User Documentation

The following pages provide some basic guidelines for preparing user documentation. The first two references (User's Guide and User's Reference Manual) deal with the content of specific user documentation (manuals). Depending upon the system and size of the intended audience, it is possible to combine these into one document. The User Training Manual is typically always a separate document. The last section of this document provides some guidelines and tips as to the actual preparation of user documentation.

Section I – User's Guide and Reference Manual

User's Guide
Explains how to build and run the software. Contains a description of the build environment, description of any parameters, description of any input data, and description of any output data.
User's Reference Manual
Describes the function, interfaces and usage of the software. Contains illustrations, sample screens and examples.

Section II – User Training Manual

User Training Manual
Provides an organized structure for conducting training sessions.

Section III – Writing a User Friendly Manual

How to Write a user Friendly Manual
Provides a step-by-step approach to preparing user documentation along with guidelines for standardization.

I. User's Guide and Reference Manual

Although specific headings are indicated below, each user document should contain the following information in a presentable format:

Introduction

Introduce the software product and list its main features in this section. Also, discuss the objectives of this manual.

Intended Readership

Provide a description of the intended audience.

Purpose

Describe the context and creation process of the software product being specified in this manual.

How to use this document

Describe the conditions where and how this document is applicable and how it will be best used.

Related Documents

Refer to any other documents that need to be consulted in parallel with the reading of this document.

Conventions

Mention the terminology and conventions used in the manual to avoid confusion.

Overview

Provide a short description of the software product and its purpose, including benefits and goals. Relate these to corporate goals or business strategies, if possible. Describe the exact services that the software product offers.

Problem Reporting Instructions

This section describes how to report the problems or defects that were encountered while this software was being used. Identify who to contact and how. Always request the basic system configuration information, the specific error messages and circumstances surrounding the encountered problem. The user should also be requested to indicate whether the problem is reproducible or not.

General Instructions

This section is different for each type of user. If it is for the end user it will contain only the most important software features. If it is for software system management personnel it will have a different format. Below are two separate sections for each type of format and its own preferred guidelines. Keep in mind that if they are combined and depending upon the complexity, the simple end user may become confused and overwhelmed.

General Instructions for End Users

Navigation

Describe how to navigate in the application to access specific tasks. Explain the menu features and how to select options. Explain the commands and how to use them to generate particular responses. Screen illustrations are excellent graphical enhancements.

Cautions and Warnings

Mention any cautions that the user must bear in mind while operating the software. List other security features, if applicable.

Probable errors and possible causes

List down possible errors and causes of errors in the software.

Help

Describe what is available as online help and the issues it addresses. Also, describe how to obtain help and how to exit it. Mention other help material (guides, training material, web sites, customer support numbers, etc).

Special Keys

Mention any hotkeys or special key combinations and their functions.

Operations

Description

Give a brief description of the task, its purpose and its function. Include specific information, such as format and field definitions.

Instructions (to carry out the operations)

Describe how to access and exit the function. Give step-by-step instructions on how to perform the task. Give an account of what should happen normally and what should the user do if it doesn't respond normally.

Step #	Instruction	Error messages	Trouble shooting instruction

Cross references to other operations

Use this section to show dependency of one operation on another.

	Operation 1	Operation 2	…	Operation n
Operation 1				
Operation 2				
Operation n				

General Instructions for System Managers

System Initialization and Shutdown

Overview
Give a list of operations to be performed.

Operation #: 1			Operation Name:	
	Prerequisite			
	Step #	Instruction	Error messages	Trouble shooting instruction
Operation #: 2			Operation Name:	
	Prerequisite			
	Step #	Instruction	Error messages	Trouble shooting instruction
Operation #: 3			Operation Name:	
	Prerequisite			
	Step #	Instruction	Error messages	Trouble shooting instruction

Cross Reference to other operations
Use this section to show dependency of one operation on another

	Operation 1	Operation 2	…	Operation n
Operation 1				
Operation 2				
Operation n				

Security
Describe the security issues of the application

Data
Describe data security in terms of integrity: backup and restore operations, data corruption issues, archiving, auditing, transmission.

Access
Describe the application security in terms of access: authorizing access, types of access (privileges), monitoring, and data transmission.

User Support

Task #: 1		Task Name:
	Step #	User Support Task Description
Task #: 2		Task Name:
	Step #	User Support Task Description
Task #: 3		Task Name:
	Step #	User Support Task Description

Reporting

Routine Management Task Log Template

Task #	Category	Audit Criteria	Step #	Description

Management task categories

B	Backup
SC	System control commands
R	Reports
S	Safety features and requirements
RM	Routine maintenance

Problem Solving

Step #	Instruction	Error messages	Trouble shooting instruction
1.			
2.			
n.			

