

# Documentation Proposal

Author : David Rizzi

Date: June 2004

## Overview

An important aspect of any computer application or system is documentation. With good documentation, future developers and/or users of the system will be able to understand and interact with the system more rapidly.

This report discusses approaches to system documentation and particular methodologies for the implementation of the documentation. It offers a proposal for deciding what aspects of the system should be included in the documentation, and what form that documentation should take.

This report uses examples based upon a web application built on PHP and MySQL, but the concepts can be applied to other technologies / languages also.

## Documentation Sub Classes

Two types of documentation will be discussed.

1. **User** – This type of documentation is meant to be accessible to the user of the system. In this discussion, the user of the system could be the customer that utilizes the product or company internal users who perform administrative tasks. The focus here will be on the latter group although the concepts can also be applied to the former.
2. **Developer** – This type of documentation is meant to be accessible by developers of the system. In creating this type of documentation, it is assumed that the target developer audience is familiar with the technologies used to implement the system, but has little or no familiarity with the actual implementation or functionality of the system.

## USER DOCUMENTATION

Although user documentation is designed for the users of the system, this information will also be helpful to the developer who has little or no familiarity with the system. The user documentation will provide an overview of the system functions and a detailed accounting of how they are used.

**Note:** The term *user* in this context refers to internal company users who are performing administrative tasks. The ideas presented here could also apply to external users of the system (customers) although the way in which the documentation is presented and disseminated may be different.

The first thing user documentation will address is an overview of the system.

- 1) What the system is meant to do in a general sense
- 2) How to access the system
- 3) The specific areas of functionality that are accessible and their general purpose

An (abbreviated) example of the first requirement:

*The Blue Mountain administrative system allows the vendor to view and modify (where appropriate) all information related to products, product sales, and customer accounts.*

An (abbreviated) example of the second requirement:

*Using Internet Explorer 5.5 or above, browse to: **www.foo.com/admin**. At the login prompt, enter the name and password that was assigned to you. Name and password are case sensitive.*

The third area of the overview outlines the specific areas of the system's functionality along with a general description of what the specific functionality accomplishes. This normally would coincide with the menu options that are presented to the user. Example:

Menu	Sub Choices	Description
Retail Products	Add/Edit/View Products	View and/or modify the products that are available to customers. Change price, description; temporarily deactivate, etc.
	Add/Edit/View Groups	View and/or modify product groups, which are used to group together several products of a similar nature.
Customer	Add/Edit/View Customer	View and/or modify customer information; name, address, password, etc.
	Newsletter	Create and/or modify the newsletter templates that are used to create customized promotional email.
Weapons	Launch	Initialize countdown of rouge missiles targeted to randomly selected evil empires.
	Recall	Currently not implemented

At this point, the user has a general idea of what the system does, and how each available module is used. The next step in the user documentation is to provide a detailed description of each functionality. Example:

#### **Adding a new product**

*From the Retail Products menu, choose: **Add/Edit/View Products**. The product list for the first product group (see group information below for how to change the first group) appears. Click the **ADD NEW PRODUCT** icon in the upper right corner. The product information appears.*

The instructions continue in this manner, providing a detailed explanation of every screen and most of the fields. Some fields could be assumed to be self explanatory, but it is better to err on the side of a little too much information than not enough. Readers will skim.

This part of the user documentation can get very lengthy. In the example above, we have barely scratched the surface of what is available on the product information screen.

This part of the user documentation is task oriented. To the user reading the documentation, the answers to questions such as *how to*:

- *add a product*
- *edit a product*
- *delete a product*
- *change customer information*

- *view an order*
- *prepare a promotional email newsletter*
- *etc.*

will be found under the appropriate task heading.

Generally, tasks will be described in the natural order in which they will occur. For instance – assuming the user is starting from scratch – they will need to add a product before they can edit it. Therefore, the user documentation section dealing with the addition of a new product will go into detailed explanations of each entry involved (perhaps several sub screens and multiple entry fields) while the section dealing with the modification of an existing product need only outline the few steps required to navigate to the product editing screen itself. If however, there are some differences in the way entries are handled between an addition and a modification operation, those differences will need to be documented.

**Summary:** The user documentation serves as a user manual, allowing the user to understand the overall goals of the system and how to perform particular tasks. The user documentation is likely to be quite lengthy. A developer who is new to the project will find useful information within the user documentation overview section. Depending upon system complexity, the developer may or may not find the detailed section of the user documentation useful.

## DEVELOPER DOCUMENTATION

Of necessity, the developer documentation will be more complex and more technical than the user documentation. The developer documentation will also need to present an overview of the system but this overview will take a different form.

### Directory structure overview

A good place to start with a web application is an overview of how the files used in the application are organized within the filing system directory structure.

