A Letter of Introduction from Manning's Publisher

Dear Manning Author,

We have sent out your contract and with that let me introduce you to a few names here at Manning, and to some of our processes. Contact information is at the end of this letter.

I will be glad to respond to any questions or ideas you may want to send me. If you do not quickly get a response from me please contact your editor or any administrative staff member, and they will be glad to help you get in touch.

By now you should be registered in Gregor and know the location of the Writing Guide and Author Template. Please contact us right away if you do not have access or don't know with which materials you should start. It is important that you read the orientation documents before you start writing, and that you begin using the Author Template right away. In the following pages is a short primer on Manning Book and Chapter Structure; please study it carefully. (You can also download a copy of the Structure primer from the Author Orientation folder in Gregor.)

In about a week, we'll schedule an opportunity for you to discuss the first steps in getting your book off the ground with me. We call this the "Author Launch" and it is a great time to ask any questions you have about the Manning CMS, the writing guidelines, or any other aspect of the publishing process. To make this conversation as productive as possible, please take some time in the next few days to login to Gregor and download the documents in the Author Resources folder. Please also post in your Proposal folder your most recent Table of Contents (or even Chapter One in your Manuscript folder if it is ready), as we may add your editor(s) to the call to discuss your plan.

We hope to receive the first draft of Chapter One as soon as you have it, hopefully not long after your launch. Discussing and improving it will give you and us a way to get on the same wavelength and it's important for that to happen before you have written a lot of other material. We ask that the first chapter you write be the first chapter in the book.

Development is managed by Maureen Spencer. She will assign a Developmental Editor to your book (from time to time reassignments may occur) and will monitor your progress during the writing phase. Your DE will be your main contact with Manning during development, so if at any time you find you are having trouble communicating with your DE, please let me know (maba@manning.com).

When we have 1/3 of the manuscript we will organize the first manuscript review by outside advisers, a mixed group of experts and some non-experts. The review will be managed by our review editor, Karen Tegtmeyer. Karen will send you an introductory message soon. She will also organize a 2/3 and final manuscript review.

Once your manuscript is accepted for publication it goes into production. Production of your book will be managed by Mary Piergies. I am sure you will have many occasions to deal with her. During manuscript development, any questions you may have about the word processing template we ask you to use and about graphics and text formatting should go to her.

Further down the line, the cover designer will be Leslie Haimes and the promotion of your book will be managed by Ron Tomich. If you have any promotion ideas at any time, please send them to Ron with a cc to me.

In the future, any questions about your royalties and advances should be addressed to Kimberly Dickinson who is our Business Manager.

We are all here to try to make the writing process smooth for you. Of course, the burden of writing is on you – nothing we can do about that. With contract discussions out of the way, I hope you are well focused and have begun to work on the manuscript.

Here's the contact info for the Manning people I mentioned and some others you might run into. Please keep it for future reference:

Marjan Bace, Publisher <maba@manning.com> (203) 653-2280
Michael Stephens, Associate Publisher <mist@manning.com> (317) 816-1514
Mary Piergies, Director of Production <mp@manning.com> (708) 771-0766
Kimberly Dickinson, Business Manager <kidi@manning.com> (609) 835-2793
Maureen Spencer, Development Manager <masp@manning.com> (502) 222-2416
Megan Yockey, Assistant Acquisitions Editor <meyo@manning.com> (212) 731-2408
Christina Rudloff, Assistant Acquisitions Editor <chru@manning.com> (619) 567-6944
Karen Tegtmeyer, Reviews Editor <kate@manning.com> (515) 331-4262
Gabriel Dobrescu, Webmaster <gado@manning.com> (647) 291-1279
Leslie Haimes, Cover Designer <leslie@raphael.mit.edu> (She prefers email contact)
Ron Tomich, Director of Sales & Marketing <roto@manning.com> (720) 204-1022

Best regards,
Marjan

Marjan Bace, Ph.D.
Publisher
Manning Publications Co.
Greenwich, CT
203 653 2280
http://www.manning.com

Manning Book and Chapter Structure

Chapter structure

A chapter is organized in sections numbered m.n (m = chapter number, n = section number). The first and last sections in every chapter are standard:

- The first section provides the background or introduction the reader needs to understand the concepts in the chapter.
- The last section is the chapter summary.

While the last section should as a rule have the title, "Summary" the first section can be called anything, but it should leave the reader with the sense that he will get the help he needs to follow the rest.

