Epydoc: API Documentation Extraction in Python

Edward Loper
Department of Computer and Information Science
University of Pennsylvania, Philadelphia, PA 19104-6389, USA

Abstract

Epydoc is a tool for generating API documentation for Python modules, based on their docstrings. It supports several output formats (including HTML and PDF), and understands four different markup languages (Epytext, Javadoc, reStructuredText, and plaintext). A wide variety of *fields* can be used to supply specific information about individual objects, such as descriptions of function parameters, type signatures, and groupings of related objects.

1 Introduction

Documentation is a critical contributor to a library's usability. Thorough documentation shows new users how to use a library; and details the library's specific behavior for advanced users. Most libraries can benefit from three different types of documentation: *tutorial documentation*, which introduces new users to the library by showing them how to perform typical tasks; *reference documentation*, which explains the library's overall design, and describes how the different pieces of the library fit together; and *API documentation*, which describes the individual objects (classes, functions, variables, etc.) that are defined by the library.

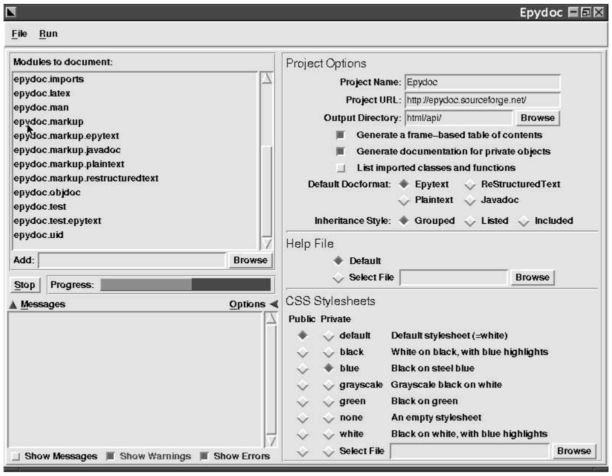
Since API documentation describes individual objects, it is tightly coupled to the library's code. As a result, it can be difficult to ensure that the external API documentation is kept up-to-date whenever the code is changed. Python provides an elegant solution to this problem: docstrings. A *docstring* is a string constant that appears at the top of an object's definition, and is available via inspection. By using docstrings to document a library's API, we can significantly increase the chances that the code and documentations will be kept in sync.

Docstrings are typically accessed via the `pydoc` library, which converts a library's docstrings into manpage style output; or via direct inspection. However, these two methods have a number of limitations:

- All API documentation must be written (and read) in plaintext.
- There is no easy way to navigate through the API documentation.
- The API documentation is not searchable.
- A library's API documentation cannot be viewed until that library is installed.
- There is no mechanism for documenting variables.
- There is no mechanism for "inheriting" documentation (e.g. in a method that overrides its base class method). This can lead to duplication of documentation, which can often get out-of-sync.

Epydoc is a tool that automatically extracts a library's docstrings, and uses them to create API documentation for the library in a variety of formats. Epydoc addresses all of these limitations:

- Docstrings can be written in a variety of markup languages, including reStructuredText and Javadoc. These markup languages allow docstrings to include non-plaintext content, such as tables, symbols, and images.
- Epydoc's HTML output makes API documentation easy to navigate.
- Once the documentation has been converted to HTML or PDF, it can be indexed and searched using existing tools.
- Epydoc uses special markup "fields" to let a user document individual variables.
- Epydoc provides both explicit and automatic methods for documentation inheritance.



Modules to document:
epydoc.imports
epydoc.latex
epydoc.man
epydoc.markup
epydoc.markup.epytext
epydoc.markup.javadoc
epydoc.markup.plaintext
epydoc.markup.restructuredtext
epydoc.objdoc
epydoc.test
epydoc.test.epytext
epydoc.uid

Add:

Progress:

□ Show Messages ■ Show Warnings ■ Show Errors

Project Options
Project Name: Epydoc
Project URL: http://epydoc.sourceforge.net/
Output Directory: html/api/
■ Generate a frame–based table of contents
■ Generate documentation for private objects
□ List imported classes and functions
Default Docformat: ◆ Epytext ◇ ReStructuredText ◇ Plaintext ◇ Javadoc
Inheritance Style: ◆ Grouped ◇ Listed ◇ Included

