string or None)

Return Value  
The translated crossreference link. (type=string)
translate_indexterm(self, indexterm)

Translate an index term to the appropriate output format. The output will typically include a crossreference anchor.

Parameters

indexterm: The index term to translate. (type=ParsedDocstring)

Return Value

The translated index term. (type=string)

url_for(self, identifier)

Given an identifier, return a URL pointing at that identifier. This is used to create hyperlinks in dotgraphs. This method is *optional* — i.e., it may raise NotImplementedError.

22.6 Class ConcatenatedDocstring

22.6.1 Methods

__init__(self, *parsed_docstrings)

split_fields(self, errors=None)

summary(self)

to_html(self, docstring_linker, **options)

to_latex(self, docstring_linker, **options)

to_plaintext(self, docstring_linker, **options)

index_terms(self)
22.7 Class ParseError


The base class for errors generated while parsing docstrings.

22.7.1 Methods

```python
__init__(self, descr, linenum=None, is_fatal=1)
```

x.__init__(...) initializes x; see x.__class__.__doc__ for signature

Parameters

- descr: A description of the error. *(type=string)*
- linenum: The line on which the error occurred within the docstring. The linenum of the first line is 0. *(type=int)*
- is_fatal: True if this is a fatal error. *(type=boolean)*

Overrides: exceptions.BaseException.__init__

```python
is_fatal(self)
```

Return Value

true if this is a fatal error. If an error is fatal, then epydoc should ignore the output of the parser, and parse the docstring as plaintext. *(type=boolean)*

```python
linenum(self)
```

Return Value

The line number on which the error occurred (including any offset). If the line number is unknown, then return *None.* *(type=int or None)*
**set_linenum_offset** *(self, offset)*

Set the line number offset for this error. This offset is the line number where the docstring begins. This offset is added to _linenum when displaying the line number of the error.

**Parameters**
- **offset**: The new line number offset. *(type=int)*

**Return Value**
- *None*

**descr**(self)

**__str__**(self)

Return a string representation of this ParseError. This multi-line string contains a description of the error, and specifies where it occurred.

**Return Value**
- the informal representation of this ParseError.*(type=string)*

**Overrides**: exceptions.BaseException.__str__

**__repr__**(self)

Return the formal representation of this ParseError. ParseErrors have formal representations of the form:

<ParseError on line 12>

**Return Value**
- the formal representation of this ParseError.*(type=string)*

**Overrides**: exceptions.BaseException.__repr__

**__cmp__**(self, other)

Compare two ParseErrors, based on their line number.

- Return -1 if self.linenum<other.linenum
- Return +1 if self.linenum>other.linenum
- Return 0 if self.linenum==other.linenum.

The return value is undefined if other is not a ParseError.

**Return Value**
- *int*

Inherited from exceptions.Exception: __new__()
22.7.2 Properties

Inherited from exceptions.BaseException: args, message

22.7.3 Instance Variables

_descr
A description of the error.
Type: string

_fatal
True if this is a fatal error.
Type: boolean

_linenum
The line on which the error occurred within the docstring. The linenum of the first line is 0.
Type: int

_offset
The line number where the docstring begins. This offset is added to _linenum when displaying the line number of the error. Default value: 1.
Type: int
Module epydoc.markup.doctest

Syntax highlighting for doctest blocks. This module defines two functions, `doctest_to_html()` and `doctest_to_latex()`, which can be used to perform syntax highlighting on doctest blocks. It also defines the more general `colorize_doctest()`, which could be used to do syntax highlighting on doctest blocks with other output formats. (Both `doctest_to_html()` and `doctest_to_latex()` are defined using `colorize_doctest()`.)

23.1 Functions

**doctest_to_html(s)**

Perform syntax highlighting on the given doctest string, and return the resulting HTML code. This code consists of a `<pre>` block with class=py-doctest. Syntax highlighting is performed using the following CSS classes:

- **py-prompt** – the Python PS1 prompt (`>>>`)
- **py-more** – the Python PS2 prompt (`...`)
- **py-keyword** – a Python keyword (for, if, etc.)
- **py-builtin** – a Python built-in name (abs, dir, etc.)
- **py-string** – a string literal
- **py-comment** – a comment
- **py-except** – an exception traceback (up to the next `>>>`)
- **py-output** – the output from a doctest block.
- **py-defname** – the name of a function or class defined by a `def` or `class` statement.

**doctest_to_latex(s)**

Perform syntax highlighting on the given doctest string, and return the resulting LaTeX code. This code consists of an `alltt` environment. Syntax highlighting is performed using the following new LaTeX commands, which must be defined externally:

- \pysrcprompt – the Python PS1 prompt (`>>>`)
- \pysrcmore – the Python PS2 prompt (`...`)
- \pysrckeyword – a Python keyword (for, if, etc.)
- \pysrcbuiltin – a Python built-in name (abs, dir, etc.)
- \pysrcstring – a string literal
- \pysrccomment – a comment
- \pysrcexcept – an exception traceback (up to the next `>>>`)
- \pysrcoutput – the output from a doctest block.
- \pysrcdefname – the name of a function or class defined by a `def` or `class` statement.
23.2 Class DoctestColorizer

An abstract base class for performing syntax highlighting on doctest blocks and other bits of Python code. Subclasses should provide definitions for:

- The `markup()` method, which takes a substring and a tag, and returns a colorized version of the substring.
- The `PREFIX` and `SUFFIX` variables, which will be added to the beginning and end of the strings returned by `colorize_codeblock` and `colorize_doctest`.

23.2.1 Methods

```python
colorize_inline(self, s)
```
Colorize a string containing Python code. Do not add the `PREFIX` and `SUFFIX` strings to the returned value. This method is intended for generating syntax-highlighted strings that are appropriate for inclusion as inline expressions.

```python
colorize_codeblock(self, s)
```
Colorize a string containing only Python code. This method differs from `colorize_doctest` in that it will not search for doctest prompts when deciding how to colorize the string.

```python
colorize_doctest(self, s, strip_directives=False)
```
Colorize a string containing one or more doctest examples.

```python
subfunc(self, match)
```
Class DoctestColorizer

Module epydoc.markup.doctest

### markup(self, s, tag)
Apply syntax highlighting to a single substring from a doctest block. `s` is the substring, and `tag` is the tag that should be applied to the substring. `tag` will be one of the following strings:

- **prompt** – the Python PS1 prompt (`>>>`)
- **more** – the Python PS2 prompt (...
- **keyword** – a Python keyword (for, if, etc.)
- **builtin** – a Python builtin name (abs, dir, etc.)
- **string** – a string literal
- **comment** – a comment
- **except** – an exception traceback (up to the next `>>>`)
- **output** – the output from a doctest block.
- **defname** – the name of a function or class defined by a `def` or `class` statement.
- **other** – anything else (does *not* include output.)

### 23.2.2 Class Variables

**PREFIX**
A string that is added to the beginning of the strings returned by colorize_codeblock and colorize_doctest. Typically, this string begins a preformatted area.

Value: None

**SUFFIX**
A string that is added to the end of the strings returned by colorize_codeblock and colorize_doctest. Typically, this string ends a preformatted area.

Value: None

**NEWLINE**
The string used to divide lines

Value: `\n`

**KEYWORDS**
A list of the names of all Python keywords. (`as` is included even though it is technically not a keyword.)


**_BUILTINS**
A list of all Python builtns.

Value: `[‘clear’, ‘copy’, ‘fromkeys’, ‘get’, ‘has_key’, ‘items’, ...`
Class DoctestColorizer

Module epydoc.markup.doctest

KEYWORD GRP
A regexp group that matches keywords.
Value: ’\\band\\b|\\bdel\\b|\\bfor\\b|\\bis\\b|\\braiseassert\\b...

BUILTIN GRP
A regexp group that matches Python builtins.
Value: ’(?<!\\.)(?:\\bclear\\b|\\bcopy\\b|\\bfromkeys\\b|\\bget\...

STRING GRP
A regexp group that matches Python strings.
Value: ’("""("""|.*?((?!").)"""))|("("|.*?((?!").)"))|(\’\’\’(\’...

COMMENT GRP
A regexp group that matches Python comments.
Value: ’(#.*?$)’

PROMPT1 GRP
A regexp group that matches Python ”>>>” prompts.
Value: ’^[ \\t]*>>>(?:[ \\t]|$)’

PROMPT2 GRP
A regexp group that matches Python ”...” prompts.
Value: ’^[ \\t]*\\.\\.\\.(?:[ \\t]|$)’

DEFINE GRP
A regexp group that matches function and class definitions.
Value: ’\\b(?:def|class)[ \\t]+\\w+’

PROMPT RE
A regexp that matches Python prompts
Value: re.compile(r’(?ms)(^[ \t]*>>>(?:[ \t]|$)|[ \t]*\.\.\.(?:[...

210


23.3 Class XMLDoctestColorizer

A subclass of DoctestColorizer that generates XML-like output. This class is mainly intended to be used for testing purposes.
23.3.1 Methods

**markup**(*self, s, tag*)

Apply syntax highlighting to a single substring from a doctest block. *s* is the substring, and *tag* is the tag that should be applied to the substring. *tag* will be one of the following strings:

- **prompt** – the Python PS1 prompt (```>>>``)
- **more** – the Python PS2 prompt (```...```
- **keyword** – a Python keyword (for, if, etc.)
- **builtin** – a Python builtin name (abs, dir, etc.)
- **string** – a string literal
- **comment** – a comment
- **except** – an exception traceback (up to the next ```>>>``)
- **output** – the output from a doctest block.
- **defname** – the name of a function or class defined by a `def` or `class` statement.
- **other** – anything else (does *not* include output.)

**Overrides:** epydoc.markup.doctest.DoctestColorizer.markup *(inherited documentation)*

Inherited from epydoc.markup.doctest.DoctestColorizer *(Section 23.2, p. 208):* `colorize_codeblock()`, `colorize_doctest()`, `colorize_inline()`, `subfunc()`

23.3.2 Class Variables

**PREFIX**

A string that is added to the beginning of the strings returned by `colorize_codeblock` and `colorize_doctest`. Typically, this string begins a preformatted area.

Value: `'<colorized>
`

**SUFFIX**

A string that is added to the end of the strings returned by `colorize_codeblock` and `colorize_doctest`. Typically, this string ends a preformatted area.

Value: `'</colorized>
`

23.4 Class HTMLDoctestColorizer

A subclass of DoctestColorizer that generates HTML output.

23.4.1 Methods

**markup**(self, s, tag)

Apply syntax highlighting to a single substring from a doctest block. s is the substring, and tag is the tag that should be applied to the substring. tag will be one of the following strings:

- *prompt* – the Python PS1 prompt (>>>)
- *more* – the Python PS2 prompt (...)  
- *keyword* – a Python keyword (for, if, etc.)  
- *builtin* – a Python builtin name (abs, dir, etc.)  
- *string* – a string literal  
- *comment* – a comment  
- *except* – an exception traceback (up to the next >>>)  
- *output* – the output from a doctest block.  
- *defname* – the name of a function or class defined by a def or class statement.  
- *other* – anything else (does *not* include output.)

Overrides: epydoc.markup.doctest.DoctestColorizer.markup (inherited documentation)

Inherited from epydoc.markup.doctest.DoctestColorizer (Section 23.2, p. 208): colorize_codeblock(), colorize_doctest(), colorize_inline(), subfunc()

23.4.2 Class Variables

**PREFIX**

A string that is added to the beginning of the strings returned by colorize_codeblock and colorize_doctest. Typically, this string begins a preformatted area.

Value: `'<pre class="py-doctest">\n'`

**SUFFIX**

A string that is added to the end of the strings returned by colorize_codeblock and colorize_doctest. Typically, this string ends a preformatted area.

Value: `'</pre>\n'`
23.5 Class LaTeXDoctestColorizer

LaTeXDoctestColorizer

A subclass of DoctestColorizer that generates LaTeX output.

23.5.1 Methods

**markup(self, s, tag)**

Apply syntax highlighting to a single substring from a doctest block. s is the substring, and tag is the tag that should be applied to the substring. tag will be one of the following strings:

- **prompt** – the Python PS1 prompt (```)```.
- **more** – the Python PS2 prompt (```(...```).
- **keyword** – a Python keyword (for, if, etc.)
- **builtin** – a Python built-in name (abs, dir, etc.)
- **string** – a string literal
- **comment** – a comment
- **except** – an exception traceback (up to the next ```)
- **output** – the output from a doctest block.
- **defname** – the name of a function or class defined by a `def` or `class` statement.
- **other** – anything else (does *not* include output.)