A chapter is divided into 4-6 sections. These are not rigorous rules but just to give you a sense of what is reasonable. Having 10 sections in a chapter is not a problem (unless you do it all the time.) So a 5-section chapter should look like this:

Section 1 Background needed for topic of chapter

Section 2 Core topic

Section 3 Core topic

Section 4 Core topic

Section 5 Summary

In the Examples section below, we show some nice ways to organize a chapter.

Section structure

A section is divided into a somewhat smaller number of subsections (3-5 say), numbered m.n.k. The subsection is where the rubber hits the road – it's where the character of your book is determined. Here is what we recommend: Each subsection

- is a relatively short (1-3 page), relatively self-contained lesson;
- each has an example; and
- the text uses writing techniques we call:
    - segue
    - metatext

- crutch

These terms are explained below.

Subsection

Think of a subsection as devoted to one small, easily identifiable issue. Each example is short and simple and makes only one point, which is related to the subject named in the subsection title. Each subsection gives readers a sense of closure – that they have now learned all they need about that one subject. [For emphasis, you can occasionally end a subsection with a reward such as "… That's it! This is all you need to know about XYZ."]

Digression on terminology (background, metatext, segue)

Occasionally we hear that "background" and "metatext" are confusingly similar in the minds of some. They should not be. This may be an understatement, but these terms are entirely, totally, absolutely different... :)

1 Background

Background passages prepare the reader before getting into the specific subject at hand. This can mean reminding him of the prerequisites needed to understand the coming section. A common type of background is to give the big picture before getting into the details of something: the view from one story higher, as it were. A background sets the stage so that the specific topic, when it arrives, makes immediate sense. Below is an example from Spring in Action, the beginning of Section 4.1, page 134. The way it starts tells you a lot: "Before we jump into Spring's different DAO frameworks, let's talk about..."

2 Metatext

The only similarity between background and metatext is that both interrupt the flow, although they do that at different scales. Any similarity ends there. Metatext discusses the subject being presented, but in a special way: it helps the reader locate the present point in a long chain of an argument or multi-step explanation. It is not a part of the argument per se but is instead a navigational aid. While background prepares the reader for what's to come by giving him prerequisites and the larger-scale knowledge he needs, metatext guides him through the argument or explanation. Background comes ahead of the core text, metatext is inserted within the text. Background is generally longer, metatext shorter.

3 Segue

Another feature of good writing that's related to the two above is segue. In a sense, segues are just specialized metatext: they comment on the flow typically between sections and subsections, providing a link between them. A possibly useful way to think of metatext is as segues between different lines of thought or argument. They can of course occur within sections when the argument gets long or difficult to follow. Possibly

better terms for "segue" are "bridge" and "transition."

4 Crutch

Whenever you discuss an important or difficult concept start with a “crutch” which motivates the topic through a concrete example. Too many authors happily launch into a new and difficult topic with nary a word of motivation. The crutch is a special kind of introduction, one that describes a scenario you might want to solve and shows how it can be difficult to do so in the absence of the to-be-described concept. Thereafter, the scenario serves as the running context in which the reader can better understand the discussion of the concept.

A crutch livens up the writing and should be used frequently, perhaps as much as every three pages or so. See examples below.

Examples

EXAMPLE OF CHAPTER ORGANIZATION INTO SECTIONS

From *Object Oriented Perl*, Chapter 6 Inheritance:

6.1 How Perl handles inheritance 168

6.2 Tricks and traps 178

6.3 Example: Inheriting the CD class 193

6.4 Where to find out more 201

6.5 Summary 202

Or from *Spring in Action*, Chapter 7 Accessing enterprise services:

7.1 Retrieving objects from JNDI

7.2 Sending e-mail

7.3 Scheduling tasks

7.4 Sending messages with JMS

7.5 Summary

EXAMPLE TOC (AVAILABLE FOR ALL MANNING BOOKS ON WEBSITE)

http://www.manning.com/walls2/excerpt_contents.html

EXAMPLE OF BACKGROUND

From Spring in Action, chapter 4, page 134:

"With the core of the Spring container now under your belt, it's time to put it to work in real applications. A perfect place to start is with a requirement of nearly any enterprise application: persisting data. Each and every ...

....

4.1 Learning Spring's DAO philosophy

Before we jump into Spring's different DAO frameworks, let's talk about Spring's DAO support in general. From the first section, you know that one of Spring's goals is to allow you to develop applications following the sound object-oriented (OO) principle of coding to interfaces. Well, Spring's data access support is no exception.