Help File
◆ Default
◇ Select File

CSS Stylesheets

Public	Private		
◆	◇	default	Default stylesheet (=white)
◇	◇	black	White on black, with blue highlights
◇	◆	blue	Black on steel blue
◇	◇	grayscale	Grayscale black on white
◇	◇	green	Black on green
◇	◇	none	An empty stylesheet
◇	◇	white	Black on white, with blue highlights
◇	◇	Select File	

Figure 1: The Epydoc GUI

2 Using Epydoc

2.1 The Command-Line Interface

To generate the HTML documentation for a library, simply run `epydoc` with a list of packages or modules in the library:

```
% epydoc ~/programming/epydoc/
Importing 20 modules.
  [....................]
Building API documentation for 20 modules.
  [....................]
Writing HTML docs (108 files) to 'html/'.
  [  0%] ...............................
  [ 56%] .....................
```

A variety of flags can be used to customize the output and change the output format. Run `epydoc --help` for a brief list, or see the `epydoc(1)` manpage for more complete information.

2.2 The Graphical Interface

Epydoc also provides a graphical interface (Figure 1) for users who prefer not to use command-line interfaces (e.g., Windows users). Currently, the graphical interface only supports HTML output; but we plan to add support for other output formats in the future.

3 Epydoc Output

Currently, epydoc supports two basic output formats: HTML and LaTeX; and three output formats derived from the LaTeX output: PDF, PS, and DVI.

Module Page Contents
Module description
Module metadata (author, etc.)
Sub-modules
Class summary
Exception summary
Function summary
Variable summary
Function details
Variable details

Figure 2: HTML Output – Module Page Contents. This table lists the sections that is included in a module's documentation page. All sections are omitted when empty.

3.1 HTML Output

Epydoc's HTML output is based on Javadoc and Doxygen, and should be familiar to anyone who has used those tools [3, 1].

3.1.1 Object Documentation Pages

A separate page is used to document each module and class. Each page begins with a general description taken from the module or class's docstring; and is followed by a description of each contained object. Figures (2) and (3) list the sections that can be included on each documentation page. The "summary" sections contain tables that provide a brief description of each object, and a link to its detailed description. The "details" selections contain full descriptions of each object. Figures (4), (5), and (6) list the contents of each entry in the details sections.

3.1.2 Navigation

In addition to the links within the documentation pages, Epydoc provides two tools for navigating the API documentation: a frames-based table of contents and a navigation bar.

The table of contents is shown on the left of Figure (7). It consists of two frames on the left of the page that can be used to quickly navigate to any object's documentation. The *project contents frame* contains a list of all packages and modules that are defined by the project. Clicking on an entry will display its contents in the module contents frame. Clicking on a special entry, labeled "Everything," will display the contents of the entire project. The *module contents frame* contains a list of every submodule, class, type, exception, function, and variable defined by a module

Class Page Contents
Base Tree
Known Subclasses
Class description
Class metadata (author, etc.)
Method summary
Property summary
Instance variable summary
Class variable summary
Method details
Property details
Instance variable details
Class variable details

Figure 3: HTML Output – Class Page Contents. This table lists the sections that is included in a class's documentation page. All sections are omitted when empty.

Function Details Contents
Function description
Parameter types and descriptions
Return value type and description
Exceptions raised
Function metadata (author, etc.)

Figure 4: HTML Output – Function Details Contents. This table lists the information that is included in a function or method's entry in a details section. All sections are omitted when empty.

Variable Details Contents
Variable description
Type
Value

Figure 5: HTML Output – Variable Details Contents. This table lists the information that is included in a variable or method's entry in a details section. All sections are omitted when empty.

Property Details Contents
Property description
Accessor methods

Figure 6: HTML Output – Property Details Contents. This table lists the information that is included in a property or method's entry in a details section. All sections are omitted when empty.

