Software documentation systems

Basic introduction to various user-oriented and developer-oriented software documentation systems.

Ondrej Holotnak

Ondrej Jombik

Software documentation systems: Basic introduction to various user-oriented and developer-oriented software documentation systems.

by Ondrej Holotnak and Ondrej Jombik

Table of Contents

List of Examples

Chapter 1. Introduction

Almost every software developer has sometime in his life been confronted with the problem of documenting his project. This, one would say banal matter, consists of more problems than one would think. Software that does not include quality detailed documentation is heading for downfall. In our document we will try to illustrate some of the possibilities for quick and effective documentation of software product.

The basic factor to think about is the target group. Common software user is interested in different type of information than application developer. This needs to be taken into consideration. One needs to adapt the form and content of documentation. That is why distinguish two different types of documentation:

1. User-oriented documentation

   Common user is usually not interested in software architecture nor its inner implementation. He or she does not care how things are done, what means and programming languages were used in the process of development. What interests him/her the most is how the program runs, how is it used and also in what environment and under which operation system it can be started. That is why the emphasis should be put on detailed, but mainly understandable explanation of all functions and possibilities of the program.

   It is better if someone else than the author of the program writes the user-oriented documentation. He may consider many things to be absolutely clear while they may not be clear at all to the common user. It is appropriate if the user-oriented documentation includes many examples and is written in awareness that it will be read by person who does not understand programming at all.

2. Developer-oriented documentation

   A developer working on a project needs to have documented all the inner technical matters of the software, such as class inheritance hierarchy, data structure description as well as its attributes, list of source files, global identifiers etc. The description of the correct use of corresponding development tools used for the program creation should also be included.

   The basic purpose of existence of developer-oriented documentation is the maintenance and extensibility of the software product. It is an absolute must to document all the nonstandard program constructions as well as a detailed description of the process of debuging and synchronizing the program.

It is now clear that these two types of documentation are absolutely different. It is not possible that the documentation of the first type replaces the second or the other way around. The developer-oriented documentation should be written along with the program creation. If the inner implementation of structure changes, the corresponding part of documentation is changed as well. It is reasonable to start

writing the user-oriented documentation as soon as possible, but this should not be before the program reaches its basic functionality.

Both types of documentation are written by different people and the tools for its creation and maintenance are adapted correspondingly. Their use is illustrated in the following chapters of this document.

Chapter 2. User-oriented software documentation systems

The most famous and the most used system for user-oriented documentation creation and maintaining is DocBook (http://www.oasis-open.org/docbook/). This open tool build upon open technologies provides a system for writing structured documents using SGML or XML. It is particularly well-suited to books and papers about computer hardware and software, though it is by no means limited to them. DocBook is an SGML document type definition (DTD).

Many well-known companies, such as TcX (the developers of database system MySQL) or TrollTech (the developers of graphical library QT2) use their own tools for their product documentation. Not pointing out other factors, the result of using own documentation system could be fairly hard and complicated improvement and extensibility of the documentation by different contributors. This is the reason why the use of open technologies in process of creating documentation is highly desirable and productive.

DocBook document is written using the so-called markup languages, such as SGML and XML. DocBook exists in both of these versions. Very much like in the HTML documents, individual parts or elements are marked by special tags. While in a HTML document the tags define the design of the document (change of color, font, etc.), in DocBook there is no tag that manipulates directly with design. The document is structured *logically*, which means that the parts of the same meaning are tagged (filenames, variables, products, persons, etc.). The resultant design can be specified using the DSSSL (http://www.jclark.com/dsssl/) (Document Style Semantics and Specification Language), but is not essential for the common user.

The differences between the XML and SGML versions of DocBook are minimal. Apart from different header of the document and different implementation of perhaps two tags are both versions identical. The main difference is in the processing of the document. The SGML version uses Jade Wrapper (jw) while the XML version uses tools based on the libxml2 library. In our examples, we will use the SGML version.

The basic document looks like this:

Example 2-1. The basic DocBook SGML document

```
<!DOCTYPE book PUBLIC "-//OASIS//DTD DocBook V3.1//EN" [ ]>
<book lang="en">
<bookinfo>
<date>2002-12-16</date>
<title>My first DocBook document</title>
<subtitle>with this subtitle</subtitle>
</bookinfo>

<chapter>
```

```
<title>Title of the first chapter</title>

<para>Text of the first chapter.</para>

</chapter>
</book>
```

DocBook document (<book>) is divided into chapters (<chapter>), sections in the chapters (<sect1>) and subsections (<sect2>). Subsections can have their own subsections (<sect3>, <sect4>). Chapters can be grouped with the <part> tag. Similarly, more books can be grouped into one compact unit using the <set> tag.

