13 Single sourcing

What's in this chapter

- The traditional workflow
- Evaluating whether single sourcing is right for a project
- Benefits of single sourcing
- Objections to single sourcing
- Planning for single sourcing
- Choosing single-sourcing tools

Single sourcing refers to a writing process in which you create multiple deliverables from one set of files.

Applications of single sourcing include:

- Creating two (or more) versions of a deliverable—Suppose the administrator and user guides for a product have similar content, but the administrator book contains programmer-level information that is not appropriate for the user guide. You can use one set of files to produce both manuals instead of creating a set of source files for each book.

Similarly, you can create guides for similar products that share some features.

- Creating multiple output media—If you need both printed and online versions of product information, you can set up files that produce both types of output.

Because single-sourcing processes can be highly automated, it's possible to create different deliverables and output types with little or no manual conversion or post-processing cleanup.

NOTE: In many single-sourcing environments, authors create multiple versions of a document *and* then different output types for the versions; the two are not necessarily exclusive.

Successful single sourcing requires quite a bit of planning. You can, however, use single sourcing create high-quality deliverables *and* save money and time. If you are the lone technical writer in your company or if your department is understaffed, single sourcing could be the only approach that makes it possible to complete multiple deliverables on time and within your budget.

The traditional workflow

Without single sourcing, a documentation department would typically use one of two approaches to complete multiple, related deliverables:

- Developing deliverables simultaneously (parallel development)
- Developing one deliverable first, and then the other (serial development)

Parallel development

Parallel development means that different versions are are created, in separate files, at the same time. In the case of documents with similar information, one writer completes one version of the document while another writer works on the other version. When creating printed books and online help, parallel development often means that one technical writer creates the printed book and another creates the online help.

This approach has a few advantages:

- Both deliverables are ready at the same time.
- Writers can specialize in print or online work and optimize the information for each deliverable, or they can focus on writing for a specific audience (for example, user-level documentation vs. more complex administrator-level material).

But there is also a significant disadvantage. Parallel development is very labor-intensive and maintenance is problematic because you're creating the same information twice. This duplication of effort can also lead to differences in terminology in the multiple versions, which confuses the readers.

Parallel development does make sense when the information in the deliverables is different. However, when there is content overlap, parallel development is time consuming and inefficient.

Serial development

Serial development could be considered single sourcing. You write the information once, and then you strip out or add new information for a similar product, or you

convert the source files from one format to another. Serial development has some advantages:

- Because the information is written once, information is consistent across all deliverables.
- The second deliverable is not created until the first deliverable is completed, so the information is finalized.
- Maintenance is simplified because you only convert once per release and do not have to maintain two sets of documentation.
- One writer can first create the first deliverable and then complete the second, so it's less expensive than having two writers working in parallel.

But there are some serious disadvantages:

- Serial development means that one deliverable will lag significantly behind the other. For example, the printed version of a document might be ready three or four weeks before the online help. This leads to scheduling problems before a release because you have to build in several weeks for the conversion and reformatting process. A similar delay occurs when you're creating slightly different versions of a document. The time it takes to strip out nonapplicable material, add new information, and change terminology creates a lag time between deliverables.
- When you're creating printed and online formats, the print tool and the online help tool sometimes do not work well together, so formatting is lost when you transfer information from one tool to the other. The cleanup that's done in the second version must be repeated for each release (unless you keep the

files and only put in the changes, in which case you've switched to a parallel development process).

Many companies use serial or parallel development. However, single sourcing provides them with another option. In a single-sourcing workflow, you create a single set of files that contains all the information for multiple versions, different output media, or both.

Evaluating whether single sourcing is right for a project

Single sourcing is not appropriate for every project. If the content of the various deliverables does not overlap, there's little reason to use single-sourcing techniques. For example:

- Unless two products share several features, it's not helpful to combine information about both products into one set of source files.
- If the content of your online help has little in common with your printed user guide, there's no sense in trying to create the help and the printed user guide from one file set.

Benefits of single sourcing

Using one set of files to produce multiple deliverables does offer substantial benefits for your documentation effort, including the following:

- Reducing time to market
- Minimizing errors and inconsistencies

- Saving money
- Presenting information customized for each delivery medium

Reducing time to market

Because multiple deliverables are generated from the same source files, you no longer have to maintain two (or more!) sets of content in a parallel development process.