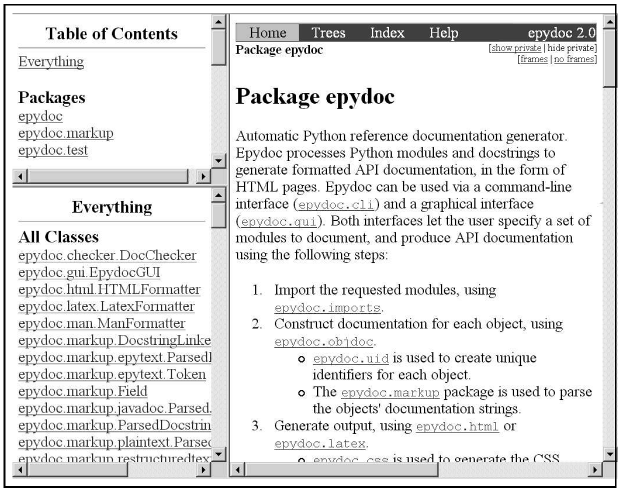


Table of Contents

Everything

Packages

epydoc
epydoc.markup
epydoc.test

Everything

All Classes

epydoc.checker.DocChecker
epydoc.gui.EpydocGUI
epydoc.html.HTMLFormatter
epydoc.latex.LatexFormatter
epydoc.man.ManFormatter
epydoc.markup.DocstringLinke
epydoc.markup.epytext.ParsedI
epydoc.markup.epytext.Token
epydoc.markup.Field
epydoc.markup.javadoc.Parsed.
epydoc.markup.ParsedDocstrin
epydoc.markup.plaintext.Parsec
epydoc.markup.restructuredtex

Package epydoc

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 to document, and produce API documentation using the following steps:

1. Import the requested modules, using `epydoc.imports`.
2. Construct documentation for each object, using `epydoc.objdoc`.
   - `epydoc.uid` is used to create unique identifiers for each object.
   - The `epydoc.markup` package is used to parse the objects' documentation strings.
3. Generate output, using `epydoc.html` or `epydoc.latex`.
   - `epydoc.css` is used to generate the CSS

Figure 7: Epydoc HTML Output

or package. Clicking on an entry will display its documentation in the API documentation frame. Clicking on the name of the module, at the top of the frame, will display the documentation for the module itself.

The navigation bar is shown at the top of the object documentation page in Figure (7). It provides quick links to all top-level pages and a "bread-crumb trail" of pointers to containing objects. It also includes a toggle which can be used to hide and show private objects[1]; and a toggle which can be used to turn the frames-based table of contents on or off.

3.1.3 Other Pages

The *Trees* page, available from the navigation bar, displays the library's module and class hierarchies. The *Index* page provides a variety of indices, including an identifier index; a todo index; and a definition index (for definitions explicitly tagged by markup). The *Help* page provides a quick tutorial describing how to use Epydoc's HTML output.

3.2 LaTeX Output

Epydoc can generate LaTeX output, which can then be automatically converted into PDF or PS[2]. The LaTeX output contains a single chapter for each package or module in the library. Classes, functions, and variables are included as sections with their containing modules' chapters. Figure 8 lists the sections that can be included in each module's chapter.

[1] In Python, private objects are defined as objects whose name starts with an underscore, but do not end with an underscore. For example, `_coconut` and `__log` are private names; but `__init__` is not.

[2] assuming that `latex`, `dvips`, and `ps2pdf` are installed

Module Chapter Contents
Module description
Module metadata (author, etc.)
Table of sub-modules
Function details
Variable details
Class sections • Base tree • Class summary • Method details • Property details • Instance variable details • Class variable details

Figure 8: LaTeX Output – Module Chapter Contents. This table lists the sections that is included in the chapter documenting a module. All sections are omitted when empty.

When the LaTeX output is generated, each module's documentation is written to a separate file. This makes it easy to include one or more module's API documentation as a chapter or section in other LaTeX documents (e.g., as an appendix to reference documentation). If you want to include API documentation for select classes, you can use the `--separate-classes` switch to tell epydoc to write each class's documentation to a separate file.

3.3 Manpage Output

We are currently working on adding manpage-style output. These manpages could be viewed interactively (similarly to pydoc) or written to manpage files (similarly to Tk's API manpages).

4 Docstring Markup

By using a markup language to write docstrings, programmers can create API documentation that is easier to read. For example, the programmer can use lists, tables, symbols, and images to document their code. Furthermore, paragraphs can be re-wrapped for display on a variety of display sizes (ranging from large monitors to small PDAs).

4.1 Markup Languages

Epydoc currently supports three markup languages for docstrings (in addition to plaintext):

- **Epytext**, a lightweight markup language that's easy to write and to understand. [2]
- **ReStructuredText**, an ”easy-to-read, what-you-see-is-what-you-get plaintext markup syntax.” It is more powerful than epytext (e.g., it includes markup for tables and footnotes); but it is also more complex, and sometimes harder to read. [4]
- **Javadoc**, a documentation markup language that was developed for Java. It consists of HTML, augmented by a set of special tagged fields. [3]

The markup language used in a module's docstrings is specified by the `__docformat__` variable, which should contain the name of a markup language, optionally followed by a language code (such as `en` for English). Conventionally, the `__docformat__` variable definition immediately follows the module's docstring.