 Overrides: epydoc.markup.doctest.DoctestColorizer.markup (inherited documentation)

Inherited from epydoc.markup.doctest.DoctestColorizer(Section 23.2, p. 208): colorize_codeblock(), colorize_doctest(), colorize_inline(), subfunc()
### SUFFIX

A string that is added to the end of the strings returned by `colorize_codeblock` and `colorize_doctest`. Typically, this string ends a preformatted area.

**Value:** `'\end{alltt}\n'`

### NEWLINE

The string used to divide lines

**Value:** `'\\'`

Inherited from `epydoc.markup.doctest.DoctestColorizer` *(Section 23.2, p. 208)*:
Module epydoc.markup.epytext

Parser for epytext strings. Epytext is a lightweight markup whose primary intended application is Python documentation strings. This parser converts Epytext strings to a simple DOM-like representation (encoded as a tree of `Element` objects and strings). Epytext strings can contain the following structural blocks:

- **epytext**: The top-level element of the DOM tree.
- **para**: A paragraph of text. Paragraphs contain no newlines, and all spaces are soft.
- **section**: A section or subsection.
- **field**: A tagged field. These fields provide information about specific aspects of a Python object, such as the description of a function’s parameter, or the author of a module.
- **literalblock**: A block of literal text. This text should be displayed as it would be displayed in plaintext. The parser removes the appropriate amount of leading whitespace from each line in the literal block.
- **doctestblock**: A block containing sample python code, formatted according to the specifications of the `doctest` module.
- **ulist**: An unordered list.
- **olist**: An ordered list.
- **li**: A list item. This tag is used both for unordered list items and for ordered list items.

Additionally, the following inline regions may be used within `para` blocks:

- **code**: Source code and identifiers.
- **math**: Mathematical expressions.
- **index**: A term which should be included in an index, if one is generated.
- **italic**: Italicized text.
- **bold**: Bold-faced text.
- **uri**: A Universal Resource Indicator (URI) or Universal Resource Locator (URL)
- **link**: A Python identifier which should be hyperlinked to the named object’s documentation, when possible.

The returned DOM tree will conform to the the following Document Type Description:

```xml
<!ENTITY % colorized '(code | math | index | italic |
  bold | uri | link | symbol)*'>

<!ELEMENT epytext ((para | literalblock | doctestblock |
section | ulist | olist)*, fieldlist?)>

<!ELEMENT para (#PCDATA | %colorized;)*>

<!ELEMENT section (para | listblock | doctestblock |
section | ulist | olist)+>

<!ELEMENT fieldlist (field+)>  
<!ELEMENT field (tag, arg?, (para | listblock | doctestblock)
  ulist | olist)+)>

<!ELEMENT tag (#PCDATA)> 
<!ELEMENT arg (#PCDATA)> 
```
24.1 Functions

**parse** *(str, errors=None)*

Return a DOM tree encoding the contents of an epytext string. Any errors generated during parsing will be stored in `errors`.

**Parameters**
- **str**: The epytext string to parse. *(type=string)*
- **errors**: A list where any errors generated during parsing will be stored. If no list is specified, then fatal errors will generate exceptions, and non-fatal errors will be ignored. *(type=list of ParseError)*

**Return Value**
- a DOM tree encoding the contents of an epytext string. *(type=Element)*

**Raises**
- `ParseError` If `errors` is `None` and an error is encountered while parsing.

**_raise_graphs**(tree, parent)

**_pop_completed_blocks**(token, stack, indent_stack)

Pop any completed blocks off the stack. This includes any blocks that we have dedented past, as well as any list item blocks that we’ve dedented to. The top element on the stack should only be a list if we’re about to start a new list item (i.e., if the next token is a bullet).
_add_para(doc, para_token, stack, indent_stack, errors)

Colorize the given paragraph, and add it to the DOM tree.

_add_section(doc, heading_token, stack, indent_stack, errors)

Add a new section to the DOM tree, with the given heading.

_add_list(doc, bullet_token, stack, indent_stack, errors)

Add a new list item or field to the DOM tree, with the given bullet or field tag. When necessary, create the associated list.

_tokenize_doctest(lines, start, block_indent, tokens, errors)

Construct a Token containing the doctest block starting at lines[start], and append it to tokens. block_indent should be the indentation of the doctest block. Any errors generated while tokenizing the doctest block will be appended to errors.

Parameters

- **lines**: The list of lines to be tokenized (type=list of string)
- **start**: The index into lines of the first line of the doctest block to be tokenized. (type=int)
- **block_indent**: The indentation of lines[start]. This is the indentation of the doctest block. (type=int)
- **errors**: A list where any errors generated during parsing will be stored. If no list is specified, then errors will generate exceptions. (type=list of ParseError)
- **tokens**: (type=list of Token)

Return Value

The line number of the first line following the doctest block (type=int)
_tokenize_literal(lines, start, block_indent, tokens, errors)

Construct a Token containing the literal block starting at lines[start], and append it to tokens. block_indent should be the indentation of the literal block. Any errors generated while tokenizing the literal block will be appended to errors.

Parameters
lines: The list of lines to be tokenized (type=list of string)
start: The index into lines of the first line of the literal block to be tokenized. (type=int)
block_indent: The indentation of lines[start]. This is the indentation of the literal block. (type=int)
errors: A list of the errors generated by parsing. Any new errors generated while tokenizing this paragraph will be appended to this list. (type=list of ParseError)
tokens: (type=list of Token)

Return Value
The line number of the first line following the literal block. (type=int)

_tokenize_listart(lines, start, bullet_indent, tokens, errors)

Construct Tokens for the bullet and the first paragraph of the list item (or field) starting at lines[start], and append them to tokens. bullet_indent should be the indentation of the list item. Any errors generated while tokenizing will be appended to errors.

Parameters
lines: The list of lines to be tokenized (type=list of string)
start: The index into lines of the first line of the list item to be tokenized. (type=int)
bullet_indent: The indentation of lines[start]. This is the indentation of the list item. (type=int)
errors: A list of the errors generated by parsing. Any new errors generated while tokenizing this paragraph will be appended to this list. (type=list of ParseError)
tokens: (type=list of Token)

Return Value
The line number of the first line following the list item’s first paragraph. (type=int)
_tokenize_para(lines, start, para_indent, tokens, errors)

Construct a Token containing the paragraph starting at lines[start], and append it to tokens.
para_indent should be the indentation of the paragraph. Any errors generated while tokenizing
the paragraph will be appended to errors.

Parameters
lines: The list of lines to be tokenized. (type=list of string)
start: The index into lines of the first line of the paragraph to be tokenized.
(type=int)
paraIndent: The indentation of lines[start]. This is the indentation of the
paragraph. (type=int)
errors: A list of the errors generated by parsing. Any new errors generated
while tokenizing this paragraph will be appended to this list.
(type=list of ParseError)
tokens: (type=list of Token)

Return Value
The line number of the first line following the paragraph. (type=int)

_tokenize(str, errors)

Split a given formatted docstring into an ordered list of Tokens, according to the epytext markup
rules.

Parameters
str: The epytext string. (type=string)
errors: A list where any errors generated during parsing will be stored. If no list is
specified, then errors will generate exceptions. (type=list of ParseError)

Return Value
A list of the Tokens that make up the given string. (type=list of Token)

_colorize(doc, token, errors, tagName='para')

Given a string containing the contents of a paragraph, produce a DOM Element encoding that
paragraph. Colorized regions are represented using DOM Elements, and text is represented using
DOM Texts.

Parameters
errors: A list of errors. Any newly generated errors will be appended to this list.
(type=list of string)
tagName: The element tag for the DOM Element that should be generated.
(type=string)

Return Value
A DOM Element encoding the given paragraph. (type=Element)
Functions

epydoc.markup.epytext

-colorize_graph(doc, graph, token, end, errors)

Eg:
G{classtree} G{classtree x, y, z} G{importgraph}

-colorize_link(doc, link, token, end, errors)

to_epytext(tree, indent=0, secllevel=0)

Convert a DOM document encoding epytext back to an epytext string. This is the inverse operation from parse. I.e., assuming there are no errors, the following is true:

• parse(to_epytext(tree)) == tree

The inverse is true, except that whitespace, line wrapping, and character escaping may be done differently.

• to_epytext(parse(str)) == str (approximately)

Parameters

tree: A DOM document encoding of an epytext string. (type=Element)

indent: The indentation for the string representation of tree. Each line of the returned string will begin with indent space characters. (type=int)

secllevel: The section level that tree appears at. This is used to generate section headings. (type=int)

Return Value
The epytext string corresponding to tree. (type=str)

to_plaintext(tree, indent=0, secllevel=0)

Convert a DOM document encoding epytext to a string representation. This representation is similar to the string generated by to_epytext, but to_plaintext removes inline markup, prints escaped characters in unescaped form, etc.

Parameters

tree: A DOM document encoding of an epytext string. (type=Element)

indent: The indentation for the string representation of tree. Each line of the returned string will begin with indent space characters. (type=int)

secllevel: The section level that tree appears at. This is used to generate section headings. (type=int)

Return Value
The epytext string corresponding to tree. (type=str)
### to_debug(tree, indent=4, seclevel=0)

Convert a DOM document encoding epytext back to an epytext string, annotated with extra debugging information. This function is similar to `to_epytext`, but it adds explicit information about where different blocks begin, along the left margin.

**Parameters**

- **tree**: A DOM document encoding of an epytext string. (type=Element)
- **indent**: The indentation for the string representation of `tree`. Each line of the returned string will begin with `indent` space characters. (type=int)
- **seclevel**: The section level that `tree` appears at. This is used to generate section headings. (type=int)

**Return Value**

The epytext string corresponding to `tree`. (type=string)

### to_rst(tree, indent=0, seclevel=0, wrap_startindex=0)

Convert a DOM document encoding epytext into a reStructuredText markup string. (Because rst is fairly loosely defined, it is possible that this function will produce incorrect output in some cases.)

**Parameters**

- **tree**: A DOM document encoding of an epytext string. (type=Element)
- **indent**: The indentation for the string representation of `tree`. Each line of the returned string will begin with `indent` space characters. (type=int)
- **seclevel**: The section level that `tree` appears at. This is used to generate section headings. (type=int)

**Return Value**

The reStructuredText string corresponding to `tree`. (type=string)

### pparse(str, show_warnings=1, show_errors=1, stream=sys.stderr)

Pretty-parse the string. This parses the string, and catches any warnings or errors produced. Any warnings and errors are displayed, and the resulting DOM parse structure is returned.

**Parameters**

- **str**: The string to parse. (type=string)
- **show_warnings**: Whether or not to display non-fatal errors generated by parsing `str`. (type=boolean)
- **show_errors**: Whether or not to display fatal errors generated by parsing `str`. (type=boolean)
- **stream**: The stream that warnings and errors should be written to. (type=stream)

**Return Value**

a DOM document encoding the contents of `str`. (type=Element)

**Raises**

*SyntaxError* If any fatal errors were encountered.
parse_as_literal(str)

Return a DOM document matching the epytext DTD, containing a single literal block. That literal block will include the contents of the given string. This method is typically used as a fall-back when the parser fails.

Parameters
str: The string which should be enclosed in a literal block. (type=string)

Return Value
A DOM document containing str in a single literal block. (type=Element)

parse_as_para(str)

Return a DOM document matching the epytext DTD, containing a single paragraph. That paragraph will include the contents of the given string. This can be used to wrap some forms of automatically generated information (such as type names) in paragraphs.

Parameters
str: The string which should be enclosed in a paragraph. (type=string)

Return Value
A DOM document containing str in a single paragraph. (type=Element)

parse_docstring(docstring, errors, **options)

Parse the given docstring, which is formatted using epytext; and return a ParsedDocstring representation of its contents.

Parameters
docstring: The docstring to parse (type=string)
errors: A list where any errors generated during parsing will be stored. (type=list of ParseError)
options: Extra options. Unknown options are ignored. Currently, no extra options are defined.