II. User Training Manual Guideline

Each of the following corresponds to a section of the training manual document. Keep in mind that the previously discussed documentation materials can be used to supplement the user training documents. Therefore, the following may be brief in content depending upon the training audience.

Application Overview
Give an overview of the application and how it integrates into overall computer and business environment. Give description of its main functions and its main users. Describe the application's business objectives, scope, and non-goals.

Course Overview
Give an overview of the course and its goals and non-goals. Describe the scope and objectives of the course and mention the category of users for whom it is designed.

Course Materials
List all course material and reference documentation that will be used during training.

Course Outline
This is the main section of the training document. Break the course outline into modules and sections for convenience and easy reference. For each module, include the following information:

- Introduction and objectives
- Functionality of the application

Practice/Case Study
Include a case study or practice exercise wherever appropriate. Design case studies and practice exercises to elaborate functionality of the application to its full.

Glossary
The glossary contains entries for all terms used in the training document.

Attachment Appendices
Attach the following appendices with the user training manual.

Appendix A - Error Messages and Recovery Procedures
Appendix B – Glossary
Appendix C – Index (for manuals 40 pages or more)

III. Writing A User Friendly Manual

Depending on whom you speak to, documentation is either the best part of a software project...or the worst.

Most developers wouldn't be caught dead writing a user manual - they much prefer spending their time building better, more efficient algorithms. Their users, on the other hand, don't really care about the code that powers a software application; they're more interested in getting their work done quickly, with minimal errors.

That's where support documentation, in the form of a user manual (guide or reference document), comes in. Usually considered one of the least important deliverables, it is slowly coming of age, as software companies begin to realize the value of high-quality documentation that answers most user questions and reduces support calls (and expense).

Support documentation allows the user to use the software with ease and efficiency. Ideally, it comprises the items discussed previously and re-summarized as:

1. Interface text: The labels on interface elements like menu items, fields, instructions, confirmations, error messages et al.
2. Application messages: Operational error messages and warnings.
3. Online documentation: Online help, tutorial and searchable help pages.
4. Print documentation: User manual and technical reference manual.

These, in totem, are the user's support system for usage of the software product.

This following section focuses on the user manual, explaining, from a technical writer's perspective, the process by which such a manual is developed, reviewed and delivered. However, the process and planning tips are generic enough to apply to other print documents, in accordance with both their purpose and scope.

Step-by-step

One of the first decisions project managers face is whether the user manual should be a development team production, or whether a technical writer should be assigned the job. Practically, this decision is a function of the size and budget of the project, but an understanding of a technical writer's contribution helps in making an informed choice.

While the developer's responsibility is to be an expert on the structure, features and working of the software, the technical writer is responsible for understanding the users' mindset - their expectations, present level of expertise and possible questions - and then formulating the best way to communicate with them. The quality (ability to communicate) of the support documentation directly affects the level of after-installation support required. Since technical writers are trained in the black art of communication, it usually makes more sense to hire one for the support documentation needed, rather than

have a developer do it.

Documentation, as a process, begins (ideally!) right at the point where the development team has finalized the software design, because:

1. That provides enough information for you, the writer, to start planning the structure of the manual.
2. You have a better chance of understanding the software if you ask questions (lots of them!) while it's still in development. Post-development, the team is usually into the next project, and you end up doing a lot of backtracking and guesswork.
3. The project schedule will typically never allow time between development and delivery for anything more than very superficial documentation.
4. Most important, the ideal scenario is to release the documentation in time for the software testing, so that the test team tests the documents as well as the software simultaneously.

An organized process of documentation will usually have the following phases:

1. Planning
2. Stylesheet creation
3. Development
4. Review
5. Version management
6. Delivery

Asking the hard questions

You should start thinking about the user manual right at the start, and try to have the following questions answered by the time you actually get down to writing it.

1. *Who is the audience?*

This helps you decide the tone and level of technicality of your language, the depth in which the concepts need to be explained, and (very important) the analogies that you can use (familiar ground is best when trying to explain something new). Knowing the following parameters about the intended users would help:

- What is their average age?
- Which computer software packages are they familiar with?
- What are the obstacles they usually experience while using these software applications?
- What are the top five task(s) they plan to use your software for?
- What is their current level of expertise (novice/intermediate/expert) in using particular software packages?