Working with one set of source files reduces the amount of time necessary to complete the documentation for a product. Less time spent on documentation can help get a product to market more quickly. Also, once a single-sourcing process is set up, it takes less time to update documentation for new releases of a product.

Minimizing errors and inconsistencies

Maintaining the same information in two locations is obviously time consuming (you have to make every update twice), but there's another, less apparent cost. Maintaining multiple information sets increases your chances of making mistakes or adding inconsistent information.

If you have to make 100 changes to your content, you would make 100 changes in a single-sourcing environment. But if you're maintaining two sets of source files, you have to make 200 changes. Or, more accurately, you have to make 100 changes two times. This is extremely tedious for the writer and can lead to additional errors.

Saving money

At a minimum, single sourcing eliminates the problem of making updates in two places. This alone can provide significant savings. But you can also eliminate the costs associated with maintaining multiple file sets. This can make the other savings look puny—especially if your content is later translated into other languages.

You also save money because of increased efficiency and cleaner files.

Presenting information customized for each delivery medium

In addition to saving money and eliminating the need to maintain multiple file sets, single sourcing lets you develop a template for each delivery medium that takes full advantage of that medium. For example, a glossary might be delivered as a series of pop-up topics in online help, but it can be the standard alphabetical list in a book. Reference information can be hyperlinked and made searchable online. Detailed conceptual drawings are probably best left in the printed, high-resolution book. But in online help, hyperlinked flow charts are possible.

Making the business case for single sourcing

Consider the case of a typical software company. The company has approximately 6,000 pages of documentation in a total of 10 books. The material is converted to online help and is translated into eight languages.

The writers develop books in a text processing tool and then save the files to rich text format (RTF), which preserves fonts, italics, and other formatting. The writers import the RTF files into an online help development tool, and then they clean up formatting problems to create WinHelp. Typical formatting tasks include reimporting graphics (which are lost during conversion), recreating hyperlinks and cross-references (lost during conversion), breaking material into topics, creating pop-up topics, and correcting numbering problems. On average, the cleanup process for a 200–300 page book takes about three weeks (120 hours). At a very conservative $30 per hour cost, that works out to $36,000 in conversion costs for 10 books. Therefore, the conversion costs for the English source and all eight languages means a total of $324,000 (9 languages x $36,000)—and that doesn't include translation costs.

The company now introduces a single-sourcing process. The conversion to online help is handled by converting the source files to online formats with an automated mapping tool. The conversion process takes just one hour per book! The company incurs approximately $13,000 in one-time costs for training, template development, software licensing, and consulting. Even with a much higher conversion rate of $120 per hour (versus the $30 for the conversion with the online help development tool), the recurring conversion cost is only $10,800 (1 hour x $120 per hour x 10 books x 9 languages). Translation costs aren't included in this total, either.

The old process costs about $324,000 per release. The new process costs about $23,800 for the first release—and that includes the $13,000 in one-time setup costs.

In short, the cost savings are enormous. If this company produces two releases per year (one every six months), the single-sourcing solution will save more than half a million dollars in the first year!

Objections to single sourcing

The biggest objection to single-sourcing concerns using one set of files to create multiple outputs. Single-sourcing

opponents believe that it's impossible to write content that works well both in printed and online documents. They say that the presentation requirements for print and online are too different for a common source file. It's certainly true that some single-sourcing efforts have produced mediocre results. For example, you've probably seen online help that read more like a book—windows full of conceptual information that a user really doesn't want when trying to figure out how to perform one task, and text that didn't take advantage of online hypertext linking.

It's quite possible to write content that works in several output formats. Extensive planning and using the right tools can give you complete control over the content and appearance of multiple deliverables generated from one set of files.

The 90 percent point

Although it's probably true that "hand-crafting" printed and online materials separately will result in higher-quality results, a solid single-sourcing process can produce very good results at a fraction of the cost. This is the 90 percent point. If you can achieve 90 percent of the "hand-crafted" quality at a fraction of the price, you must decide whether the cost you incur to achieve that last 10 percent is worth the effort.

Planning for single sourcing

Planning is an important part of any writing project, but for a single-sourcing project, it's required. There several factors you must consider before applying single-sourcing techniques.