(formatting note: if you are reading this on screen, not a printout, try hiding table gridlines for the diagram below – or not. it's MS Word; mileage varies) // end ms bash

/		outside world main web root files. The PHP files here are tiny, 2 – 8 lines each. They call the template; the work is done in /content
admin		admin main scripts
	css	style sheets for admin only
	images	images for admin only
	inc	include files for admin only
	js	javascript files for admin only
content		the content – would need large discussion
css		style sheets for any part of the site
images		general site images
	help	images used exclusively for the user help
	product	images used by the product pages
	upload	uploaded image files, used when admin wants to leave a copy (perhaps in its original size) of an image
inc		include files for any part of the site
js		javascript files for any part of the site

The directory diagram helps the developer in reaching a general understanding of how the application is organized. If the application places all (or most) files in a single directory, this portion of the developer documentation isn't going to help much. From the perspective of a developer who is new to the project – and especially with a larger application consisting of dozens or hundreds of files – organizing the files and documenting their organization, is extremely useful.

### Flow charts and other diagrams

In some cases, it may be beneficial to construct a simple flow chart that illustrates the logic flow of the application or a portion of it. For instance, an e-commerce ordering process can be more easily understood by looking at an overview of the logic in the form of a flow chart. See Appendix A for a sample.

When more than a simple database is involved in the application, a data diagram that shows the relationships between the data entities can be extremely useful. See Appendix B for a sample.

### Individual files overview

Next, the individual files need to be documented using a broad overview that outlines their general functionality. This overview allows the newly inaugurated developer to understand how the files are used within the application without resorting to detailed explanation of how the functionality is implemented. Example:

Dir	File	Description
/admin	adminLogin.php	Handles logging in to the system. Control passes to this file whenever it is determined that the request is coming from an unauthenticated user.
	orderList.php	Shows a list of orders according to specified criteria
	404.php	File .htaccess establishes this file as the custom 404 handler
/admin/inc	appStart.inc	This file is included by all admin modules. It establishes the include path and specifies all of the other standard include files.
	appConfig.inc	Configuration items for the admin application

This portion of the developer documentation is meant to show what the various files are used for and not intended to document *how* the files achieve their goals. Even so, the above example uses extremely abbreviated descriptions; in the actual documentation, the file descriptions would need to be more explicit when describing the file's functionality.

### Individual files details

The details of how the files achieve their goals can either:

- be discussed in the external documentation – or
- be included within the source code files themselves

In the case of PHP files, since it is not compiled, comments internal to the source code increase execution time. However, the increase in execution time may or may not be significant, depending upon the size of the files, the number of comments, and the speed of the web server.

Comments within the source code offer the best readability since they are normally placed in proximity to the language constructs and statements. Comments however, should not be used in place of garbled code.

If optimizing for execution speed is a top priority, comments will need to be moved to external documentation. Since source code comments generally require the language constructs and statements they are describing to be nearby, this in effect means maintaining two copies of every source file: one stripped of comments that is used for execution, and another used for documentation.

This approach however, is not practical and would prove very difficult to maintain. And although it would be possible to employ an automated processing system which allowed developers to work on heavily commented source code files, and then published those files to the execution points after stripping all comments, this is an inelegant approach and would also not have a general applicability because it would not apply to a compiled language.

The best solution for providing functionality details of individual source code files is to use concise meaningful comments within the files themselves, and to utilize principles of self documenting code.

### **Self documentation**

Self documenting code will go a long way towards contributing to the new developer's understanding of the project. Self documenting code has many aspects including:

- Use of meaningful names
- Use of named constants
- Use of concise and narrow scope functions / methods
- Code formatting

and much more. These days most programming is done using object oriented languages. Object oriented programming by its nature lends itself well to the principle of self documentation. PHP supports OOP although currently not to the same degree as other languages.

When dealing with an existing code base that has not been developed using principles of self documentation and does not have much in the way of internal comments, the readability of the code is *greatly* decreased. In this situation, a developer new to the project will have a more difficult time getting up to speed. There are two solutions:

1. Refactor the code to be more inherently readable
2. Insert copious comments throughout the code

In the long run, the first choice is better. Either approach requires a thorough knowledge of the existing code. If the job of enhancing the code readability falls to a developer who is unfamiliar with the project, that person will need to spend (possibly) considerable time to gain an understanding of the logic and functionality involved.

### **Database documentation**

When an application uses a database, it is extremely helpful for the incoming developer to have database documentation as a reference source. The database documentation can take different forms but it needs to explain the purposes of, or define the following:

- Tables
- Fields
- Keys and constraints
- Relationships

Some information requires less explanation than others, and some database management systems have facility for documenting within the database itself.

The documentation of tables will briefly explain what each table is used for:

Table	Description
Customer	Holds the basic customer info such as name, address, etc.
CustomerHistory	Used to track changes to the customer's email address. They use their email address to login to the system.
metaLink	many to many mapping, each table row defines a unique meta tag occurrence by specifying a unique meta name, meta page, and meta content combination.

As shown, some tables require more explanation than others. Even though the purpose of some tables is likely to be apparent, it is a good idea to at least briefly describe them.

Similarly, the fields of each table need to be documented. Some fields will require only the smallest explanation and others will require more elaboration.