This information is useful when your software builds on existing software currently in use. For example, if you are delivering an intranet email utility that plugs into Microsoft Outlook, it would make sense to find out if your audience has ever used it, and to what level.

This also brings up an important decision: do you decide the minimum technical expertise required of the users of your software, state it as such in the user manual, and get on with things? Or, given the results of your user profiling, do you take on the responsibility of bridging the gap between the current and required level of expertise (maybe by providing a short tutorial as a precursor to the manual)? The schedule and budget would normally make this decision for you.

Some research into the business processes of the target organization will give you even greater insight into the context of user tasks, as well as information for analogies that may be easily understood by them. Additionally, customer meetings, prototype presentations including technical reviews, are great sources of audience information.

2. *What is the scope of the document?*

While the broad goals of the user manual would be to provide information on the installation, usage, administration and troubleshooting of the product, questions like these would help scope the document further:

- Current user expertise versus required expertise: What is the extent of background/explanation that needs to be given?
- Supported platforms: What are the different platforms/operating systems that the manual should address?
- Troubleshooting: What level of troubleshooting are the users supposed to handle? Is there a reporting mechanism for support? Is there separate documentation for troubleshooting?

3. *Should you use a tool for document development?*

The user manual, online help and searchable help essentially build on the same information. Which means that choosing a tool, and its ability to allow you to reuse information from one document for the faster development of another, could be is crucial (especially if your project's on a tight schedule). A number of good tools are available for this purpose. RoboHelp (http://www.ehelp.com/) is an example.

4. *What is the mode of document delivery?*

The user manual can have two modes of delivery and distribution:

- Print: In this case, you take the responsibility of printing it in-house and delivering it to the user (many users demand this). The downside: you get to incur

printing and distribution costs (and the accompanying logistical issues), together with recurring costs every time the documentation is revised.
- Electronic: In this case, you may choose to deliver documentation in electronic format, via CD-ROM at installation time, or provide downloadable material on your Web site. The de facto standard for such electronic documents is Adobe's Portable Document Format (PDF).

Making friends and influencing people

Another important aspect of planning is figuring out your resource requirements, especially if you are a technical writer expressly brought in to the project for support documentation. There are a number of resources you can tap - here's a brief list:

1. **SMEs**: SMEs (Subject Matter Experts) are your guides throughout the documentation project. These are usually members of the development team who will familiarize you with the application, answer your questions and generally be your information bank. This is a good time to determine which developers from the team are to be your SMEs.

Your relationship with the SMEs will go a long way in determining the success of this task.

- Determine a method of communication that is suitable to both. An option is that you post your questions to the SMEs via e-mail, who may then respond in their spare time, or (if the explanation is long-drawn) schedule a meeting.
- Ask the right questions. Understand that on the other side of your question lies a lot of information, and what you get to know will be in direct response to only what you ask. So, spend some time getting your questions right.
- Get familiar with the platform and terminology used in the software. This way, again, you make your meetings with the SMEs efficient.
- Let the SMEs know that you need to know of every change made in the project; any change in the software that affects flow, functionality or interface affects your document. In fact, even with changes that don't affect the user interface, it's a good idea to be in the loop, because there could be reactions that you would want to know about. Again, try and set up an information chain or e-mail trigger for the same.

2. **Project specifications**: Needless to say, getting acquainted with the specification documentation is crucial to understanding the project. The objective of the project from the user's point of view is usually defined very clearly in these - make sure you re-use that, as your users will relate to it.

3. **Prototype**: Since you're going to be writing about the behavior of each feature in the software, playing around with the actual interface is a must. On the other hand, documentation usually begins in parallel with development, so you don't really have anything to go by.

The workaround here is the prototype. The delivery of the prototype by the development team will be a big milestone in your schedule...because that's where you actually start developing the manual. Get this date from the developers, and circle it in your calendar.

Note also that changes take place frequently in the early stages of development, not only in the behavior of the software, but also in the interface elements, text labels and messages. Ensure that your manual reflects the delivered product by referring to the latest prototype.