Considerations for creating multiple versions of a document

When setting up files for multiple versions of a document, you need to determine how much information is shared among the deliverables and how much is unique, and what terms and names differ across the deliverables.

Determining what information is shared

The information that is shared across documents provides the backbone for each version.

Suppose you're documenting a printer that has three different models, and you need a user guide for each. You need to map out which features are shared and which are unique. You can create a table to figure this out. Table 5 shows a feature map for different models of a printer.

Table 5: *Mapping the features of three printer models*

Printer model number	500-sheet paper drawer	Secondary paper drawer	Color printing	Duplex printing attachment
Model A-1000	X	X		
Model B-2000	X	X	X	
Model C-3000	X	X	X	X

Based on the table, information about the 500-sheet paper drawer and the secondary paper drawer would go

in all manuals. However, information about the duplex printing attachment (which enables printing on both sides of the paper) would go only in the manual for Model C-3000.

Determining similarities and differences in terminology

Even if two or more products have overlap in their features, the products most likely don't have the same name, and similar features may not even be called the same thing in the different products. Also, the titles of the products' manuals are probably not the same (particularly if the product name is part of each title). To handle these kinds of differences, you need to come up with a list of placeholders for names and other terminology. For example, you would need a placeholder called ManualTitle for the printer documentation; you would change the definition of that placeholder based on which book you were creating.

Considerations for creating multiple output types

Because components in a printed document become online help components (or vice versa, depending on what tools you are using), you must consider each element of the book and what it will become in the online version or versions.

Information that looks the same in the book can have different functions in the online help, so it's important to examine not just the appearance of the book, but also its underlying structure. The glossary mentioned in "Presenting information customized for each delivery medium" on page 195 is a good example of this.

What information goes into the deliverables?

The first step is to identify your information types. Scriptorium Publishing generally uses four fairly simple ones (described in detail in "Different types of content" on page 83); other documentation groups may use fewer or more categories. Information types include:

- Interface information, which describes the user interface (the information the user sees on-screen)
- Conceptual information, which describes the how and why of a particular product
- Reference information, which provides information about the syntax of a system (for example, a list of commands and their arguments)
- Procedural information, which explains how to accomplish a particular task

For each information type, you need to identify which deliverables should include that information. For example, conceptual information should be in the print version but not in the online help. Users want specific task information from online help and from manuals, so procedural information would go into both the online help and the hardcopy books.

In addition to various types of information, you also have to consider navigational aids. These include the tables of contents, indexes, alphabetical listings, pagination, headers and footers, and any other information that helps readers find their way through a book or help file. Navigation is generally very different from one medium to another; for example, page numbers are a necessity in a hardcopy book, but they are of little use in online help.

A high-level planning grid

As you break down information into different categories, start looking at the high-level elements in the book and figure out what elements they will become in the help (Table 6).

Table 6: *High-level elements*

Element	Treatment
Table of contents	Generally available in book and help. Help may include more levels to make navigation easier.
Introduction	Usually conceptual; often eliminated from help using conditional text.
Glossary	Usually alphabetical in book; often provided as pop-ups from the words inside the topics in help.
Index	Automatically hyperlinked in the online version.
Context-sensitive interface help	Often, no interface information is provided in the book.
Section	Book sections become topics in the online help.

A paragraph-level planning grid

After assembling a grid for your high-level book elements, you can move on to paragraphs, graphics, tables, and other paragraph-level elements.

In a book, you may have a Body tag that serves more than one function. For example, it might be used for

body text but also for the definition of glossary terms. In a single-sourcing environment, you want a one-to-one correspondence between the paragraph styles and their functions, so you would probably need to create a GlossaryDefinition tag. For a print-only book, this is unnecessary, but if you plan to convert the book to online help, you need to be able to identify glossary text uniquely.

Headings generally indicate the start of help topics, so it's a good idea to make sure that the document is broken up into reasonably-sized sections.

In a book, you may have a See Also listing at the end of a section. In the online help, this might become a "Related Topics" list, or just a list of links.

Choosing single-sourcing tools

Choosing the right tool is as essential as planning a single-sourcing process. To ensure your process is as efficient as possible, a tool must provide particular features, as explained in the following sections.