4.2 Fields

Using a markup language to write docstrings allows us to write specialized *fields* that describe specific properties of a documented object. For example, fields can be used to define the parameters and return value of a function; the instance variables of a class; and the author of a module. Each field consists of a tag, an optional argument, and a body. The next page contains a list of the fields currently supported by epydoc. (All fields are shown in epytext markup; other markup languages have different ways to mark fields.)

A library writer can also define new information fields, using the `deffield` field or the special module-level `__extra_epydoc_fields__`.

Functions and Methods

`@param` *p* : ...	A description of the parameter *p*
`@type` *p* : ...	The expected type for *p* .
`@return:` ...	The return value.
`@rtype:` ...	The type of the return value.
`@kwparam` *p* : ...	A description of the keyword parameter *p* .
`@raise` *e* : ...	A description of the circumstances under which exception *e* is raised.

Variables

`@ivar` *v* : ...	A description of the instance variable *v* .
`@cvar` *v* : ...	A description of the class variable *v* .
`@var` *v* : ...	A description of the module variable *v* .
`@type` *v* : ...	The type of the variable *v* .

Content Operations

`@group` *g* : $c_1, \ldots, c_n$	Organizes a set of related objects into a group. *g* is the name of the group; and $c_1, \ldots, c_n$ are the names of the objects in the group.
`@undocumented` $c_1, \ldots, c_n$	Specifies a list of objects that should not be mentioned in the API documentation.

Summarization

`@summary:` ...	A summary description for an object. This description overrides the default summary (which is constructed from the first sentence of the object's description).
`@include:` *o*	Copy the contents of object *o*'s docstring into this docstring.

Notes and Warnings

`@warning:` ...	A warning about an object.
`@bug:` ...	A description of a bug.
`@note:` ...	A note about an object.
`@attention:` ...	An important note.

Related Topics

`@see:` ...	A description of a related topic, often including a cross-reference link.

Status

`@version:` ...	The version of an object.
`@todo:` ...	A planned change to an object.
`@depreciated:` ...	Indicates that an object is depreciated. The body of the field describe the reason why the object is depreciated.
`@since:` ...	The date or version when an object was first introduced.
`@status:` ...	The current status of an object.

Formal Conditions

`@requires:` ...	A requirement for using an object.
`@precondition:` ...	A condition that must be true before an object is used.
`@postcondition:` .	A condition that is guaranteed to be true after an object is used.
`@invariant:` ...	A condition which should always be true for an object.

Bibliographic Information

`@author:` ...	The author(s) of an object. Multiple `author` fields may be used if an object has multiple authors.
`@organization:` ...	The organization that created or maintains an object.
`@copyright:` ...	Copyright information about an object.
`@license:` ...	Licensing information about an object.
`@contact:` ...	Contact information for the author or maintainer of a module, class, function, or method. Multiple `contact` fields may be used if an object has multiple contacts.

5 Design Issues

5.1 Parsing vs Inspection

Epydoc primarily uses inspection to extract information about the libraries it documents. However, inspection has some significant limitations:

- Some information is not available via inspection. For example, Python does not keep track of what module a function was defined in; which variables were imported; and the class where nested classes are defined.
- Variables do not have docstrings.
- Some libraries use advanced ("magic") techniques to manipulate import mechanisms; and these techniques may interfere with inspection.

One alternative is to skip inspection altogether, and extract documentation from parsing. This is the technique used by most non-python API documentation extraction tools. However, Python presents some unique challenges to parsing:

- Many important modules are not written in Python.
- Python is an extremely flexible language, allowing the user to manipulate almost every aspect of execution. As a result, it is extremely difficult to robustly determine the set of objects that are visible in Python from a simple parse tree.

Epydoc has therefore elected to use a hybrid approach: inspection forms the basis for documentation; but parsing is used to overcome the limitations of inspection, where necessary.

References

[1] The doxygen homepage. `http://www.doxygen.org/`.

[2] The epytext markup language. `http://epydoc.sourceforge.net/epytext.html`.

[3] The javadoc homepage. `http://java.sun.com/j2se/javadoc/`.

[4] The restructuredtext homepage. `http://docutils.sourceforge.net/rst.html`.