Example 2-2. The basic division of the DocBook SGML document

```
<chapter><title>Title of the first chapter</title>
<sect1><title>First section in the first chapter</title>>
<sect2><title>First subsection in the fist section</title>
<para>Text. Text. Text. Text.</para>
</sect2>
<sect2><title>Second subsection in the first section</title>
</sect2>
</sect1>
<sect1 ...>
<para> ... </para>
</sect1>
</chapter>
```

As it has already been mentioned, a DocBook document has its logical structure according to the meaning. Tags are used to mark expressions or words of the same type. Therefore things like filenames, variable names, constant names, function names, commands, products are tagged accordingly, as well as e-mail addresses or links to web pages. It is also possible to create lists, tables, insert pictures, and so on.

Example 2-3. Illustration of some DocBook SGML tags

```
<filename>/etc/passwd</filename>
<filename class=headerfile>stdio.h</filename>
<varname>$selected_user</varname>
<constant>BUFFER_SIZE</constant>
<function>proctable_init()</function>
<command>grep</command>
<application>jade</application>
<productname>DocBook</productname>
<email>platon@pobox.sk</email>
<ulink url="http://www.platon.sk/">Platon SDG</>
```

You can see how simple SGML tags can be shortened by using the </> tag on the last example.

One of the main advantages of DocBook documentation is the big number of possible output document formats. The most common are HTML, RTF, PDF, PS, TeX, pure text or manual pages. The generation is done by the jade program (we expect to generate `myfile.html` file from `myfile.sgml` file).

```
$ jade -d /usr/lib/sgml/stylesheets/dbtohtml.dsl -t sgml myfile.sgml > myfile.htm
```

It is not necessary to use the jade parser directly. Some shell scripts for generating particular types of output files already exist. Those include db2html, db2pdf, docbook2tex, or docbook2txt. That means that to generate the HTML documentation you just need to use the following command.

```
$ db2html myfile.sgml
```

The design of the generated HTML document can be simply changed by using CSS (Cascading Style Sheets).

Chapter 3. Developer-oriented software documentation systems

There are many tools for generating the developer-oriented documentation according to the programming language the application is written in. For C/C++ it is Doxygen (http://www.doxygen.org/), which was evolved from Doc++ (http://docpp.sourceforge.net/). For PHP there are several documentation tools: phpDoc (http://www.phpdoc.de/), phpDocumentor (http://www.phpdoc.org/), PearDoc and PearDoc2 (http://pear.php.net/manual/). The last named probably has the best future, because it is being actively developed and will probably become the main documentation tool for PHP. Finally, for Java it is the well-known JavaDoc (http://java.sun.com/j2se/javadoc/).

The developer-oriented documentation is written straight in the source code of the program in the means of special comments. All entities of the programming language are documented: classes and their methods and attributes, global and local functions and variables, constants, data types such as structures, enumerations and others. Documentation system distinguishes the type of information according to special marks and places it in the resulting document accordingly.

The main advantages of this approach are complexity and timesaving. Thanks to the fact that the documentation is written into the source code files, it is not necessary to work with both source code files and documentation file simultaneously. As already mentioned, it is appropriate to write the developer-oriented documentation when working on the code of the application. So right after adding a new method, one should write a simple comment that would be used to inform about the functionality of the method right in the source code, and will of course be used to generate the developer-oriented documentation. It is still possible to use classical comments in the source code. They will not be incorporated into the documentation.

What needs to be said is that developer-oriented documentation systems are able to derive a lot of information directly from the source code, because they usually contain analyzers of the programming language. For example, you do not need to write the name of the function, method or class in its comment, because it can be extract from its declaration. This is the main reason why there are different developer-oriented documentation systems from every programming language, even though their use is basically the same.

In the following examples, you can see the documentation of a simple class in three different programming languages (C++, PHP, Java) using their documentation systems (Doxygen, PearDoc, JavaDoc). The emphasis is put on the features that are the same for all of these systems. Let us start with the example for Doxygen.