Return Value
ParsedDocstring

24.2 Variables

_HEADING_CHARS
Value: '-~'

_ESCAPES
Value: {'lb': '{', 'rb': '}'}
SYMBOLS
A list of the escape symbols that are supported by epydoc. Currently the following symbols are supported:

- \( S(<->)=\sim; \)
- \( S(\neg\neg)=\neg\neg; \)
- \( S(\neg)=\neg; \)
- \( S(\forall)=\forall; \)
- \( S(\exists)=\exists; \)
- \( S(\alpha)=\alpha; \)
- \( S(\beta)=\beta; \)
- \( S(\gamma)=\gamma; \)
- \( S(\delta)=\delta; \)
- \( S(\epsilon)=\epsilon; \)
- \( S(\zeta)=\zeta; \)
- \( S(\eta)=\eta; \)
- \( S(\theta)=\theta; \)
- \( S(\iota)=\iota; \)
- \( S(\kappa)=\kappa; \)
- \( S(\lambda)=\lambda; \)
- \( S(\mu)=\mu; \)
- \( S(\nu)=\nu; \)
- \( S(\xi)=\xi; \)
- \( S(\omicron)=\omicron; \)
- \( S(\pi)=\pi; \)
- \( S(\rho)=\rho; \)
- \( S(\sigma)=\sigma; \)
- \( S(\tau)=\tau; \)
- \( S(\upsilon)=\upsilon; \)
- \( S(\phi)=\phi; \)
- \( S(\chi)=\chi; \)
- \( S(\psi)=\psi; \)
- \( S(\omega)=\omega; \)
- \( S(\alpha)=\alpha; \)
- \( S(\beta)=\beta; \)
- \( S(\Gamma)=\Gamma; \)
- \( S(\Delta)=\Delta; \)
- \( S(\epsilon)=\epsilon; \)
- \( S(\zeta)=\zeta; \)
- \( S(\eta)=\eta; \)
- \( S(\Theta)=\Theta; \)
- \( S(\iota)=\iota; \)
- \( S(\kappa)=\kappa; \)
- \( S(\Lambda)=\Lambda; \)
- \( S(\mu)=\mu; \)
- \( S(\nu)=\nu; \)
- \( S(\Xi)=\Xi; \)
- \( S(\omicron)=\omicron; \)
- \( S(\Pi)=\Pi; \)
- \( S(\Rho)=\Rho; \)
- \( S(\Sigma)=\Sigma; \)
- \( S(\tau)=\tau; \)
- \( S(\Upsilon)=\Upsilon; \)
- \( S(\Phi)=\Phi; \)
- \( S(\Psi)=\Psi; \)
- \( S(\Chi)=\Chi; \)
__SYMBOLS
Value: {'->': 1, '<-': 1, '<=': 1, '>=': 1, 'Alpha': 1, 'Beta': ...

__doc__
Value: __doc__.replace('<<<SYMBOLS>>>', symblist)

__COLORIZING_TAGS
Value: {'B': 'bold', 'C': 'code', 'E': 'escape', 'G': 'graph', '...

__LINK_COLORIZING_TAGS
Value: ['link', 'uri']

__BULLET_RE
Value: re.compile(r'-( +|$)|\d+\.( +|$)|@\w+( [^{}\n]:)?')

__LIST_BULLET_RE
Value: re.compile(r'-( +|$)|\d+\.( +|$)')

__FIELD_BULLET_RE
Value: re.compile(r'@\w+( [^{}\n]:)?')

__BRACE_RE
Value: re.compile(r'\[\]')

__TARGET_RE
Value: re.compile(r'^([^\<>]+)>$')

GRAPH_TYPES
Value: ['classtree', 'packagetree', 'importgraph', 'callgraph']

SYMBOL_TO_PLAINTEXT
Value: {'crarr': '\'}
24.3 Class Element

A very simple DOM-like representation for parsed epytext documents. Each epytext document is encoded as a tree whose nodes are `Element` objects, and whose leaves are `strings`. Each node is marked by a `tag` and zero or more `attributes`. Each attribute is a mapping from a string key to a string value.

24.3.1 Methods

```python
__init__(self, tag, *children, **attribs)
```

```python
__str__(self)
```

Return a string representation of this element, using XML notation.

**Bug:** Doesn’t escape ‘<’ or ‘&’ or ‘>’.

```python
__repr__(self)
```

24.3.2 Instance Variables

```python
tag
```

A string tag indicating the type of this element.

**Type:** string

```python
children
```

A list of the children of this element.

**Type:** list of (string or Element)

```python
attribs
```

A dictionary mapping attribute names to attribute values for this element.

**Type:** dict from string to string

24.4 Class Token

Tokens are an intermediate data structure used while constructing the structuring DOM tree for a formatted docstring. There are five types of Token:
- Paragraphs
- Literal blocks
- Doctest blocks
- Headings
- Bullets

The text contained in each `Token` is stored in the `contents` variable. The string in this variable has been normalized. For paragraphs, this means that it has been converted into a single line of text, with newline/indentation replaced by single spaces. For literal blocks and doctest blocks, this means that the appropriate amount of leading whitespace has been removed from each line.

Each `Token` has an indentation level associated with it, stored in the `indent` variable. This indentation level is used by the structuring procedure to assemble hierarchical blocks.

### 24.4.1 Methods

```
__init__(self, tag, startline, contents, indent, level=None, inline=False)
```
Create a new `Token`.

**Parameters**
- `tag`: The type of the new `Token`. *(type=string)*
- `startline`: The line on which the new `Token` begins. *(type=int)*
- `contents`: The normalized contents of the new `Token`. *(type=string)*
- `indent`: The indentation of the new `Token` (in number of leading spaces). A value of `None` indicates an unknown indentation. *(type=int or None)*
- `level`: The heading-level of this `Token` if it is a heading; `None`, otherwise. *(type=int or None)*
- `inline`: Is this `Token` inline as a `<span>`? *(type=bool)*

```
__repr__(self)
```
Return Value

the formal representation of this `Token`. `Tokens` have formal representations of the form:

```
<Token: para at line 12>
```

*(type=string)*

```
to_dom(self, doc)
```
Return Value

a DOM representation of this `Token`. *(type=Element)*
### 24.4.2 Class Variables

<table>
<thead>
<tr>
<th>Token</th>
<th>Description</th>
<th>Type</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>PARA</td>
<td>The tag value for paragraph Tokens.</td>
<td>string</td>
<td>'para'</td>
</tr>
<tr>
<td>LBLOCK</td>
<td>The tag value for literal Tokens.</td>
<td>string</td>
<td>'literalblock'</td>
</tr>
<tr>
<td>DTBLOCK</td>
<td>The tag value for doctest Tokens.</td>
<td>string</td>
<td>'doctestblock'</td>
</tr>
<tr>
<td>HEADING</td>
<td>The tag value for heading Tokens.</td>
<td>string</td>
<td>'heading'</td>
</tr>
<tr>
<td>BULLET</td>
<td>The tag value for bullet Tokens. This tag value is also used for field tag Tokens, since fields function syntactically the same as list items.</td>
<td>string</td>
<td>'bullet'</td>
</tr>
</tbody>
</table>

### 24.4.3 Instance Variables

<table>
<thead>
<tr>
<th>Token</th>
<th>Description</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>contents</td>
<td>The normalized text contained in this Token.</td>
<td>string</td>
</tr>
</tbody>
</table>
indent

The indentation level of this Token (in number of leading spaces). A value of None indicates an unknown indentation; this is used for list items and fields that begin with one-line paragraphs.

Type: int or None

inline

If True, the element is an inline level element, comparable to an HTML <span> tag. Else, it is a block level element, comparable to an HTML <div>.

Type: bool

level

The heading-level of this Token if it is a heading; None, otherwise. Valid heading levels are 0, 1, and 2.

Type: int or None

startline

The line on which this Token begins. This line number is only used for issuing errors.

Type: int

tag

This Token’s type. Possible values are Token.PARA (paragraph), Token.LBLOCK (literal block), Token.DTBLOCK (doctest block), Token.HEADINGC, and Token.BULLETC.

Type: string

24.5 Class TokenizationError

An error generated while tokenizing a formatted documentation string.
24.5.1 Methods

Inherited from epydoc.markup.ParseError (Section 22.7, p. 204): \_\_cmp\_\_(), \_\_init\_\_(), \_\_repr\_\_(), \_\_str\_\_(), descr(), is\_fatal(), linenum(), set\_linenum\_offset()

Inherited from exceptions.Exception: \_\_new\_\_()

Inherited from exceptions.BaseException: \_\_delattr\_\_(), \_\_getattribute\_\_(), \_\_getitem\_\_(), \_\_getslice\_\_(), \_\_reduce\_\_(), \_\_setattr\_\_(), \_\_setstate\_\_()

24.5.2 Properties

Inherited from exceptions.BaseException: args, message

24.5.3 Instance Variables

Inherited from epydoc.markup.ParseError (Section 22.7, p. 204): descr, fatal, linenum, offset

24.6 Class StructuringError

An error generated while structuring a formatted documentation string.

24.6.1 Methods

Inherited from epydoc.markup.ParseError (Section 22.7, p. 204): \_\_cmp\_\_(), \_\_init\_\_(), \_\_repr\_\_(), \_\_str\_\_(), descr(), is\_fatal(), linenum(), set\_linenum\_offset()

Inherited from exceptions.Exception: \_\_new\_\_()

Inherited from exceptions.BaseException: \_\_delattr\_\_(), \_\_getattribute\_\_(), \_\_getitem\_\_(), \_\_getslice\_\_(), \_\_reduce\_\_(), \_\_setattr\_\_(), \_\_setstate\_\_()
24.6.2 Properties

Inherited from exceptions.BaseException: args, message

24.6.3 Instance Variables

Inherited from epydoc.markup.ParseError(Section 22.7, p. 204): _descr, _fatal, _linenum, _offset

24.7 Class ColorizingError

An error generated while colorizing a paragraph.

24.7.1 Methods

```python
__init__(self, descr, token, charnum, is_fatal=1)
```

Construct a new colorizing exception.

**Parameters**

- **descr**: A short description of the error. *(type=string)*
- **token**: The token where the error occurred *(type=Token)*
- **charnum**: The character index of the position in token where the error occurred. *(type=int)*

**Overrides**: exceptions.BaseException.__init__

```python
descr(self)
```

**Overrides**: epydoc.markup.ParseError.descr

Inherited from epydoc.markup.ParseError(Section 22.7, p. 204): __cmp__, __repr__, __str__, is_fatal(), linenum(), set_linenum_offset()

Inherited from exceptions.Exception: __new__()
Inherited from exceptions.BaseException: `delattr()`, `getattribute()`, `getitem()`, `getslice()`, `reduce()`, `setattr()`, `setstate()`

24.7.2 Properties

Inherited from exceptions.BaseException: `args`, `message`

24.7.3 Class Variables

<table>
<thead>
<tr>
<th>CONTEXT_RANGE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Value: 20</td>
</tr>
</tbody>
</table>

24.7.4 Instance Variables

Inherited from epydoc.markup.ParseError (Section 22.7, p. 204): `_descr`, `_fatal`, `_linenum`, `_offset`

24.8 Class ParsedEpytextDocstring

```python
def __init__(self, dom_tree, **options)
def __str__(self)
```

Known Subclasses: epydoc.markup.pyval_repr.ColorizedPyvalRepr
to_html(
    self, docstring_linker, directory=None, docindex=None,
    context=None, **options)

    Translate this docstring to HTML.

    Parameters
    docstring_linker: An HTML translator for crossreference links into and out of the
docstring.
    options: Any extra options for the output. Unknown options are ignored.

    Return Value
    An HTML fragment that encodes this docstring. (type=str)

    Overrides: epydoc.markup.ParsedDocstring.to_html (inherited documentation)

to_latex(
    self, docstring_linker, directory=None, docindex=None,
    context=None, **options)

    Translate this docstring to LaTeX.

    Parameters
    docstring_linker: A LaTeX translator for crossreference links into and out of the
docstring.
    options: Any extra options for the output. Unknown options are ignored.

    Return Value
    A LaTeX fragment that encodes this docstring. (type=str)

    Overrides: epydoc.markup.ParsedDocstring.to_latex (inherited documentation)

to_plaintext(
    self, docstring_linker, **options)

    Translate this docstring to plaintext.