DAO stands for data access object, which perfectly describes a DAO's role in an application. DAOs exist to provide a means to read and write data to the database. They should expose this functionality through an interface by which the rest of the application will access them. Figure 4.1 shows the proper approach to designing your data access tier.

As you can see, the service objects are accessing the DAOs through interfaces. This has a couple of advantages. First, it makes your service objects easily testable since they are not coupled to a specific data access implementation. In fact, you can create mock implementations of these data access interfaces. That would allow you to test your service object without ever having to connect to the database, which would significantly speed up your unit tests.

In addition, the data access tier is accessed in a persistence technology-agnostic manner. That is, the data access interface does not expose what technology it is using to access data. Instead, only the relevant data access methods are exposed. This makes for a flexible application design. If the implementation details of the data access tier were to leak into other parts of the application, the entire application becomes coupled with the data access tier, leading to a rigid application design.

One way Spring helps you insulate your data access ties from the rest of your application is by providing you with a consistent exception hierarchy that is used across all of its DAO frameworks.

EXAMPLE OF METATEXT

Page 112 of *Java Reflection in Action*:

"Now let's improve George's logging facility by using call stack introspection. First, we

present a better interface in listing 5.2. This interface eliminates those parameters that can be determined reflectively...."

EXAMPLE OF SEGUE

From *Spring in Action*, end of Section 7.3.1 connects to 7.3.2, page 250 (PDF of Chapter 7 available from Manning's site):

"(end of Section 7.3.1)... Unfortunately, that's a limitation of using Java's Timer. You can specify how often a task runs, but you can't specify exactly when it will be run. In order to specify precisely when the e-mail is sent, you'll need to use the Quartz scheduler instead.

7.3.2 *Using the Quartz scheduler*

The Quartz scheduler provides richer support for scheduling jobs. Just as with Java's Timer, you can..."

EXAMPLE OF CRUTCH

From *Spring in Action*, Section 7.2, page 244 (PDF of Chapter 7 available from Manning's site):

"Suppose that the course director of Spring Training has asked you to send her a daily e-mail outlining all of the upcoming courses, including a seat count and how many students have enrolled in the course. She'd like this report to be e-mailed at 6:00 a.m. every day so that she can see it when she first gets to work. Using this report, she'll schedule additional offerings of popular courses and cancel courses that aren't filling up very quickly. As laziness is a great attribute of any programmer you decide to automate the e-mail so that you don't have to pull together the report every day yourself. The first thing to do is to write the code that sends the e-mail (you'll schedule it for daily delivery in section 7.3).

To get started, you'll need a mail sender, defined by Spring's MailSender interface. A mail sender is an abstraction around a specific mail implementation. This decouples the application code from the actual mail implementation being used. Spring comes with two implementations of this interface:..."

Two examples from from *Object Oriented Perl*:

Example 1. Suppose that you wanted to create a subroutine to return the items that two arrays have in common. This might come up in a program that had to compare a master list of files with a list of files that had changed, in preparation for saving the changes. How could you pass two lists to one subroutine? You couldn't just pass the lists one right

after the other, like this:

@master_list = qw(a b c d e f g);_

@change_list = qw(b c f);_

$intersect = get_intersect @master_list @change_list;

Because of flattening, get_intersect wouldn't receive two arguments.

It would get ten, one for each element of both arrays.

(a b c d e f g b c f) not (a b c d e f g) (b c f)

The solution is to pass references. A reference is…

Example 2. Suppose we wanted to implement a subroutine called listdir that provides the functionality of our operating system's directory listing command (i.e., dir or ls). Such a subroutine might take arguments specifying which files to list, what type of files to consider, whether to list hidden files, what details of each file should be reported, whether files and directories should be listed separately, how to sort the listing, whether

directories should be listed recursively, how many columns to use, and whether the output should be pages or just dumped. But we certainly don't want to have to specify every one of those nine parameters every time we call listdir:

Listdir("*", "any", 1, 1, 0, 0, "alpha", 4, 1);

Even if we arranged things so that specifying an undefined value for an argument selects a default behavior for that argument, the call is no easier to code and no more readable:

Listdir(undef, undef, 1, 1, undef, undef, undef, 4, 1);

Some programming languages provide a mechanism for naming the arguments passed to a subroutine. This facility is especially useful when implementing a subroutine like listdir, where there are many potential parameters, but only a few of them may be needed for a particular call. Perl supports named arguments in a cunning way…

POSTING TO GREGOR

Please follow these protocols and procedures when posting files to Gregor.

