

# Source Code Documentation as a Live User Manual

PTLogica

## Abstract

Source code documentation is the manual and guide that helps in understanding and correctly utilizing the source code. Complementing coding standards and naming conventions with natural language results in enhanced clarity for the designer, a handbook for the client and a guide for the maintainer of the code, facilitating code reuse and refactoring.

source for detailed design and interface descriptions.

A solution to this problem is to employ a tool to extract meaningful source code comments to create external documentation. This tool may be an integral part of the software build process by automating the documentation effort. Nevertheless, as with code, this process is truly effective only if designers can see and generate the documentation themselves, allowing them to perfect and debug it.

## 1 A Live Document

While it makes sense to have independent high-level design documentation, it is not reasonable to detach the detailed design from the code, because it is well-known that independent documentation is difficult to maintain and not updated as frequently.

Designers in many ISO-approved companies simply copy and paste their methods or functions comments into a detailed design document so they can deliver the required documentation to conform to their software development process. While this might be satisfactory to auditors and customers, it is not an acceptable solution, because the deliverable in this case is a hard-to-maintain snapshot of an evolving entity.

The natural tendency is to reduce paper document dependency in the design process and to use the code and its comments as the direct

## 2 Commenting Code

Effective and practical documentation captures the intended use of modules, classes and routines in a user-manual-like form. In order to produce highly maintainable documentation, practice indicates that comments should not focus on explaining how the code works, but rather on how the code is to be used.

If comments are used to explain how the code works, any alteration to the code will render the documentation obsolete, resulting in duplication of effort and inaccurate documentation.

Plain English at the top of the file and each code construct should be used to describe its unique purpose and objective. Inline, detailed comments should be avoided because they cannot express the mechanics of the code more accurately than the code itself.

By avoiding the description of “anything between the brackets”, the formal documentation reaches the non-redundant level of abstraction needed to streamline the authorship, usage and maintenance of the code.

### 3 Documentation Tools

When considering a source code documentation tool, it is useful to examine the following aspects:

1. Usability
2. Target media
3. Documentation structure
4. Level of detail
5. Comment extraction capabilities
6. Languages supported
7. Inline formatting and style elements
8. Source code readability

#### 3.1 Usability

Source code documentation tools are meant to help designers to document their code clearly and produce external documentation from it without the tools getting in the way. They should be unobtrusive and simple enough to actually get used while coding. Designers are likely to adopt a simple and familiar framework that provides rapid feedback.

#### 3.2 Target media

HTML is the appropriate medium for a live document. It is an open standard that can be easily generated on the fly and rendered on any web

browser. It also allows depiction of the structure and layout of a page with sufficient precision, particularly when combined with cascade style sheets.

Printed documentation is useful only on a selective basis. Generating printed material for 50, 100, 500 or more files simply makes no sense due to the high volume of paper. If however you can print documentation for individual files or portions of them (as a reference), then printed documentation serves its purpose.

#### 3.3 Documentation structure

The book paradigm is a proven approach to determine the documentation structure. As a minimum, the generated documentation should contain an index to pages or chapters. Each chapter should contain a title, introduction or foreword, table of contents and sections.

#### 3.4 Level of detail

The level of detail in the documentation is an important element. Too much detail renders the documentation ineffective; rather the tool should generate documentation as intended by the author and not as produced by an exhaustive parsing tool.

#### 3.5 Comment extraction capabilities

Documentation tools should be able to extract source code comments regardless of the commenting style used by the author. If conventions are necessary, they should not be excessively intrusive.

### 3.6 Languages supported

Source code documentation tools supporting multiple programming languages are preferable to those focusing on a specific programming language. The result is consistent documentation for your solution and an improved return on your investment and training.

### 3.7 Inline formatting and style elements

Because a document is being created from essentially a loosely formatted source, special elements such as tags or markup are generally required to determine the layout, structure and style of the documentation. A heavily typed and precise XML/SGML vocabulary is certainly a serious burden for the software designer and ultimately detrimental to the clarity of the source. The preferable alternative is using only a limited number of tags. The ideal system should not require noticeable formatting elements.

### 3.8 Source code readability

An external set of documents is not the only expected deliverable in the source code documentation process; clear source code comments should support all the generated documentation as well. Even if no external documentation is generated at all, a source code documentation tool returns its original investment if a software solution is documented at the source code level in a consistent and human-readable way.

## 4 TwinText

TwinText™ is a source code documentation tool that obeys the book paradigm by generating

the required elements automatically with configurable format and style. In addition, the tool properly mixes selected comment sections with precise code analysis to generate usable documentation with the right level of detail. Virtually all programming languages and commenting styles are supported.

For most practical situations, the inline formatting elements required to define the structure of the documentation are those used naturally when documenting code. Also time honored ASCII conventions are used to enhance style and formatting. TwinText does not force you to commit to noticeable formatting conventions, yet you can still deliver documentation based on your code.

When processing individual files or projects, the associated page is promptly displayed in the embedded web browser, giving useful feedback to the author who can then proceed to perfect the results if necessary.

Printed documentation can be generated on a selective basis by right-clicking the appropriate document window.

TwinText's affordable licensing means that you can deploy it on a per-designer basis, resulting in accurate documentation for each functional area or module in your solution.

With TwinText you are not only delivering outstanding external documentation, but also clear and readable source code comments.

TwinText sensibly complies with the different aspects discussed above, embedding the elements necessary to make the documentation process effective, with a minimal learning curve for the software designer.