    Parameters
    docstring_linker: A plaintext translator for crossreference links into and out of the
docstring.
    options: Any extra options for the output. Unknown options are ignored.

    Return Value
    A plaintext fragment that encodes this docstring. (type=str)

    Overrides: epydoc.markup.ParsedDocstring.to_plaintext (inherited documentation)

_index_term_key(self, tree)

_to_html(
    self, tree, linker, directory, docindex, context, indent=0,
    secllevel=0)

_build_graph(
    self, graph_type, graph_args, linker, docindex, context)
_to_latex_(self, tree, linker, directory, docindex, context, indent=0, secllevel=0, breakany=0)

summary_(self)

Return Value
A pair consisting of a short summary of this docstring and a boolean value indicating whether there is further documentation in addition to the summary. Typically, the summary consists of the first sentence of the docstring. (type=(ParsedDocstring, bool))

Overrides: epydoc.markup.ParsedDocstring.summary (inherited documentation)

split_fields_(self, errors=None)

Split this docstring into its body and its fields.

Parameters
errors: A list where any errors generated during splitting will be stored. If no list is specified, then errors will be ignored.

Return Value
A tuple (body, fields), where body is the main body of this docstring, and fields is a list of its fields. If the resulting body is empty, return None for the body. (type=(ParsedDocstring, list of Field))

Overrides: epydoc.markup.ParsedDocstring.split_fields (inherited documentation)

index_terms_(self)

Return Value
The list of index terms that are defined in this docstring. Each of these items will be added to the index page of the documentation. (type=list of ParsedDocstring)

Overrides: epydoc.markup.ParsedDocstring.index_terms (inherited documentation)

_index_terms_(self, tree, terms)

Inherited from epydoc.markup.ParsedDocstring(Section 22.3, p. 199): _add_(), concatenate()

24.8.2 Class Variables

**SYMBOL_TO_HTML**
Value: {'->': '&rarr;', '<-': '&larr;', '<=': '&le;', '>=': '&ge'}

**SYMBOL_TO_LATEX**
Value: {'->': '\(\rightarrow\)', '<-': '\(\leftarrow\)', '<=': '\(\le\)', '>=': '\(\ge\)'}
25 Module epydoc.markup.javadoc

Epydoc parser for Javadoc\(^2\) docstrings. Javadoc is an HTML-based markup language that was developed for documenting Java APIs with inline comments. It consists of raw HTML, augmented by Javadoc tags. There are two types of Javadoc tags:

- **Javadoc block tags** correspond to Epydoc fields. They are marked by starting a line with a string of the form "@tag [arg]", where tag indicates the type of block, and arg is an optional argument. (For fields that take arguments, Javadoc assumes that the single word immediately following the tag is an argument; multi-word arguments cannot be used with javadoc.)
- **inline Javadoc tags** are used for inline markup. In particular, epydoc uses them for crossreference links between documentation. Inline tags may appear anywhere in the text, and have the form "{@tag [args...]}", where tag indicates the type of inline markup, and args are optional arguments.

Epydoc supports all Javadoc tags, except:

- `{@docRoot}`, which gives the (relative) URL of the generated documentation’s root.
- `{@inheritDoc}`, which copies the documentation of the nearest overridden object. This can be used to combine the documentation of the overridden object with the documentation of the overriding object.
- `@serial`, `@serialField`, and `@serialData` which describe the serialization (pickling) of an object.
- `@value`, which copies the value of a constant.

 Warning: Epydoc only supports HTML output for Javadoc docstrings.

25.1 Functions

parse_docstring(docstring, errors, **options)

Parse the given docstring, which is formatted using Javadoc; and return a ParsedDocstring representation of its contents.

Parameters

- **docstring**: The docstring to parse (type=string)
- **errors**: A list where any errors generated during parsing will be stored. (type=list of ParseError)
- **options**: Extra options. Unknown options are ignored. Currently, no extra options are defined.

Return Value

ParsedDocstring

\(^2\)http://java.sun.com/j2se/javadoc/
Class ParsedJavadocDocstring

An encoded version of a Javadoc docstring. Since Javadoc is a fairly simple markup language, we don’t do any processing in advance; instead, we wait to split fields or resolve crossreference links until we need to.

25.2.1 Methods

```python
_init__(self, docstring, errors=None)
```
Create a new `ParsedJavadocDocstring`.

**Parameters**

- `docstring`: The docstring that should be used to construct this `ParsedJavadocDocstring`. `type=str`
- `errors`: A list where any errors generated during parsing will be stored. If no list is given, then all errors are ignored. `type=list of ParseError`

```python
_check_links(self, errors)
```
Make sure that all `@{link}`s are valid. We need a separate method for this because we want to do this at parse time, not html output time. Any errors found are appended to `errors`.

```python
to_plaintext(self, docstring_linker, **options)
```
Translate this docstring to plaintext.

**Parameters**

- `docstring_linker`: A plaintext translator for crossreference links into and out of the docstring.
- `options`: Any extra options for the output. Unknown options are ignored.

**Return Value**

A plaintext fragment that encodes this docstring. `type=str`

**Overrides:** `epydoc.markup.ParsedDocstring.to_plaintext` (inherited documentation)
summary(self)

Return Value
A pair consisting of a short summary of this docstring and a boolean value indicating whether there is further documentation in addition to the summary. Typically, the summary consists of the first sentence of the docstring. (type=(ParsedDocstring, bool))

Overrides: epydoc.markup.ParsedDocstring.summary (inherited documentation)

split_fields(self, errors=None)

Split this docstring into its body and its fields.

Parameters
errors: A list where any errors generated during splitting will be stored. If no list is specified, then errors will be ignored.

Return Value
A tuple (body, fields), where body is the main body of this docstring, and fields is a list of its fields. If the resulting body is empty, return None for the body. (type=(ParsedDocstring, list of Field))

Overrides: epydoc.markup.ParsedDocstring.split_fields (inherited documentation)

to_html(self, docstring_linker, **options)

Translate this docstring to HTML.

Parameters
docstring_linker: An HTML translator for crossreference links into and out of the docstring.

options: Any extra options for the output. Unknown options are ignored.

Return Value
An HTML fragment that encodes this docstring. (type=string)

Overrides: epydoc.markup.ParsedDocstring.to_html (inherited documentation)

25.2.2 Class Variables

_SUMMARY_RE
Value: re.compile(r'\s*\[\w\W\]*?\.(\s|$)')
### Field Splitting

**_ARG_FIELDS_**
A list of the fields that take arguments. Since Javadoc doesn’t mark arguments in any special way, we must consult this list to decide whether the first word of a field is an argument or not.

Value: `['group', 'variable', 'var', 'type', 'cvariable', 'cvar', ...]`

**_FIELD_RE_**
A regular expression used to search for Javadoc block tags.

Value: `re.compile(r'(?m)\^s*@\w+\[s\$\])')`

### HTML Output

**_LINK_SPLIT_RE_**
A regular expression used to search for Javadoc inline tags.

Value: `re.compile(r'\{link\(?plain\)?\s\[\}\+\}')`

**_LINK_RE_**
A regular expression used to process Javadoc inline tags.

Value: `re.compile(r'\{link\(?plain\)?\s+(\w#\.)+(?::\?\-\.tap\))...')`
Module `epydoc.markup.plaintext`

Parser for plaintext docstrings. Plaintext docstrings are rendered as verbatim output, preserving all whitespace.

### 26.1 Functions

**`parse_docstring(docstring, errors, **options)`**

<table>
<thead>
<tr>
<th>Return Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>A pair ((d, e)), where (d) is a <code>ParsedDocstring</code> that encodes the contents of the given plaintext docstring; and (e) is a list of errors that were generated while parsing the docstring.((type=\text{ParsedPlaintextDocstring, list of ParseError}))</td>
</tr>
</tbody>
</table>

### 26.2 Class `ParsedPlaintextDocstring`

#### 26.2.1 Methods

**`_init_(self, text, **options)`**

**`to_html(self, docstring_linker, **options)`**

- **Parameters**
  - `docstring_linker`: An HTML translator for crossreference links into and out of the docstring.
  - `options`: Any extra options for the output. Unknown options are ignored.

- **Return Value**
  - An HTML fragment that encodes this docstring.\((type=\text{string})\)

- **Overrides**: `epydoc.markup.ParsedDocstring.to_html` (inherited documentation)
to_latex(self, docstring_linker, **options)

Translate this docstring to LaTeX.

Parameters

docstring_linker: A LaTeX translator for crossreference links into and out of the
docstring.

options: Any extra options for the output. Unknown options are ignored.

Return Value

A LaTeX fragment that encodes this docstring. (type=string)

Overrides: epydoc.markup.ParsedDocstring.to_latex (inherited documentation)

to_plaintext(self, docstring_linker, **options)

Translate this docstring to plaintext.

Parameters

docstring_linker: A plaintext translator for crossreference links into and out of
the docstring.

options: Any extra options for the output. Unknown options are ignored.

Return Value

A plaintext fragment that encodes this docstring. (type=string)

Overrides: epydoc.markup.ParsedDocstring.to_plaintext (inherited documentation)

summary(self)

Return Value

A pair consisting of a short summary of this docstring and a boolean value indicating
whether there is further documentation in addition to the summary. Typically,
the summary consists of the first sentence of the docstring. (type=(ParsedDocstring,
bool))

Overrides: epydoc.markup.ParsedDocstring.summary (inherited documentation)

Inherited from epydoc.markup.ParsedDocstring (Section 22.3, p. 199): __add__(),
concatenate(), index_terms(), split_fields()

26.2.2 Class Variables

 SUMMARY_RE

Value: re.compile(r'\s*\[(\w\W)*?(?:\.\(\s|\$)\|n\[\t ]*n)\)\)')
Module epydoc.markup.pyval

Syntax highlighter for Python values. Currently provides special colorization support for:

- lists, tuples, sets, frozensets, dicts
- numbers
- strings
- compiled regexps

The highlighter also takes care of line-wrapping, and automatically stops generating repr output as soon as it has exceeded the specified number of lines (which should make it faster than pprint for large values). It does not bother to do automatic cycle detection, because maxlines is typically around 5, so it’s really not worth it.

The syntax-highlighted output is encoded using a ParsedEpytextDocstring, which can then be used to generate output in a variety of formats.

27.1 Functions

```python
is_re_pattern(pyval)
```

```python
colorize_pyval(pyval, parse_repr=None, min_score=None, linelen=75, maxlines=5, linebreakok=True, sort=True)
```

27.2 Class _ColorizerState

An object used to keep track of the current state of the pyval colorizer. The `mark()/restore()` methods can be used to set a backup point, and restore back to that backup point. This is used by several colorization methods that first try colorizing their object on a single line (setting linebreakok=False); and then fall back on a multi-line output if that fails. The `score` variable is used to keep track of a 'score', reflecting how good we think this repr is. E.g., unhelpful values like '<Foo instance at 0x12345>' get low scores. If the score is too low, we'll use the parse-derived repr instead.

27.2.1 Methods

```python
__init__(self)
```

```python
mark(self)
```

```python
restore(self, mark)
```
27.2.2 Instance Variables

| score | How good this representation is? |

27.3 Class _Maxlines

A control-flow exception that is raised when PyvalColorizer exceeds the maximum number of allowed lines.

27.3.1 Methods

Inherited from exceptions.Exception: __init__(), __new__()  

Inherited from exceptions.BaseException: __delattr__(), __getattr__(), __getattribute__(), __getitem__(), __getslice__(), __reduce__(), __repr__(), __setattr__(), __setstate__(), __str__()

27.3.2 Properties

Inherited from exceptions.BaseException: args, message

27.4 Class _Linebreak

A control-flow exception that is raised when PyvalColorizer generates a string containing a newline, but the state object’s linebreakok variable is False.
27.4.1 Methods

Inherited from exceptions.Exception: __init__(), __new__()

Inherited from exceptions.BaseException: __delattr__(), __getattr__(), __getitem__(), __getslice__(), __reduce__(), __repr__(), __setattr__(), __setstate__(), __str__()

27.4.2 Properties

Inherited from exceptions.BaseException: args, message

27.5 Class ColorizedPyvalRepr

```python
class ColorizedPyvalRepr:
    def __init__(self, tree, score, is_complete):
        pass
```

27.5.1 Methods

Inherited from epydoc.markup.epytext.ParsedEpytextDocstring (Section 24.8, p. 232): __str__(), _build_graph(), _index_term_key(), _index_terms(), _to_html(), _to_latex(), index_terms(), split_fields(), summary(), to_html(), to_latex(), to_plaintext()

27.5.2 Class Variables

Inherited from epydoc.markup.epytext.ParsedEpytextDocstring (Section 24.8, p. 232): SYMBOL_TO_HTML, SYMBOL_TO_LATEX, SUMMARY_RE

244
27.5.3 Instance Variables

**is_complete**
True if this colorized repr completely describes the object.

**score**
A score, evaluating how good this repr is.

27.6 Class `PyvalColorizer`

Syntax highlighter for Python values.