New Documents vs. New Versions

All initial uploads of elements (chapters, appendices etc.) should be posted in your Manuscript folder using the "Add Document" feature. Please use the file name and versioning protocols detailed below. All subsequent versions must be posted using the "New Version" feature. This will cause the versions of each element to be "stacked" and will avoid the disorder and confusion of having multiple versions of a single element living in the Manuscript folder side-by-side.

To upload a new version, click on the document you want to update and select "View/Update Details". Once you are in the document's Details screen, you will find a "New Version" button on the toolbar (the first button on the left). Clicking the new version button will take you to an upload screen, where you can choose the file to be uploaded and select the version number (see more below). The new version will have the same file name as the original upload.

Document History

Adding a new version of a document will not replace the previous versions. Any user can download any version of a document by going into the document details and clicking on the History button. However, if you select a document and choose "Download" (as opposed to entering the Details screen), you will automatically download the most recent version.

Check In/Check Out

Authors and editors may use Gregor's "Check In/Check Out" feature to prevent changes to a document while a revision is in progress. Simply go into the document details and click on the "Check Out" button. No one will be able to post a new version while the document is checked out, unless the User who has checked out the document unlocks it (by selecting "Unlock" from the document details toolbar). Other users will still be able to download checked out files. When you are ready to check a document back in, return to the document's Details screen and click on the "Check In" button. You will be presented with an upload screen, where you can select the new file to be uploaded from your hard drive and choose a new version number. This feature will be particularly useful during production, when changes to files must be tightly controlled.

Document Details

If you wish to update a document's details (file name, short description), simply navigate to the document Details screen and select "Update" from the toolbar. You can see the status of any file by clicking the History button; there is a tab for viewing Changes and a tab for viewing the Download history. You can also add a Comment from within the document Details screen.

File Names

All manuscript files should have brief, descriptive, unique and consistent file names. We

request that you select a short identifier for your book files, preferably an abbreviation of the book's title. (Keep in mind that many of our books are part of our "in Action" series, so using a simple acronym might not be totally unique. Selecting a keyword or two from the title might make your files more distinctive.) Chapter files should use the abbreviation CH and chapter numbers should be listed using two digits. Front Matter can be abbreviated as FM and Appendices can be abbreviated APP.

For example, if you are writing a book titled "Computer Books in Action", your manuscript files should look like this:

CBiA CH 01 (or Computer CH 01)
CBiA CH 02
CBiA APP A
CBiA APP B
CBiA FM

We also recommend that you enter the chapter's title and the part to which it belongs in the Short Description field (e.g. for CBiA CH 01 the short description might read "Part One: Introduction").

Once the book enters production, all illustrations (line art, screen shots, etc.) should be placed in the Final Graphics subfolder within your Accepted Manuscript folder. Files should be numbered Figure 1.1, 1.2, 1.3, etc. with the first number representing the chapter number and the second number representing the figure number within that chapter. This must be done prior to handoff to production; you may use temporary file names for figures during development if you find it easier to manage them with more descriptive titles. Posting figures to Gregor during development is not necessary, but if you would like to do so you can use the folders in the Authors Space -- they are to be used at your discretion. If you would like subfolders created in the Author Space, please contact a staff member.

The Gregor file name does not need to be the same as the files you keep on your hard drive, but to avoid confusion you may wish to use the same conventions on your own computer as we expect to see in Gregor.

Versioning Protocols
As stated above, each manuscript element should be posted as a new document, and all revisions should be uploaded as new versions. Gregor has built-in versioning capabilities for this purpose; you will have an option to indicate the version number during the post (you will not need to include this in the file name). **Authors should post their files using whole numbers.** Initial chapters (posted as new documents) will automatically be designated as version 1.0. When an editor posts an edited version, they will use decimals (1.1, 1.2, etc.). When an author incorporates revision requests and posts their revision of a chapter, they should use the next whole number (2.0, 3.0, etc.).

Your Proposal/ToC folder will contain a copy of your proposal, as well as a copy of any

external reviews gathered during the acquisitions phase. The proposal folder will also be the location of your book's Table of Contents. Please maintain a current Table of Contents in your Proposal/ToC folder at all times, and post new versions using the same protocols described above. You can abbreviate Table of Contents as TOC.

Please explore, experiment and get familiar with Gregor. If you have questions or encounter problems, please let us know.