Example 3-1. Documented C++ source code for Doxygen

```
/**
 * Example class
```

```
 *
 * More elaborate class description.
 */
class Example
{
public:

/**
 * Example object constructor
 *
 * More elaborate description of the constructor.
 *
 * @param    bar    a bar constructor parameter
 * @param    foo    a foo constructor parameter
 * @return   void
 */
Example(int bar = 0, const char *foo = "foo");

/**
 * Example object destructor
 *
 * More elaborate description of the destructor.
 */
~Example();
};
```

Example 3-2. Documented PHP source code for PearDoc

```
/**
 * Example class
 *
 * More elaborate class description.
 */
class Example
{
/**
 * Example object constructor
 *
 * More elaborate description of the constructor.
 *
 * @param    int      bar   a bar constructor parameter
 * @param    string   foo   a foo constructor parameter
 * @return   boolean        true if this example is right,
 *                          false otherwise
 */
function Example($bar = 0, $foo = 'foo')
{
// Constructor code goes here. Note that PHP does not
// have anything like header files with declarations.
return true;
}
}
```

Example 3-3. Documented Java source code for JavaDoc

```
/**
 * Example class
 *
 * More elaborate class description.
 */
class Example
{
/**
 * Example object constructor
 *
 * More elaborate description of the constructor.
 *
 * @param    bar    a bar constructor parameter
 * @param    foo    a foo constructor parameter
 * @return   void
 */
Example(int bar = 0, const char *foo = "foo")
{
// Constructor code goes here
}
}
```

It can be seen from the examples that the special comments begin with the sequence /**. Classical /* comments are ignored. @param marks were used to define the input function parameter and @return to define the return value of the function. The source code of C++ already includes the information about the data type of the input parameter, therefore it is not included in the comment. Doxygen can find out the information by itself. On the other hand, the PHP source code does not include this information, therefore it needs to be included in the documentation comment.

Some of the other marks used for documenting functions or methods include (apart from @param and @return) @access representing the accessibility of the method (public, private, ...) and @retval used for the description of return value to parameter of the function defined by reference (address). Other universally usable marks are:

```
@author      -- author of the entity (class, method, function)
@version     -- version of the entity
@date        -- date of creation
@since       -- minimum version having some functionality
@deprecated  -- old and now unsupported functionality
@see         -- reference to the corresponding entities
@todo        -- list of unfinished features
```

Example 3-4. Use of other documentation marks

```
/**
```

```
 * Extensive Example class
 *
 * This extensive class does some extensive job.
 *
 * @author     Ondrej Jombik <nepto@pobox.sk>
 * @author     Ondrej Holotnak <beast@host.sk>
 * @version    1.2
 * @date       2004-03-26
 */
class Extensive_Example
{
/**
 * Normal method doing something
 *
 * @param   some    some integer argument
 * @retval  thing   allocated string filled with thing
 * @return          test result
 * @author  Someone External <someone@example.com>
 * @see     Test()
 * @see     ~Test()
 * @todo    make this method something doing
 */
int doSomething(int some, char **thing);
};
```

Similarly to the user-oriented documentation, there is the opportunity to generate the final documentation in several formats. The Doxygen application supports the following formats: HTML, LaTeX, RTF, XML and manual pages. It is all configured using a well-defined configuration file `Doxyfile`. If you run the **doxygen** program with `-g` switch, the default configuration file will be written to disk. It includes a lot of comments, so its modification is simple and straightforward. The final document can be generated with the following command:

```
$ doxygen
```

All of the tools mentioned can be used in the same or similar way. They do not have to support all of the file formats though. But they all do include the opportunity to generate HTML output file, which is the most important and most widely used one.

Chapter 4. Other software documentation systems

There are other systems for software documentation. We cannot say whether it belongs to the user-oriented or developer-oriented group about neither one of them. Every one of them was created with a certain particular aim, which it has followed. These contain:

- PerlDoc
- man
- info

PerlDoc (http://www.perldoc.com/) is a documentation system used by the community around a script language Perl (http://www.perl.org/). The thing that helped its growth is the existence of CPAN (http://www.cpan.org/) - archive of reusable code for Perl. The documentation is written straight into the source code using special POD marks. Users then access it using Perl system script **perldoc**. There are many converters of PerlDoc documentation into different output formats, including HTML.

The documentation system of system manual pages (man pages) has been created in the times of creation of the operating system UNIX and is actively developed to these days. To format the output, we use the nroff program and pages are displayed with the **man** command. They come along on almost every UNIX or GNU/Linux system. Every man page is compressed and saved on the disk separately. There are tools for its conversion to other formats.

The result of the GNU movement (GNU is not UNIX) is info documentation, which is trying to substitute the man pages. This documentation system is not being used as much as its creators would expect due to its complicated control and user-unfriendly look. However, most of the GNU tools still includes more detailed documentation in its info pages than in corresponding man pages. The pages can be displayed with the **info** command.