27.6.1 Methods

```python
_init_(self, linelen=75, maxlines=5, linebreakok=True, sort=True)
```

**colorize**(self, pyval, parse_repr=None, min_score=None)

Return Value
A `ColorizedPyvalRepr` describing the given pyval.

```python
_colorize(self, pyval, state)
```

```python
_sort(self, items)
```

```python
_trim_result(self, result, num_chars)
```

```python
_multiline(self, func, pyval, state, *args)
```

Helper for container-type colorizers. First, try calling `func(pyval, state, *args)` with linebreakok set to false; and if that fails, then try again with it set to true.

```python
_colorize_iter(self, pyval, state, prefix, suffix)
```

```python
_colorize_dict(self, items, state, prefix, suffix)
```

```python
_colorize_str(self, pyval, state, prefix, encoding)
```

```python
_colorize_re(self, pyval, state)
```

```python
_colorize_re_flags(self, flags, state)
```
_colorize_re_tree(self, tree, state, noparen, groups)

_output(self, s, tag, state)

Add the string ‘s’ to the result list, tagging its contents with tag ‘tag’. Any lines that go beyond ‘self.linelen’ will be line-wrapped. If the total number of lines exceeds ‘self.maxlines’, then raise a ‘_Maxlines’ exception.

27.6.2 Class Variables

GROUP_TAG
    Value: 'variable-group'

COMMA_TAG
    Value: 'variable-op'

COLON_TAG
    Value: 'variable-op'

CONST_TAG
    Value: None

NUMBER_TAG
    Value: None

QUOTE_TAG
    Value: 'variable-quote'

STRING_TAG
    Value: 'variable-string'

RE_CHAR_TAG
    Value: None

RE_GROUP_TAG
    Value: 're-group'
<table>
<thead>
<tr>
<th>Tag</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>RE_REF_TAG</td>
<td>'re-ref'</td>
</tr>
<tr>
<td>RE_OP_TAG</td>
<td>'re-op'</td>
</tr>
<tr>
<td>RE_FLAGS_TAG</td>
<td>'re-flags'</td>
</tr>
<tr>
<td>ELLIPSIS</td>
<td><code>Element(code, u'...', style='variable-ellipsis')</code></td>
</tr>
<tr>
<td>LINEWRAP</td>
<td><code>Element(symbol, u'crarr')</code></td>
</tr>
<tr>
<td>UNKNOWN_REPR</td>
<td><code>Element(code, u'??', style='variable-unknown')</code></td>
</tr>
<tr>
<td>GENERIC_OBJECT_RE</td>
<td><code>re.compile(r'(?i)^&lt;.* at 0x[0-9a-f]+&gt;$')</code></td>
</tr>
<tr>
<td>ESCAPE_UNICODE</td>
<td>False</td>
</tr>
</tbody>
</table>
Module epydoc.markup.restructuredtext

Epydoc parser for ReStructuredText strings. ReStructuredText is the standard markup language used by the Docutils project. parse_docstring() provides the primary interface to this module; it returns a ParsedRstDocstring, which supports all of the methods defined by ParsedDocstring.

ParsedRstDocstring is basically just a ParsedDocstring wrapper for the docutils.nodes.document class.

Creating ParsedRstDocstrings

ParsedRstDocstrings are created by the parse_document function, using the docutils.core.publish_string() method, with the following helpers:

- An _EpydocReader is used to capture all error messages as it parses the docstring.
- A _DocumentPseudoWriter is used to extract the document itself, without actually writing any output. The document is saved for further processing. The settings for the writer are copied from docutils.writers.html4css1.Writer, since those settings will be used when we actually write the docstring to html.

Using ParsedRstDocstrings

ParsedRstDocstrings support all of the methods defined by ParsedDocstring; but only the following four methods have non-default behavior:

- to_html() uses an _EpydocHTMLTranslator to translate the ParsedRstDocstring’s document into an HTML segment.
- split_fields() uses a _SplitFieldsTranslator to divide the ParsedRstDocstring’s document into its main body and its fields. Special handling is done to account for consolidated fields.
- summary() uses a _SummaryExtractor to extract the first sentence from the ParsedRstDocstring’s document.
- to_plaintext() uses document.astext() to convert the ParsedRstDocstring’s document to plaintext.

To Do: Add ParsedRstDocstring.to_latex()
28.1 Functions

**parse_docstring**(*docstring, errors, **options*)

Parse the given docstring, which is formatted using ReStructuredText; and return a ParsedDocstring representation of its contents.

**Parameters**

- `docstring`: The docstring to parse *(type=string)*
- `errors`: A list where any errors generated during parsing will be stored. *(type=list of ParseError)*
- `options`: Extra options. Unknown options are ignored. Currently, no extra options are defined.

**Return Value**

ParsedDocstring

latex_head_prefix()

**python_code_directive**(*name, arguments, options, content, lineno, content_offset, block_text, state, state_machine*)

A custom restructuredtext directive which can be used to display syntax-highlighted Python code blocks. This directive takes no arguments, and the body should contain only Python code. This directive can be used instead of doctest blocks when it is inconvenient to list prompts on each line, or when you would prefer that the output not contain prompts (e.g., to make copy/paste easier).

**term_role**(*name, rawtext, text, lineno, inliner, options={}, content=[])
<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>_construct_digraph</code></td>
<td>Graph generator for <code>digraph_directive</code>.</td>
</tr>
</tbody>
</table>
| `classtreeDirective`                           | A custom restructuredtext directive which can be used to graphically display a class hierarchy. If one or more arguments are given, then those classes and all their descendants will be displayed. If no arguments are given, and the directive is in a class’s docstring, then that class and all its descendants will be displayed. It is an error to use this directive with no arguments in a non-class docstring. Options:  
  - :dir: – Specifies the orientation of the graph. One of down, right (default), left, up.                                                                 |
| `_construct_classtree`                        | Graph generator for `classtree_directive`.                                                                                                                                                                  |
| `packagetreeDirective`                         | A custom restructuredtext directive which can be used to graphically display a package hierarchy. If one or more arguments are given, then those packages and all their submodules will be displayed. If no arguments are given, and the directive is in a package’s docstring, then that package and all its submodules will be displayed. It is an error to use this directive with no arguments in a non-package docstring. Options:  
  - :dir: – Specifies the orientation of the graph. One of down, right (default), left, up.                                                                 |
| `_construct_packagetree`                      | Graph generator for `packagetree_directive`.                                                                                                                                                                |
| `importgraphDirective`                        |                                                                                                                                                                                                              |
| `_construct_importgraph`                      | Graph generator for `importgraph_directive`.                                                                                                                                                                |
| `callgraphDirective`                          |                                                                                                                                                                                                              |
28.2 Variables

**CONSOLIDATED_FIELDS**
A dictionary whose keys are the "consolidated fields" that are recognized by epydoc; and whose values are the corresponding epydoc field names that should be used for the individual fields.

Value: `{'arguments': 'arg', 'cvariables': 'cvar', 'exceptions': ...}

**CONSOLIDATED_DEFLIST_FIELDS**
A list of consolidated fields whose bodies may be specified using a definition list, rather than a bulleted list. For these fields, the 'classifier' for each term in the definition list is translated into a @type field.

Value: `['param', 'arg', 'var', 'ivar', 'cvar', 'keyword']`

**_TARGET_RE**
Value: `re.compile(r'^(.*?)\s*<(?:URI:|L:)?([^<>]+)>$')`

28.3 Class OptimizedReporter

A reporter that ignores all debug messages. This is used to shave a couple seconds off of epydoc’s run time, since docutils isn’t very fast about processing its own debug messages.

28.3.1 Methods

**debug**(self, *args, **kwargs)

Level-0, “DEBUG”: an internal reporting issue. Typically, there is no effect on the processing. Level-0 system messages are handled separately from the others.

Overrides: `docutils.utils.Reporter.debug (inherited documentation)`

Inherited from docutils.utils.Reporter: `_init_()`, `attach_observer()`, `detach_observer()`, `error()`, `info()`, `notifyObservers()`, `set_conditions()`, `severe()`, `system_message()`, `warning()`
28.3.2 Class Variables

Inherited from docutils.utils.Reporter: DEBUG_LEVEL, ERROR_LEVEL, INFO_LEVEL, SEVERE_LEVEL, WARNING_LEVEL, levels

28.3.3 Instance Variables

Inherited from docutils.utils.Reporter: debug_flag, encoding, error_handler, halt_level, max_level, observers, report_level, source, stream

28.4 Class ParsedRstDocstring

An encoded version of a ReStructuredText docstring. The contents of the docstring are encoded in the _document instance variable.

28.4.1 Methods

```python
__init__(self, document)
```
Parameters

document: (type=docutils.nodes.document)

```python
split_fields(self, errors=None)
```
Split this docstring into its body and its fields.

Parameters

errors: A list where any errors generated during splitting will be stored. If no list is specified, then errors will be ignored.

Return Value

A tuple (body, fields), where body is the main body of this docstring, and fields is a list of its fields. If the resulting body is empty, return None for the body. (type=(ParsedDocstring, list of Field))

Overrides: epydoc.markup.ParsedDocstring.split_fields (inherited documentation)
**summary**(self)

Return Value

A pair consisting of a short summary of this docstring and a boolean value indicating whether there is further documentation in addition to the summary. Typically, the summary consists of the first sentence of the docstring. (type=(ParsedDocstring, bool))

Overrides: epydoc.markup.ParsedDocstring.summary (inherited documentation)

**to_html**(self, docstring_linker=None, docindex=None, context=None, **options)**

Translate this docstring to HTML.

Parameters

- **docstring_linker**: An HTML translator for crossreference links into and out of the docstring.
- **options**: Any extra options for the output. Unknown options are ignored.

Return Value

An HTML fragment that encodes this docstring. (type=string)

Overrides: epydoc.markup.ParsedDocstring.to_html (inherited documentation)

**to_latex**(self, docstring_linker=None, docindex=None, context=None, **options)**

Translate this docstring to LaTeX.

Parameters

- **docstring_linker**: A LaTeX translator for crossreference links into and out of the docstring.
- **options**: Any extra options for the output. Unknown options are ignored.

Return Value

A LaTeX fragment that encodes this docstring. (type=string)

Overrides: epydoc.markup.ParsedDocstring.to_latex (inherited documentation)

**to_plaintext**(self, docstring_linker, **options)**

Translate this docstring to plaintext.

Parameters

- **docstring_linker**: A plaintext translator for crossreference links into and out of the docstring.
- **options**: Any extra options for the output. Unknown options are ignored.

Return Value

A plaintext fragment that encodes this docstring. (type=string)

Overrides: epydoc.markup.ParsedDocstring.to_plaintext (inherited documentation)
28.4.2 Instance Variables

_document
A ReStructuredText document, encoding the docstring.
Type: docutils.nodes.document

28.5 Class _EpydocReader

A reader that captures all errors that are generated by parsing, and appends them to a list.
28.5.1 Methods

**get_transforms** *(self)*

Transforms required by this class. Override in subclasses.

**Overrides:** `docutils.TransformSpec.get_transforms (inherited documentation)`

**__init__** *(self, errors)*

Initialize the Reader instance.

Several instance attributes are defined with dummy initial values. Subclasses may use these attributes as they wish.

**Overrides:** `docutils.readers.Reader.__init__ (inherited documentation)`

**new_document** *(self)*

Create and return a new empty document tree (root node).