4. **Schedule**: The cornerstone of this planning stage is the schedule. An important consideration here is the dependencies between your tasks and other milestones in the schedule. Understand the developers' schedule and build your own based on that. Your milestones could be something like this:

1. User profile generated
2. Product information assimilated from specifications
3. Stylesheet finalized
4. Table of contents/outline complete
5. Outline sent for review
6. Outline returned with comments
7. Comments incorporated and outline available for sign-off
8. Sign-off
9. First draft sent for review
10. First draft returned with comments
11. Comments incorporated and draft available for sign-off
12. Sign-off
13. Second draft sent for review
14. Second draft returned with comments
15. Comments incorporated and draft available for sign-off
16. Sign-off
17. Third draft sent for review
18. Third draft returned with comments
19. Comments incorporated and draft available for sign-off
20. Sign-off
21. Delivery

Review and revision efficiency are crucial to ensuring that three drafts are all it takes.

Being Conventional

Conventions in the document lead to patterns that the users can grasp. They then start expecting information in a particular format, thus increasing their level of comfort with the document. Using consistent styles also speeds up assimilation of the information, and helps spot particular information easily on re-reads.

- **Headings**: Headings are a powerful tool in making a huge mass of text look manageable. A common model is that as you go deeper in a particular topic, you indicate that by descending prominence of headings. So, all top level headings will be, say, in a large font size and bold typeface, with the next level taking a smaller font size, and so on. You might also want to number the headings to help users understand the grouping of information.
- **Styles**: While a short piece of text requires only minimal use of styles (bold for highlighting, underline for warning), a book as voluminous as a user manual needs you to be much more creative. You could set conventions for indicating screen names, interface text or text that the user needs to input.

On the other hand, too many conventions negate the purpose - remember, they should assist in quick reading and lookup, and they won't if users have to keep recollecting what a particular style indicates.

- **Indented text and footnotes**: This is text that is peripheral to the point that you are making - for example, background information on a concept that you're introducing or a warning related to some functional step that you're explaining. Add these when you don't want to distract the user from the main flow of information.
- **Bullets and numbering**: Bullets and numbering can also help to break up complex concepts into simpler, smaller information nuggets. The convention here is to use numbering for sequential information only (for example, steps to perform a task) and bullets for other related information that is best presented in points instead of a paragraph.

  Bullets also allow you to group together points related to a concept and ascribe them levels of importance. Much care and consideration should be given to the grouping of information in this manner - it could easily be as confounding as useful.
- **Terminology**: A very, very important rule of creating end-user documentation is to be consistent in your use of words. For example, if you're using the word "function" to indicate the, well, functions of your software, you shouldn't at any point switch to "features", "commands", "menu items" or "actions". To this end, make yourself a glossary of the terms that you're going to use right at the start, and stick to them consistently.
- **Images and illustrations**: Sometimes, a picture really is worth a thousand words - and screen grabs, schematics or flow diagrams can substantially increase the efficacy of your document. Plan your usage of images and illustrations in advance, and be consistent in their usage and labeling.

The Road Ahead

Previously, you were introduced to the process of creating a user manual and illustrating the groundwork you need to do prior to actually beginning work. Now, the following

material will take you through the development process, demonstrating a sample TOC and giving you a few tips on the review and version management stages of the process.

Once you've got your stylesheet done, you've finally reached the point of production - where you actually start creating the document. Generally, the best way to go about it is to create the bare-bones structure (TOC), have it approved, fill in the peripheral information (overview, scope, conventions used, glossary) and then move in to the meat of the matter. This is also the time (ideally!) when the prototypes are finally ready for you to start working with. The following is a generic user manual structure:

1. Introduction
    - The product - introduce the product to the user.
    - The user manual
    - Scope/Purpose
        - Flow
        - Conventions
        - Glossary

2. Installing the software (assuming it's not a separate guide)
    - System requirements
        - Platform Support
    - Information/resources required in the process of installation
    - Installation steps

At this point there is usually a decision to be made about how to depict installation procedures for different platforms. The main criteria here is how different the procedures are - if the steps are drastically different, you will have to explain the procedure separately for each platform. But if the steps are not very different, you could choose the most common platform as your base, and wherever the steps are different, indicate the steps for the different platforms as indented text.

3. Using the software
    - Introduction
        - Purpose of the software
        - What it does and does not do (list the exact tasks)
        - User levels and the implications (segregate the user and admin level task
    - Best configuration (for example, best viewed in 640x480 resolution)
    - Invoking the software
    - Interface elements
    - Steps to perform the required tasks

4. Administration
    - Reiterate the administration level tasks

- Segregate (if possible) into administration, maintenance and troubleshooting functions, and then get into explanations
- Always lead in to a task with scenarios (for example, you need to shut down the server over the weekends and at the end of the day - here's how...)
- Also, try and bring out exceptional scenarios at the same time (to continue the above example, the administrator would not shut down the server over the weekend if there has been a request for remote access by one of the users)

5. Troubleshooting
    - For each error condition describe:
        - What happens on the display
        - The error message displayed
        - What it means and what is the implication with respect to the attempted action (for example, the user will have to re-enter information)
        - Steps to take to rectify the error

6. Appendix
   An appendix allows you to expound on peripheral information that would be detracting when given in the main body. Detailed diagrams, flow charts, or references to books/tutorials on related software could be included here.

