

8th November 2002

PEP: 256 Title: Docstring Processing System Framework Version: \$Revision: 1.6 \$ Last-Modified: \$Date: 2002/10/24 23:32:54 \$ Author: David Goodger <goodger@users.sourceforge.net> Discussions-To: <doc-sig@python.org> Status: Draft Type: Standards Track Content-Type: text/x-rst Created: 01-Jun-2001 Post-History: 13-Jun-2001

Abstract

Python lends itself to inline documentation. With its built-in docstring syntax, a limited form of **Literate Programming** is easy to do in Python. However, there are no satisfactory standard tools for extracting and processing Python docstrings. The lack of a standard toolset is a significant gap in Python's infrastructure; this PEP aims to fill the gap.

The issues surrounding docstring processing have been contentious and difficult to resolve. This PEP proposes a generic Docstring Processing System (DPS) framework, which separates out the components (program and conceptual), enabling the resolution of individual issues either through consensus (one solution) or through divergence (many). It promotes standard interfaces which will allow a variety of plug-in components (input context readers, markup parsers, and output format writers) to be used.

The concepts of a DPS framework are presented independently of implementation details.

Road Map to the Doctring PEPs

There are many aspects to docstring processing. The “Docstring PEPs” have broken up the issues in order to deal with each of them in isolation, or as close as possible. The individual aspects and associated PEPs are as follows:

- Docstring syntax. PEP 287, “reStructuredText Docstring Format” [1], proposes a syntax for Python docstrings, PEPs, and other uses.
- Docstring semantics consist of at least two aspects:
 - Conventions: the high-level structure of docstrings. Dealt with in PEP 257, “Docstring Conventions” [2].
 - Methodology: rules for the informational content of docstrings. Not addressed.
- Processing mechanisms. This PEP (PEP 256) outlines the high-level issues and specification of an abstract docstring processing system (DPS). PEP 258, “Docutils Design Specification” [3], is an overview of the design and implementation of one DPS under development.
- Output styles: developers want the documentation generated from their source code to look good, and there are many different ideas about what that means. PEP 258 touches on “Stylist Transforms”. This aspect of docstring processing has yet to be fully explored.

By separating out the issues, we can form consensus more easily (smaller fights ;-), and accept divergence more readily.

Rationale

There are standard inline documentation systems for some other languages. For example, Perl has **POD** (“Plain Old Documentation”) and Java has **Javadoc**, but neither of these mesh with the Pythonic way. POD syntax is very explicit, but takes after Perl in terms of readability. Javadoc is HTML-centric; except for “`@field`” tags, raw HTML is used for markup. There are also general tools such as **Autoduck** and **Web** (Tangle & Weave), useful for multiple languages.

There have been many attempts to write auto-documentation systems for Python (not an exhaustive list):

- Marc-Andre Lemburg’s **doc.py**
- Daniel Larsson’s **pythondoc** & **gendoc**
- Doug Hellmann’s **HappyDoc**
- Laurence Tratt’s **Crystal**
- Ka-Ping Yee’s **pydoc** (pydoc.py is now part of the Python standard library; see below)
- Tony Ibbs’ **docutils** (Tony has donated this name to the **Docutils project**)
- Edward Loper’s **STminus** formalization and related efforts

These systems, each with different goals, have had varying degrees of success. A problem with many of the above systems was over-ambition combined with inflexibility. They provided a self-contained set of components: a docstring extraction system, a markup parser, an internal processing system and one or more output format writers with a fixed style. Inevitably, one or more aspects of each system had serious shortcomings, and they were not easily extended or modified, preventing them from being adopted as standard tools.

It has become clear (to this author, at least) that the “all or nothing” approach cannot succeed, since no monolithic self-contained system could possibly be agreed upon by all interested parties. A modular component approach designed for extension, where components may be multiply implemented, may be the only chance for success. Standard inter-component APIs will make the DPS components comprehensible without requiring detailed knowledge of the whole, lowering the barrier for contributions, and ultimately resulting in a rich and varied system.

Each of the components of a docstring processing system should be developed independently. A “best of breed” system should be chosen, either merged from existing systems, and/or developed anew. This system should be included in Python’s standard library.

PyDoc & Other Existing Systems

PyDoc became part of the Python standard library as of release 2.1. It extracts and displays docstrings from within the Python interactive interpreter, from the shell command line, and from a GUI window into a web browser (HTML). Although a very useful tool, PyDoc has several deficiencies, including:

- In the case of the GUI/HTML, except for some heuristic hyperlinking of identifier names, no formatting of the docstrings is done. They are presented within `<p><small><tt>` tags to avoid unwanted line wrapping. Unfortunately, the result is not attractive.
- PyDoc extracts docstrings and structural information (class identifiers, method signatures, etc.) from imported module objects. There are security issues involved with importing untrusted code. Also, information from the source is lost when importing, such as comments, “additional docstrings” (string literals in non-docstring contexts; see PEP 258 [3]), and the order of definitions.

The functionality proposed in this PEP could be added to or used by PyDoc when serving HTML pages. The proposed docstring processing system’s functionality is much more than PyDoc needs in its current form. Either an independent tool will be developed (which PyDoc may or may not use), or PyDoc could be expanded to encompass this functionality and *become* the docstring processing system (or one such system). That decision is beyond the scope of this PEP.

Similarly for other existing docstring processing systems, their authors may or may not choose compatibility with this framework. However, if this framework is accepted and adopted as the Python standard, compatibility will become an important consideration in these systems’ future.

Specification

The docstring processing system framework is broken up as follows:

1. Docstring conventions. Documents issues such as:
 - What should be documented where.
 - First line is a one-line synopsis.PEP 257 [2] documents some of these issues.
2. Docstring processing system design specification. Documents issues such as:
 - High-level spec: what a DPS does.
 - Command-line interface for executable script.
 - System Python API.
 - Docstring extraction rules.
 - Readers, which encapsulate the input context.
 - Parsers.
 - Document tree: the intermediate internal data structure. The output of the Parser and Reader, and the input to the Writer all share the same data structure.
 - Transforms, which modify the document tree.
 - Writers for output formats.
 - Distributors, which handle output management (one file, many files, or objects in memory).

These issues are applicable to any docstring processing system implementation. PEP 258 [3] documents these issues.

3. Docstring processing system implementation.
4. Input markup specifications: docstring syntax. PEP 287 [1] proposes a standard syntax.
5. Input parser implementations.
6. Input context readers (“modes”: Python source code, PEP, standalone text file, email, etc.) and implementations.
7. Stylists: certain input context readers may have associated stylists which allow for a variety of output document styles.
8. Output formats (HTML, XML, TeX, DocBook, info, etc.) and writer implementations.

Components 1, 2/3/5, and 4 are the subject of individual companion PEPs. If there is another implementation of the framework or syntax/parser, additional PEPs may be required. Multiple implementations of each of components 6 and 7 will be required; the PEP mechanism may be overkill for these components.

Project Web Site

A SourceForge project has been set up for this work at <http://docutils.sourceforge.net/>.

References and Footnotes

Copyright

This document has been placed in the public domain.

[1] PEP 287, reStructuredText Docstring Format, Goodger (<http://www.python.org/peps/pep-0287.html>)

Acknowledgements

This document borrows ideas from the archives of the [Python Doc-SIG](#). Thanks to all members past & present.

[2] PEP 257, Docstring Conventions, Goodger, Van Rossum (<http://www.python.org/peps/pep-0257.html>)

[3] PEP 258, Docutils Design Specification, Goodger (<http://www.python.org/peps/pep-0258.html>)