**Overrides:** `docutils.readers.Reader.new_document (inherited documentation)`

**report** *(self, error)*

Inherited from `epydoc.docwriter.xlink.ApiLinkReader (Section 19.7, p. 182): read(), read_configuration()`

Inherited from `docutils.readers.Reader: parse(), set_parser()`

Inherited from `docutils.Component: supports()`

28.5.2 Class Variables

**default_transforms**

Value: `list(ApiLinkReader.default_transforms)`

**v**

Value: `'5'`

Inherited from `epydoc.docwriter.xlink.ApiLinkReader (Section 19.7, p. 182): _conf, settings_spec`

Inherited from `docutils.readers.standalone.Reader: config_section, config_section_dependencies, document, supported`

Inherited from `docutils.readers.Reader: component_type`
28.6 Class _DocumentPseudoWriter

A pseudo-writer for the docutils framework, that can be used to access the document itself. The output of _DocumentPseudoWriter is just an empty string; but after it has been used, the most recently processed document is available as the instance variable document.

28.6.1 Methods

__init__(self)

Overrides: docutils.writers.Writer.__init__

translate(self)


Usually done with a docutils.nodes.NodeVisitor subclass, in combination with a call to docutils.nodes.Node.walk() or docutils.nodes.Node.walkabout(). The NodeVisitor subclass must support all standard elements (listed in docutils.nodes.node_class_names) and possibly non-standard elements used by the current Reader as well.

Overrides: docutils.writers.Writer.translate (inherited documentation)

Inherited from docutils.writers.Writer: assemble_parts(), get_transforms(), write()

Inherited from docutils.Component: supports()

28.6.2 Class Variables

Inherited from docutils.writers.Writer: component_type, config_section, destination, language, output
Inherited from docutils.Component: supported

Inherited from docutils.SettingsSpec: config_section_dependencies, relative_path_settings, settings_default_overrides, settings_defaults, settings_spec

Inherited from docutils.TransformSpec: default_transforms, unknown_reference_resolvers

### 28.6.3 Instance Variables

**document**

The document to write (Docutils doctree); set by `write`.

*Type:* `docutils.nodes.document`

Inherited from docutils.writers.Writer: parts

### 28.7 Class `_SummaryExtractor`

A `docutils` node visitor that extracts the first sentence from the first paragraph in a document.

#### 28.7.1 Methods

```python
__init__(self, document)
```

*Overrides:* `docutils.nodes.NodeVisitor.__init__`

```python
visit_document(self, node)
```

```python
visit_paragraph(self, node)
```

```python
visit_field(self, node)
```

```python
unknown_visit(self, node)
```

*Ignore all unknown nodes*

*Overrides:* `docutils.nodes.NodeVisitor.unknown_visit`
Inherited from docutils.nodes.NodeVisitor: dispatch_departure(), dispatch_visit(), unknown_departure()

28.7.2 Class Variables

_SUMMARY_RE
Value: `re.compile(r'\s*(\w\W*?\.)(\s|$)')`

Inherited from docutils.nodes.NodeVisitor: optional

28.8 Class _TermsExtractor

A docutils node visitor that extracts the terms from documentation.
Terms are created using the :term: interpreted text role.

28.8.1 Methods

```python
__init__(self, document)
Overrides: docutils.nodes.NodeVisitor.__init__

visit_document(self, node)

visit_emphasis(self, node)

depart_emphasis(self, node)

visit_Text(self, node)

unknown_visit(self, node)
Ignore all unknown nodes
Overrides: docutils.nodes.NodeVisitor.unknown_visit
```
unknown\_departure(self, node)

Ignore all unknown nodes

Overrides: docutils.nodes.NodeVisitor.unknown\_departure

Inherited from docutils.nodes.NodeVisitor: dispatch\_departure(), dispatch\_visit()

28.8.2 Class Variables

Inherited from docutils.nodes.NodeVisitor: optional

28.8.3 Instance Variables

terms

The terms currently found.

Type: list

28.9 Class _SplitFieldsTranslator

A docutils translator that removes all fields from a document, and collects them into the instance variable fields

28.9.1 Methods

_init_(self, document, errors)

Overrides: docutils.nodes.NodeVisitor._init_

visit\_document(self, node)

visit\_field(self, node)

_add\_field(self, tagname, arg, fbody)

visit\_field\_list(self, node)
Class _EpydocDocumentClass  Module epydoc.markup.restructuredtext

handle_consolidated_field(self, body, tagname)
    Attempt to handle a consolidated section.

handle_consolidated_bullet_list(self, items, tagname)

handle_consolidated_definition_list(self, items, tagname)

unknown_visit(self, node)
    Ignore all unknown nodes
    Overrides: docutils.nodes.NodeVisitor.unknown_visit

Inherited from docutils.nodes.NodeVisitor: dispatch_departure(), dispatch_visit(), unknown_departure()

28.9.2 Class Variables

ALLOW_UNMARKED_ARG_IN_CONsolidated_FIELD
    If true, then consolidated fields are not required to mark arguments with 'backticks'. (This
    is currently only implemented for consolidated fields expressed as definition lists; consolidated
    fields expressed as unordered lists still require backticks for now.
    Value: True

Inherited from docutils.nodes.NodeVisitor: optional

28.9.3 Instance Variables

fields
    The fields of the most recently walked document.
    Type: list of Field

28.10 Class _EpydocDocumentClass

28.10.1 Methods

section(self, level)
28.10.2 Class Variables

**SECTIONS**
Value: ['EpydocUserSection', 'EpydocUserSubsection', 'EpydocUser...

28.11 Class _EpydocLaTeXTranslator

28.11.1 Methods

```python
__init__(self, document, docstring_linker=None, directory=None, docindex=None, context=None)

Overrides: docutils.nodes.NodeVisitor.__init__
```

```python
visit_title_reference(self, node)

Overrides: docutils.writers.latex2e.LaTeXTranslator.visit_title_reference
```

```python
visit_document(self, node)

Overrides: docutils.writers.latex2e.LaTeXTranslator.visit_document
```

```python
depart_document(self, node)

Overrides: docutils.writers.latex2e.LaTeXTranslator.depart_document
```

```python
visit_dotgraph(self, node)
```

```python
visit_doctest_block(self, node)

Overrides: docutils.writers.latex2e.LaTeXTranslator.visit_doctest_block
```

```python
visit_admonition(self, node, name='')

Overrides: docutils.writers.latex2e.LaTeXTranslator.visit_admonition
```
depart_admonition(self, node=None)

Overrides: docutils.writers.latex2e.LaTeXTranslator.depart_admonition

Inherited from docutils.writers.latex2e.LaTeXTranslator: astext, attval, bookmark, depart_Text, depart_address, depart_attention, depart_author, depart_block_quote, depart_bullet_list, depart_caption, depart_cautions, depart_citation_reference, depart_classifier, depart_citespec, depart_compound, depart_contact, depart_container, depart_copyright, depart_danger, depart_date, depart_decoration, depart_definition, depart_definition_list, depart_definition_list_item, depart_description, depart_docinfo, depart_docinfo_item, depart_docstrip_block, depart_emphasis, depart_entry, depart_enumerated_list, depart_error, depart_field, depart_field_argument, depart_field_body, depart_field_list, depart_field_name, depart_figure, depart_footer, depart_footnote, depart_footnote_reference, depart_generated, depart_header, depart_hint, depart_image, depart_import, depart_inline, depart_interpret, depart_label, depart_legend, depart_line, depart_line_block, depart_list_item, depart_literal, depart_literal_block, depart_meta, depart_note, depart_option, depart_option_argument, depart_option_group, depart_option_list, depart_option_list_item, depart_option_string, depart_organization, depart_paragraph, depart_problematic, depart_reference, depart_revision, depart_row, depart_rubric, depart_section, depart_sidebar, depart_status, depart_strong, depart_subscript, depart_subtitle, depart_superscript, depart_system_message, depart_table, depart_target, depart_tbody, depart_term, depart_tgroup, depart_thead, depart_tip, depart_title, depart_title_reference, depart_topic, depart_transition, depart_version, depart_warning, encode, ensure_math, label_delim, language_label, latex_image_length, literal_block_env, to_latex_encoding, unicodes_to_latex, unimplemented_visit, visit_Text, visit_address, visit_attention, visit_author, visit_authors, visit_block_quote, visit_bullet_list, visit_citation, visit_caution, visit_citation, visit_citation_reference, visit_classifier, visit_citespec, visit_comment, visit_compound, visit_contact, visit_container, visit_copyright, visit_danger, visit_date, visit_decoration, visit_definition, visit_definition_list, visit_definition_list_item, visit_description, visit_docinfo, visit_docinfo_item, visit_emphasis, visit_entry, visit_enumerated_list, visit_error, visit_field, visit_field_argument, visit_field_body, visit_field_list, visit_field_name, visit_figure, visit_footer, visit_footnote, visit_footnote_reference, visit_generated, visit_header, visit_hint, visit_image, visit_important, visit_inline, visit_interpret, visit_label, visit_legend, visit_line, visit_line_block, visit_list_item, visit_literal, visit_literal_block, visit_meta, visit_note, visit_option, visit_option_argument, visit_option_group, visit_option_list, visit_option_list_item, visit_option_string, visit_organization, visit_paragraph, visit_problematic, visit_raw, visit_reference, visit_revision, visit_row, visit_rubric, visit_section, visit_sidebar, visit_status, visit_strong, visit_subscript, visit_substitution_definition, visit_substitution_reference, visit_subtitle, visit_superscript, visit_system_message, visit_table, visit_target, visit_tbody, visit_term, visit_tgroup, visit_thead, visit_tip, visit_title, visit_topic, visit_transition, visit_version, visit_warning
28.11.2 Class Variables

settings

Value: None

Inherited from docutils.writers.latex2e.LaTeXTranslator: attribution_formats, compound_enumerators, generator, hyperlink_color, latex_equivalents, latex_head, linking, section_enum_separator, section_prefix_for_enumerators, stylesheet, use_latex_toc, use_optionlist_for_docinfo

Inherited from docutils.nodes.NodeVisitor: optional

28.12 Class _EpydocHTMLTranslator

28.12.1 Methods

__init__(self, document, docstring_linker, directory, docindex, context)

Overrides: docutils.nodes.NodeVisitor.__init__

visit_title_reference(self, node)

Overrides: docutils.writers.html4css1.HTMLTranslator.visit_title_reference

should_be_compact_paragraph(self, node)

Determine if the <p> tags around paragraph node can be omitted.

Overrides: docutils.writers.html4css1.HTMLTranslator.should_be_compact_paragraph (inherited documentation)

visit_document(self, node)

Overrides: docutils.writers.html4css1.HTMLTranslator.visit_document

depart_document(self, node)

Overrides: docutils.writers.html4css1.HTMLTranslator.depart_document
starttag(self, node, tagname, suffix='\n', **attributes)

This modified version of starttag makes a few changes to HTML tags, to prevent them from conflicting with epydoc. In particular:

- existing class attributes are prefixed with 'rst-
- existing names are prefixed with 'rst-
- hrefs starting with '#' are prefixed with 'rst-
- hrefs not starting with '#' are given target='_top'
- all headings (<hn>) are given the css class 'heading'

 Overrides: docutils.writers.html4css1.HTMLTranslator.starttag

visit_dotgraph(self, node)

visit_doctest_block(self, node)

 Overrides: docutils.writers.html4css1.HTMLTranslator.visit_doctest_block

visit_emphasis(self, node)

 Overrides: docutils.writers.html4css1.HTMLTranslator.visit_emphasis
28.12.2 Class Variables

**settings**

Value: None

Inherited from docutils.writers.html4css1.HTMLTranslator: attribution_formats, content_type, doctype, embedded_stylesheet, generator, head_prefix_template, stylesheet_link, words_and_spaces, xml_declaration
28.13 Class \texttt{dotgraph}

A custom docutils node that should be rendered using Graphviz dot. This node does not directly store the graph; instead, it stores a pointer to a function that can be used to generate the graph. This allows the graph to be built based on information that might not be available yet at parse time. This graph generation function has the following signature:

\begin{verbatim}
>>> def generate_graph(docindex, context, linker, *args):
...     'generates and returns a new DotGraph'
\end{verbatim}

Where \texttt{docindex} is a docindex containing the documentation that epydoc has built; \texttt{context} is the \texttt{APIDoc} whose docstring contains this dotgraph node; \texttt{linker} is a \texttt{DocstringLinker} that can be used to resolve crossreferences; and \texttt{args} is any extra arguments that are passed to the \texttt{dotgraph} constructor.