Here are some quick tips to assist you in developing this structure:

- Be visual: The most comforting thing for the user will be to see on screen what they've seen on the manual's pages, or vice-versa. Try and use screen grabs and small schematic diagrams wherever appropriate.
- Importance of relevant analogies: Essential if your software introduces concepts new to the user.
- Use of transition words: "because", "therefore" and "consequently" are powerful words when talking about cause-effect relationships that the user isn't aware of.

Once you're done with filling in the details, it's time for a review.

Looking For Improvement

The review is a crucial process for any document, and more so for the user manual. You are dependent on the SMEs for verification of the technical information, on peer writers for editorial comments on structure and flow, technical support guys for evaluation of whether you've covered the most common support requests, and - of course! - a sample set of actual users to tell you whether it works.

But before you get to the point where you release drafts for review by these groups, you need to review the document yourself. Here's a quick cheat sheet that assists in this process:

1. Is all the information there? Have all the key terms been defined? Are instructions to cover all tasks there?
2. Is the usage of paragraphs appropriate? Are they too long? On the other hand, guard against the paragraphs being so short and abrupt that the information seems unrelated.
3. Does the flow make sense?
4. Are the levels logical? Is the grouping together of modules sensible?
5. Are the descriptive headings apt? Could there be a summary at the beginning and end of each section to guide the user better?
6. Are there inconsistencies in usage of terms?
7. Is the tone consistent throughout? Are you shifting from the first to the third person and back again?
8. Are all topics covered to a consistent level of depth? Sometimes, at the time of writing, you may not have the same level of information on all topics covered (or you may just have been in a hurry to get the job done). This difference is very noticeable, and looks sloppy - watch out for it and fill in the consistent level of detail.

Once you've covered this ground, you can release the draft for review by the others involved.

Cheat Sheet

A thorough review needs a lot of commitment and time. You will need to make it easier for your reviewers by including quick cheat sheets stating what they should be looking for. An example cheat sheet for the SMEs would be:

1. A technical review is not an editorial review.
2. Focus on the technical facts to verify that the technology works as documented.
3. Verify the technical accuracy of all procedural steps included in the document.
4. Verify the technical accuracy of all screen captures in the document.

User reviews are a tad trickier than the others are because of the lack of resources. First, you may not have access to the actual users to review your document. And second, they may not really be motivated at that point to take the time to review your document. The workaround is to use your marketing and QA departments, and perhaps the people from the customer's end who are involved in the project.

Once the reviews are in, you need to get down to implementing the changes suggested. One tip would be to start revision on a document only after all the review comments are in. Also, while we won't get into the art of accepting feedback, you have to be in control of the changes that you agree to make. While you can change the information quite a bit

at the time of the first review, you should try and restrict your changes to corrections only after the second review; structural changes this late in the process will throw you off.

Version Control

A characteristic of documentation is that if you notice even one inaccuracy in a document, it will put you off going through the rest of it. The gravity of this increases manifold when you're talking about a user who's looking to this document to understand your software. Ensuring that your manual reflects the latest version of the software is crucial, and this is where tying the document version number with that of the software comes in.

Another consideration here is version nomenclature. You could tie this in with the software, using x.y nomenclature that has x changing with every baseline change and y changing for every intermediate release of the document. Also, when you revise the document, you should record the reason/description of the change in the document's revision log.

Putting The Package Together

With all of this behind you, you will finally be ready to release the manual. The following frills will complete the package:

1. Cover page
    - The name of the software should be written in accordance with the brand decided.
    - The version number of the software should be clearly stated.
    - The name of the developer with address and contact numbers.
2. Table of contents
    - The topics should be linked to the matter inside.
3. Notifications for proprietorship and confidentiality
4. Headers and footers
    - Headers could include the project name and version number of the document.
    - Footers can have the page numbers and a short confidentiality notice.

It might also be a good idea to include a feedback form as the last page, as your users will probably get back to you with suggestions. This will be especially useful if there is a second phase of development for the software.

And that's about it.

Below is some further commentary.