Keys and constraints are probably most easily documented using DDL.

```
CREATE TABLE customer (  
  CustomerNum int(11) NOT NULL auto_increment,  
  CustomerID varchar(55) NOT NULL default "",  
  .  
  .  
  PRIMARY KEY (CustomerNum),  
  UNIQUE KEY CustomerID (CustomerID)
```

which can be dumped by the database management system. This can be used in conjunction with field documentation to provide not only the technical details of each field but also an added explanation of the field's purpose.

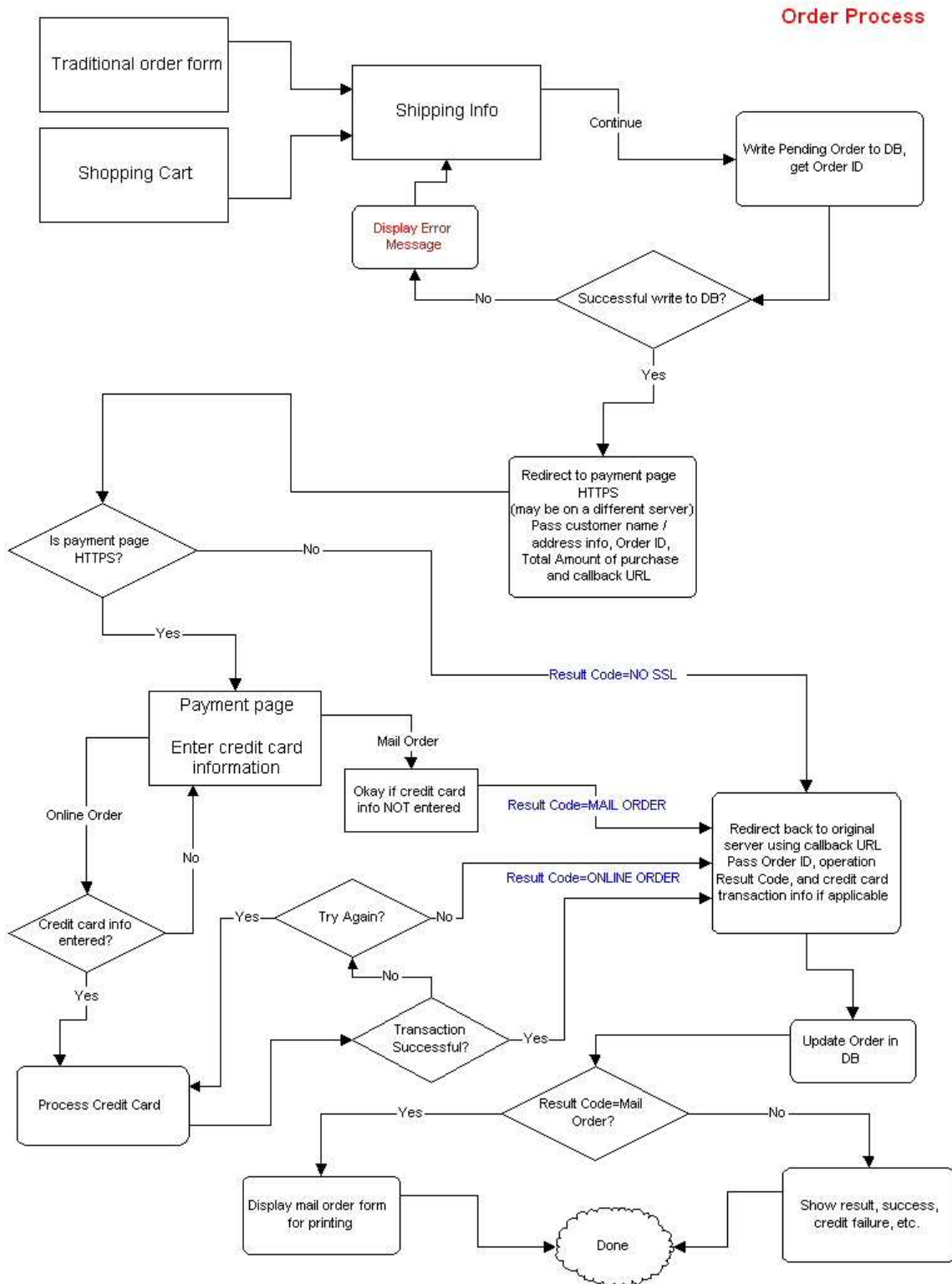
Database relationships are important to understand; a relationship diagram offers the easiest overview of the foreign constraints and keys.

**Note:** Even though some database management systems (such as MySQL using the default table type) do not enforce foreign key constraints, the relationships should be documented. In fact, under these conditions, it is even more important to document since the responsibility of enforcing the relationships falls to the application and hence, to the developer.

An example of a database relationship diagram may be found in Appendix B.

**Summary:** Developer documentation should provide an overview of the organization of the filing system, an overview explanation of each file's functionality (and how it relates to the system as a whole), flow diagrams (in some cases), and code that uses self documenting features or internal comments (usually both). In addition, the developer documentation should include details of the database design, explaining (briefly or more at length as needed) the use of each database entity.

## Appendix A - Sample Process Diagram



## Appendix B – Sample Data Diagram