28.13.1 Methods

\begin{verbatim}
    __init__(self, generate_graph_func, *generate_graph_args)

    graph(self, docindex, context, linker)
\end{verbatim}

Inherited from docutils.nodes.image: \texttt{astext()}

Inherited from docutils.nodes.Element: \texttt{__add__(), __delitem__(), __getitem__(), __iadd__(), __len__(),
\_radd__(), \_repr__(), \_setitem__(), \_unicode__(), \_dom_node(), append(), attlist(), clear(), copy(),
deepequal(), delattr(), emptytag(), endtag(), extend(), first_child_matching_class(),
first_child_not_matching_class(), get(), has_key(), hasattr(), index(), insert(), is_not_default(),
non_default_attributes(), note_referenced_by(), pformat(), pop(), remove(), replace(), replace_self(),
set_class(), setdefault(), shortrepr(), starttag(), update_basic_atts()}

Inherited from docutils.nodes.Node: \texttt{__nonzero__(), __str__(), asdom(), next_node(), setup_child(),
traverse(), walk(), walkabout()}

266
### 28.13.2 Class Variables

| Inherited from docutils.nodes.Element: child_text_separator, list_attributes, tagname |
| Inherited from docutils.nodes.Node: document, line, parent, source |

### 28.13.3 Instance Variables

| Inherited from docutils.nodes.Element: attributes, children, rawsource |
29  Package  epydoc.test

Regression testing.

29.1  Functions

main()

check_requirements(filename)

Search for strings of the form:

[Require: <module>]

If any are found, then try importing the module named <module>. If the import fails, then return False. If all required modules are found, return True. (This includes the case where no requirements are listed.)
## Module epydoc.test.util

Utility functions used by the regression tests (`.doctest`).

### 30.1 Functions

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>buildvaluedoc</strong>(s)</td>
<td>This test function takes a string containing the contents of a module. It writes the string contents to a file, imports the file as a module, and uses build_doc to build documentation, and returns it as a ValueDoc object.</td>
</tr>
<tr>
<td><strong>runbuilder</strong>(s, <strong>attribs</strong>='', <strong>build</strong>=None, <strong>exclude</strong>='')</td>
<td>This test function takes a string containing the contents of a module. It writes the string contents to a file, imports the file as a module, and uses build_doc to build documentation, and pretty prints the resulting ModuleDoc object. The <strong>attribs</strong> argument specifies which attributes of the APIDocs should be displayed. The <strong>build</strong> argument gives the name of a variable in the module whose documentation should be built, instead of building docs for the whole module.</td>
</tr>
<tr>
<td><strong>runparser</strong>(s, <strong>attribs</strong>='', <strong>show</strong>=None, <strong>exclude</strong>='')</td>
<td>This test function takes a string containing the contents of a module, and writes it to a file, uses 'parse_docs' to parse it, and pretty prints the resulting ModuleDoc object. The 'attribs' argument specifies which attributes of the 'APIDoc's should be displayed. The 'show' argument, if specified, gives the name of the object in the module that should be displayed (but the whole module will always be inspected; this just selects what to display).</td>
</tr>
<tr>
<td><strong>runintrospecter</strong>(s, <strong>attribs</strong>='', <strong>introspect</strong>=None, <strong>exclude</strong>='')</td>
<td>This test function takes a string containing the contents of a module. It writes the string contents to a file, imports the file as a module, and uses introspect_docs to introspect it, and pretty prints the resulting ModuleDoc object. The <strong>attribs</strong> argument specifies which attributes of the APIDocs should be displayed. The <strong>introspect</strong> argument gives the name of a variable in the module whose value should be introspected, instead of introspecting the whole module.</td>
</tr>
<tr>
<td><strong>print_warnings</strong>()</td>
<td>Register a logger that will print warnings &amp; errors.</td>
</tr>
<tr>
<td><strong>testencoding</strong>(s, <strong>introspect</strong>=True, <strong>parse</strong>=True, <strong>debug</strong>=False)</td>
<td>An end-to-end test for unicode encodings. This function takes a given string, writes it to a python file, and processes that file’s documentation. It then generates HTML output from the documentation, extracts all docstrings from the generated HTML output, and displays them. (In order to extract &amp; display all docstrings, it monkey-patches the HTMLwriter.docstring_to_html() method.)</td>
</tr>
</tbody>
</table>
### Helper Functions

<table>
<thead>
<tr>
<th>Function Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>write_pysting_to_tmp_dir</strong> <em>(s)</em></td>
<td></td>
</tr>
<tr>
<td><strong>cleanup_tmp_dir</strong> <em>(tmp_dir)</em></td>
<td></td>
</tr>
<tr>
<td><strong>to_plain</strong> <em>(docstring)</em></td>
<td>Conver a parsed docstring into plain text</td>
</tr>
<tr>
<td><strong>fun_to_plain</strong> <em>(val_doc)</em></td>
<td>Convert parsed docstrings in text from a RoutineDoc</td>
</tr>
<tr>
<td><strong>print_docstring_as_html</strong> *(self, parsed_docstring, *varargs, *<em>kwargs)</em></td>
<td>Convert the given parsed docstring to HTML and print it. Ignore any other arguments. This function is used by <code>testencoding</code> to monkey-patch the HTMLWriter class’s <code>docstring_to_html()</code> method.</td>
</tr>
<tr>
<td><strong>remove_surrogates</strong> <em>(s)</em></td>
<td>The following is a helper function, used to convert two-character surrogate sequences into single characters. This is needed because some systems create surrogates but others don’t.</td>
</tr>
</tbody>
</table>
31 Module epydoc.util

Miscellaneous utility functions that are used by multiple modules.

31.1 Functions

**is_src_filename(filename)**

**munge_script_name(filename)**

**plaintext_to_latex(str, nbsp=0, breakany=0)**

**Parameters**

- **breakany**: Insert hyphenation marks, so that LaTeX can break the resulting string at any point. This is useful for small boxes (e.g., the type box in the variable list table).
- **nbsp**: Replace every space with a non-breaking space (‘~’).

**Return Value**

A LaTeX string that encodes the given plaintext string. In particular, special characters (such as ‘$’ and ‘_’) are escaped, and tabs are expanded. *(type=string)*

**run_subprocess(cmd, data=None)**

Execute the command cmd in a subprocess.

**Parameters**

- **cmd**: The command to execute, specified as a list of string.
- **data**: A string containing data to send to the subprocess.

**Return Value**

A tuple (out, err).

**Raises**

- **OSError** If there is any problem executing the command, or if its exitval is not 0.

**Python source types**

**is_module_file(path)**

**is_package_dir(dirname)**

Return true if the given directory is a valid package directory (i.e., it names a directory that contains a valid __init__.py file, and its name is a valid identifier).

**is_pyname(name)**
**py_src_filename(filename)**

**Text processing**

**decode_with_backslashreplace(s)**

Convert the given 8-bit string into unicode, treating any character c such that ord(c)<128 as an ascii character, and converting any c such that ord(c)>128 into a backslashed escape sequence.

```python
>>> decode_with_backslashreplace('abc\xff\xe8')
u'abc\xff\xe8'
```

**wordwrap(str, indent=0, right=75, startindex=0, splitchars='\n')**

Word-wrap the given string. I.e., add newlines to the string such that any lines that are longer than right are broken into shorter lines (at the first whitespace sequence that occurs before index right). If the given string contains newlines, they will not be removed. Any lines that begin with whitespace will not be wordwrapped.

**Parameters**

- **indent:** If specified, then indent each line by this number of spaces. *(type=int)*
- **right:** The right margin for word wrapping. Lines that are longer than right will be broken at the first whitespace sequence before the right margin. *(type=int)*
- **startindex:** If specified, then assume that the first line is already preceded by startindex characters. *(type=int)*
- **splitchars:** A list of non-whitespace characters which can be used to split a line. *(E.g., use '/\' to allow path names to be split over multiple lines.)*

**Return Value**

*str*

**plaintext_to_html(s)**

**Return Value**

An HTML string that encodes the given plaintext string. In particular, special characters (such as '<' and '&') are escaped. *(type=str)*

### 31.2 Variables

**PY_SRC_EXTENSIONS**

Value: [''.py', '.pyw']

**PY_BIN_EXTENSIONS**

Value: [''.pyc', '.so', '.pyd']

272
31.3 Class RunSubprocessError

Inherited from exceptions.BaseException:
exceptions.ESError

Inherited from exceptions.StandardError:

Inherited from exceptions.EnvironmentError:

    Inherited from exceptions.BaseException:
        _delattr_(), _getattribute_(), _getitem_(), _getslice_(), _repr_(), _setattr_(), _setstate_()

31.3.1 Methods

    __init__(self, cmd, out, err)

    x.__init__(...) initializes x; see x.__class__.__doc__ for signature

    Overrides: exceptions.BaseException.__init__ (inherited documentation)

    Inherited from exceptions.OSError: _new_()

    Inherited from exceptions.EnvironmentError: _reduce_(), _str_()

31.3.2 Properties

    Inherited from exceptions.EnvironmentError: errno, filename, message, strerror

    Inherited from exceptions.BaseException: args

31.4 Class TerminalController

A class that can be used to portably generate formatted output to a terminal. See http://aspn.activestate.com/ASPN/Cookbook/Python/Recipe/475116 for documentation. (This is a somewhat stripped-down version.)
31.4.1 Methods

```python
_init__(self, term_stream=sys.stdout)

tigetstr(self, cap_name)

render(self, template)
    Replace each $-substitutions in the given template string with the corresponding terminal control
    string (if it's defined) or " (if it's not).

_render_sub(self, match)
```

31.4.2 Class Variables

<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>BOL</td>
<td>Move the cursor to the beginning of the line</td>
<td>''</td>
</tr>
<tr>
<td>UP</td>
<td>Move the cursor up one line</td>
<td>''</td>
</tr>
<tr>
<td>DOWN</td>
<td>Move the cursor down one line</td>
<td>''</td>
</tr>
<tr>
<td>LEFT</td>
<td>Move the cursor left one char</td>
<td>''</td>
</tr>
<tr>
<td>RIGHT</td>
<td>Move the cursor right one char</td>
<td>''</td>
</tr>
<tr>
<td>Variable</td>
<td>Description</td>
<td>Value</td>
</tr>
<tr>
<td>-------------------</td>
<td>----------------------------------------------</td>
<td>-------</td>
</tr>
<tr>
<td>CLEAR_EOL</td>
<td>Clear to the end of the line.</td>
<td>''</td>
</tr>
<tr>
<td>CLEAR_LINE</td>
<td>Clear the current line; cursor to BOL.</td>
<td>''</td>
</tr>
<tr>
<td>BOLD</td>
<td>Turn on bold mode</td>
<td>''</td>
</tr>
<tr>
<td>NORMAL</td>
<td>Turn off all modes</td>
<td>''</td>
</tr>
<tr>
<td>COLS</td>
<td>Width of the terminal (default to 75)</td>
<td>75</td>
</tr>
<tr>
<td>UNDERLINE</td>
<td>Underline the text</td>
<td>''</td>
</tr>
<tr>
<td>REVERSE</td>
<td>Reverse the foreground &amp; background</td>
<td>''</td>
</tr>
<tr>
<td>WHITE</td>
<td></td>
<td>''</td>
</tr>
<tr>
<td>YELLOW</td>
<td></td>
<td>''</td>
</tr>
</tbody>
</table>
**MAGENTA**
Value: ''

**RED**
Value: ''

**CYAN**
Value: ''

**GREEN**
Value: ''

**BLUE**
Value: ''

**BLACK**
Value: ''

__STRING_CAPABILITIES__
Value: ['BOL=cr', 'UP=cuu1', 'DOWN=cud1', 'LEFT=cub1', 'RIGHT=cu...'

__COLORS__
Value: ['BLACK', 'BLUE', 'GREEN', 'CYAN', 'RED', 'MAGENTA', 'YEL...

__ANSICOLORS__
Value: ['BLACK', 'RED', 'GREEN', 'YELLOW', 'BLUE', 'MAGENTA', 'C...

**FORCE_SIMPLE_TERM**
If this is set to true, then new TerminalControllers will assume that the terminal is not capable of doing manipulation of any kind.
Value: False
DoctestColorizer (class), 200–204

colorize_codeblock (method), 201
colorize_doctest (method), 201
colorize_inline (method), 201
markup (method), 201
subfunc (method), 201
doctest (module), 200–208
document (function), 177