Developing documentation without a tech writer

In a perfect world, every development team has one or more technical writers working alongside them to ensure quality documentation is available for their product. However, not all companies see the inherent value in technical writers, or perhaps you're working with a small group that cannot afford the extra expense. Regardless of the reason, you should keep a number of things in mind if you must develop your own documentation along with your software product.

Needs analysis
The first thing most technical writers do is called a *needs analysis*, which can be split into two vital parts: audience analysis and task analysis.

Audience analysis
In audience analysis, you identify as specifically as possible who will be reading this documentation. Are you documenting your Application Programming Interface (API) for other development teams within the company? How about other developers outside the company? There is a huge difference between the two in terms of what you can assume the audience knows about your internal processes and what company data you can share and what you cannot.

Perhaps you're writing documentation for the end user. Are your users typically computer-savvy? Novices? Do you have a wide range of user types and backgrounds? If you're not sure, there are a few ways to find out. Talk to the help desk, technical support staff or development staff to find out their experiences with your user base.

Task analysis
Task analysis involves determining how the reader will actually use this documentation. Are you preparing instructions on how to install your product? If so, then you'll want to focus on the step-by-step actions required to accomplish the installation. Are you detailing, as mentioned earlier, your API so other programmers can properly interface their products with yours? In this case, you probably want a reference format, where you've broken down the API components into some form of logical, top-down arrangement that will make it simple for another developer to find what he or she is looking for.

Sometimes a mix of the task and reference approaches is best, instead. Reference sections can be mentioned in the instructions and included as separate appendices. Other methods of accomplishing the mix and match involve tips, warnings, notes, tables, figures, and other elements that allow you to call the reader's attention to particular issues without making the write-up too clunky or dense.

Graphic elements
It's important to set apart such features as warnings and tips to avoid their blending in with the rest of the walk-through instructions or reference material. Take a look through a set of manuals and technical books and you'll see lots of examples of what works and what doesn't. Typically, adding some kind of border is of immense help, whether it means setting off a tip with a box around it or having a set of horizontal bars before and after a warning. Graphics can also be a nice touch, especially with warnings when you want to make it perfectly clear that this is something to watch out for, or not to ever do.

Wording
And then there's the issue of wording, which takes us back to the audience. When you're writing for a beginner or non-technical audience, it's absolutely essential to question every assumption you might have about the readers' knowledge. Will they know this acronym? Just in case, include a spelled-out version in the text or in a glossary. Is there a simpler way to put this term? The vice president of marketing might have no idea what an API is, but if you use "a set of tools that lets programmers have their programs talk to our programs," it's wordier, but you get your point across.

For a technical audience, you can use more jargon (i.e., stacks, threads, and forks) without having to specifically spell out what you mean in layman's terms, but still don't go overboard. Define your acronyms if you can't be sure they're common knowledge. Don't use big words just to show off your vocabulary. Keep it simple and to the point; that's far more impressive than sending people for their dictionaries. In either case, technical or non-technical, it can help to have people who fit your audience type read over the document and give you feedback.

Consistency
Finally, there's the issue of consistency. Decide up front if you're using metric or English units, or need to have both included for any measurement. Define the little things like whether you intend to use 5MB or 5 MB, 5 feet instead of 5', and so on. When you're referring to variables, this can get interesting: How do you want to refer to them when you're using a format such as variable = definition of value, as opposed to variable = example value? Perhaps name = <filename> or name = *filename* and then name = numbers.txt.

One important thing to be consistent with is how you're representing text that the user needs to type in. Some people use a monospace font such as Courier for user-entered text references. Others underline or **bold**. Be guided partially by the conventions in your industry and format, but in general, just do it the same way throughout the document(s). Keep a separate file open or piece of paper handy to record consistency issues and decisions as they come up.

Conclusion
Although it's often best to have a professional technical writer handling your documentation needs, sometimes developers are stuck having to write their own for a variety of reasons. Just take that same attention to precision and detail you use in your daily programming work and apply it to the text. Keep the formatting issues, language issues, audience issues, task issues, and so on, in mind and you'll find yourself with useable, readable documentation every time.

The Seven Deadly Assumptions of Technical Communication

Assumption 1: Roles and responsibilities are obvious on this project.
Assumption 2: It’s clear what needs to go into this manual.
Assumption 3: We only need to publish this on paper.
Assumption 4: This software tool is ideal for this kind of manual.
Assumption 5: Standards aren’t necessary for this kind of work.
Assumption 6: We don’t need any of those fancy documentation development methodologies for this project.
Assumption 7: Maintenance won’t be a problem here.