DocUrlGenerator (class), 172–174

_getitem__ (method), 172
_setitem__ (method), 178
to_dotfile (method), 178

DotGraphEdge (class), 117–118

_getitem__ (method), 117
_setitem__ (method), 117
to_dotfile (method), 117

DotGraphNode (class), 116–117

_getitem__ (method), 117
_setitem__ (method), 117
to_dotfile (method), 117

DotGraphUmlClassName (class), 118–126

_add_attribute_edge (method), 122
_attribute_cell (method), 123
_get_html_label (method), 123
_link_attribute (method), 123
_operation_arg (method), 123
_operationcell (method), 123
_qualifier_cell (method), 123
_summary (class method), 122
_tooltip (method), 122
_type_descr (method), 122

DotGraphUmlModuleName (class), 126–127

_color (method), 127
_get_html_label (method), 126

DotGraph (class), 113–116

_link_href (method), 114
_pick_language (method), 114
_run_dot (method), 115
to_dot2tex (method), 114
link (method), 114
render (method), 114
to_dotfile (method), 115
to_html (method), 114
to_latex (method), 113
write (method), 114

dotgraph (class), 259–260

graph (method), 259
dotgraph (module), 107–127
dotted_names_in (function), 91

index
DottedName \textit{(class)}, 8–10
\_add\_ (method), 9
\_cmp\_ (method), 9
\_getitem\_ (method), 9
\_hash\_ (method), 9
\_init\_ (method), 9
\_len\_ (method), 9
\_radd\_ (method), 9
\_repr\_ (method), 9
\_str\_ (method), 9
container (method), 10
dominate (method), 10
DottedName.InvalidDottedName \textit{(class)}, 10–11
Element \textit{(class)}, 219
\_init\_ (method), 219
\_repr\_ (method), 219
\_str\_ (method), 219
end\_block (function), 185
der\_progress (function), 186
EpydocGUI \textit{(class)}, 182–184
\_init\_ (method), 183
\_browse\_css (method), 183
\_browse\_help (method), 183
\_browse\_module (method), 183
\_browse\_out (method), 183
\_configure (method), 183
\_delete\_module (method), 183
\_entry\_module (method), 183
\_getopts (method), 184
\_go (method), 184
\_init\_bindings (method), 183
\_init\_menubar (method), 183
\_init\_messages (method), 183
\_init\_module\_list (method), 183
\_init\_options (method), 183
\_init\_progress\_bar (method), 183
\_messages\_toggle (method), 183
\_new\_ (method), 184
\_open\_ (method), 184
\_options\_toggle (method), 183
\_save\_as (method), 184
\_save\_ (method), 184
\_update\_messages (method), 184
\_update\_msg\_tags (method), 183
\_update\_ (method), 184
\_add\_module (method), 184
\_destroy\_ (method), 184
\_mainloop\_ (method), 184
\_open\_ (method), 184
epydoc \textit{(package)}, 2–6
epytext \textit{(module)}, 209–228

\texttt{error} \textit{(function)}, 185
\texttt{fatal} \textit{(function)}, 185
Field \textit{(class)}, 194–195
\_init\_ (method), 194
\_repr\_ (method), 195
\_arg\_ (method), 194
\_body\_ (method), 195
tag (method), 194
\_find\_base \textit{(function)}, 91
\_find\_latex\_error \textit{(function)}, 157
\_find\_overrides \textit{(function)}, 72
\_flatten \textit{(function)}, 94
\_fun\_to\_plain \textit{(function)}, 263
\_GenericValueDoc \textit{(class)}, 24–27
\_get\_canonical\_name \textit{(function)}, 81
\_get\_containing\_module \textit{(function)}, 82
\_get\_docformat \textit{(function)}, 101
\_get\_docstring \textit{(function)}, 81
\_get\_dot\_version \textit{(function)}, 111
\_get\_lhs\_parent \textit{(function)}, 89
\_get\_module\_encoding \textit{(function)}, 93
\_get\_value\_from\_filename \textit{(function)}, 83
\_get\_value\_from\_name \textit{(function)}, 83
\_get\_value\_from\_scriptname \textit{(function)}, 83
\_GUI\_Logger \textit{(class)}, 181–182
\_init\_ \textit{(method)}, 181
\_clear\_ \textit{(method)}, 181
\_read\_ \textit{(method)}, 182
gui \textit{(function)}, 177
gui \textit{(module)}, 177–184
\_handle\_special\_module\_vars \textit{(function)}, 87
\_html\_colorize \textit{(module)}, 148–153
\_html\_css \textit{(module)}, 154–155
\_html\_help \textit{(module)}, 156
\_HTML\_Doctest\_Colorizer \textit{(class)}, 205–207
\_HTML\_Logger \textit{(class)}, 68–70
\_init\_ \textit{(method)}, 68
\_elapsed\_time \textit{(method)}, 69
\_message \textit{(method)}, 69
\_write\_options \textit{(method)}, 68
\_HTML\_Writer \textit{(class)}, 128–146
\_init\_ \textit{(method)}, 130
\_arg\_name \textit{(method)}, 138
\_attr\_to\_html \textit{(method)}, 141
\_crumb \textit{(method)}, 135
\_doc\_or\_ancestor\_is\_private \textit{(method)}, 141
\_find\_top\_page \textit{(method)}, 130
\_group\_by\_letter \textit{(method)}, 139
\_import \textit{(method)}, 138
\_mkdir \textit{(method)}, 131
\_private\_subclasses \textit{(method)}, 141
\_term\_index\_to\_anchor \textit{(method)}, 139
\_terms\_from\_docstring \textit{(method)}, 139
_url (method), 140
_val_is_public (method), 140
_write_redirect_page (method), 140
_write_summary_line (method), 136
_write (method), 131
arg_name_to_html (method), 136
base_tree (method), 137
breadcrumbs (method), 135
build_identifier_index (method), 139
build_metadata_index (method), 139
build_term_index (method), 139
callgraph_link (method), 134
contextual_label (method), 137
description (method), 141
descr (method), 141
doc_kind (method), 141
docstring_to_html (method), 141
find_tree_width (method), 137
func_arg (method), 138
function_signature (method), 137
href (method), 140
labelled_list_item (method), 136
pprint_value (method), 137
property_accessor_to_html (method), 136
pysrc_link (method), 140
pysrc_url (method), 140
render_callgraph (method), 134
render_graph (method), 134
return_desc (method), 141
return_type (method), 141
summary_name (method), 138
summary (method), 141
type_descr (method), 141
url (method), 140
variable_tooltip (method), 137
write_api_list (method), 140
write_breadcrumbs (method), 135
write_class_tree_graph (method), 132
write_class_tree_item (method), 139
write_class_tree (method), 132
write_class (method), 132
write_css (method), 133
write_details_entry (method), 136
write_details_list (method), 136
write_footer (method), 135
write_frames_index (method), 133
write_function_details_entry (method), 137
write_group_header (method), 140
write_header (method), 135
write_help (method), 133
write_homepage (method), 133
write_images (method), 134
write_imports (method), 138
write_index_section (method), 132
write_indexpage_header (method), 132
write_inheritance_list (method), 132
write_javascript (method), 134
write_link_index (method), 132
write_metadata_index (method), 132
write_module_list (method), 138
write_module_toc (method), 133
write_module_tree_item (method), 139
write_module_tree (method), 132
write_module (method), 131
write_navbar (method), 135
write_project_toc (method), 133
write_property_details_entry (method), 137
write_redirect_index (method), 133
write_redirect_page (method), 139
write_sourcecode (method), 131
write_standard_fields (method), 139
write_standard_field (method), 139
write_summary_group (method), 136
write_summary_line (method), 136
write_summary_table (method), 136
write_table_header (method), 140
write_toc_section (method), 133
write_toc (method), 133
write_treepage_header (method), 132
write_url_record (method), 140
write_var_list (method), 136
write_variable_details_entry (method), 137
write (method), 131
html (module), 128–147
import_graph (function), 111
importgraph_directive (function), 243
info (function), 185
inherit_docs (function), 77
init_arglist (function), 91
initialize_api_doc (function), 98
introspect_class (function), 81
introspect_docstring_lineno (function), 83
introspect_docs (function), 80
introspect_module (function), 80
introspect_other (function), 81
introspect_property (function), 81
introspect Routine (function), 81
is_append_to_all (function), 91
is_classmethod (function), 82
is_future_feature (function), 81
is_getset (function), 83
is_member (function), 83
is_module_file (function), 264
is_package_dir (function), 264
is_property (function), 83
is_pyname (function), 264

281
INDEX

is_re_pattern (function), 235
is_src_filename (function), 264
is_staticmethod (function), 82
isclass (function), 81
javadoc (module), 229–232
latex_head_prefix (function), 242
latex_sty (module), 164
LaTeXDoctestColorizer (class), 207–208
LatexWriter (class), 157–162
  __init__ (method), 157
  _arg_name (method), 160
  _base_name (method), 159
  _base_tree_line (method), 159
  _descriptor (method), 160
  _filter_deprecated (method), 161
  _find_tree_width (method), 159
  _is_deprecated (method), 161
  _makedirs (method), 158
  _parents_if_func (method), 159
  _write_sty (method), 158
  _write (method), 158
  base_tree (method), 159
  crossref (method), 161
doc_kind (method), 161
docstring_to_latex (method), 160
docstring (method), 160
function_signature (method), 160
get_latex_encoding (method), 161
indexterm (method), 161
LatexWriter_LatexDocstringLinker (class), 162
merge_overrides (function), 76
write_property (method), 160
write_topfile (method), 158
write_var (method), 160
write (method), 157
latex (module), 157–163
lhs_is_instance (function), 89
link_imports (function), 76
Logger (class), 187–188
close (method), 187
end_block (method), 188
end_progress (method), 188
log (method), 187
progress (method), 188
start_block (method), 188
start_progress (method), 188
log (module), 185–189
lookup_name (function), 93
lookup_value (function), 93
lookup_variable (function), 93
main (function), 62, 261
markup (package), 190–199
merge_attribute (function), 76
merge_bases (function), 76
merge_docs_extracted_by (function), 76
merge_docstring (function), 76
merge_docs (function), 75
merge_del (function), 76
merge_get (function), 76
merge_set (function), 76
merge_posarg_defaults (function), 76
merge_proxy_for (function), 76
merge_submodules (function), 76
merge_value (function), 76
merge_variables (function), 76
mk_valdoc_node (function), 112
ModuleDoc (class), 31–36
  init_submodule_groups (method), 32
  select_variables (method), 32
munge_script_name (function), 264
name_list (function), 112
NamespaceDoc (class), 27–31
  __init__grouping (method), 28
  group_names (method), 28
  init_sorted_variables (method), 28
  init_variable_groups (method), 28
  report_unused_groups (method), 28
OptimizedReporter (class), 244–245
option_defaults (function), 62
package_tree_graph (function), 109
packagetree_directive (function), 243
parse_arguments (function), 62
parse_as_literal (function), 215

282
verify_name (function), 82
VoidUrlGenerator (class), 171–172
warning (function), 185
wordwrap (function), 265
write-html (function), 62
write-latex (function), 63
write-pickle (function), 62
write-pystring-to-tmp-dir (function), 263
write-text (function), 63
xlink (module), 167–176
XMLDoctestColorizer (class), 204–205

bold, 209
code, 209
contained ValueDocs, 52
doctest-block, 209
eytext, 209
field, 209
index, 209
inline Javadoc tags, 229
inline regions, 209
italic, 209
Javadoc block tags, 229
li, 209
link, 209
literal-block, 209
loggers, 187
math, 209
merged, 12
merged object, 14
olist, 209
para, 209
project files, 177

reachable ValueDocs, 52
section, 209
simple fields, 98
special fields, 98
structural blocks, 209
token tree, 88
ulist, 209
uri, 209