Considerations for creating multiple versions of a document

When evaluating a tool for creating multiple versions of a document, consider how well it lets you complete the following tasks:

- Labeling text for selectively displaying content
- Assembling chunks of text into a document
- Defining placeholders

Labeling text for selectively displaying content

One way to specify what material goes in each version of the document is to label text so that you can selectively display it.

FrameMaker software does this very well with a feature called *conditional text.* You can show or hide text based on the tags you apply to content.

You could use conditional text to show and hide information about the three printer models described in Table 5 on page 198. According to that table, you would not need to apply a condition to the text about the 500-sheet and secondary paper drawers because that information applies to all models. However, you would need to apply a condition to the text about the duplex printing attachment so that it is displayed only in the book about Model C-3000, as shown in Figure 36 on page 204.

Another way to label text for selective display is through attributes that you assign to content. You can think of attributes as hidden labels writers use to organize their content; users of documents don't see them. See "Element attributes" on page 217 for more information about attributes.

Assembling chunks of text into a document

You can also control what information goes into each version of a document through the use of *modules*. Modules are chunks of information that you use to build a document; for example, one chunk for the printer documentation could explain how to attach the duplex printing attachment. When a particular chunk of information does not apply to a printer model, you would not include it in that model's documentation.

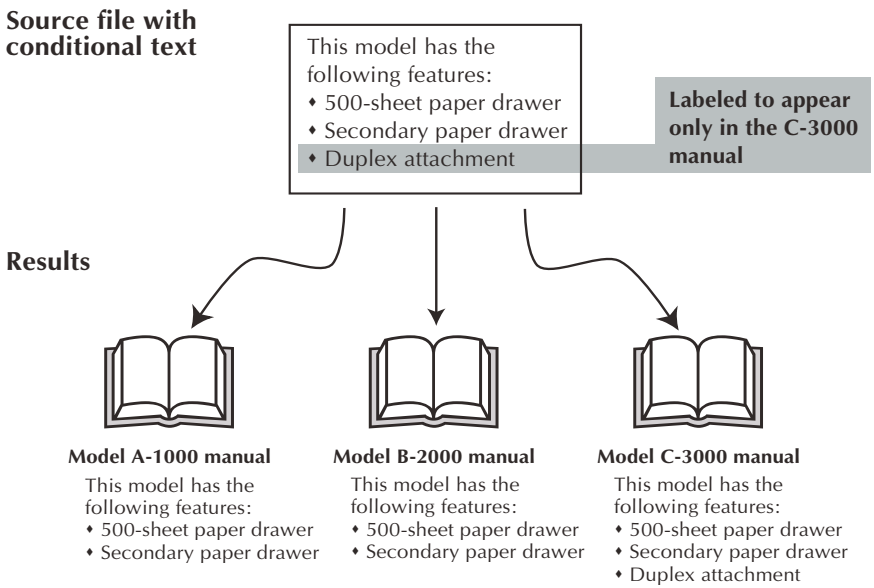

Figure 36: *Conditional text usage*

In addition to giving you the ability to control what information appears in a document, modular documentation also provides the following benefits:

- Easy reuse of content—Reusing and sharing information in modules eliminates rewriting, which wastes time and can introduce inconsistencies in terminology and presentation.
- Automatic updating of embedded modules—When you modify information in a module, it is automatically updated in the documents where it is embedded. Automatic updating eliminates the need for figuring out where modules are used in your

source files and making the same changes in the different locations. Making the change in a single location also ensures consistency.

You can find more information about creating modular documentation in *Single Sourcing: Building Modular Documentation* by Kurt Ament (ISBN 0815514913).

Defining placeholders

Placeholders let you easily update product names and other terminology in different versions of a document. Continuing with the example about the three printer models, you could create a Model placeholder in the source files to handle the three model numbers. (You would also need placeholders for any other features and terms that do not have the same name across all models.) You could then create three separate files that contain just the placeholder definitions for each model. Before creating the deliverable for each model, you would import the definitions into the source files (Figure 37 on page 206).

Considerations for creating multiple outputs from one document

When evaluating a tool for creating multiple outputs, a primary consideration is how much it automates the conversion process. For example, some tools require you to reapply some formatting in converted output, but other applications create output that requires minimal or no post-processing.

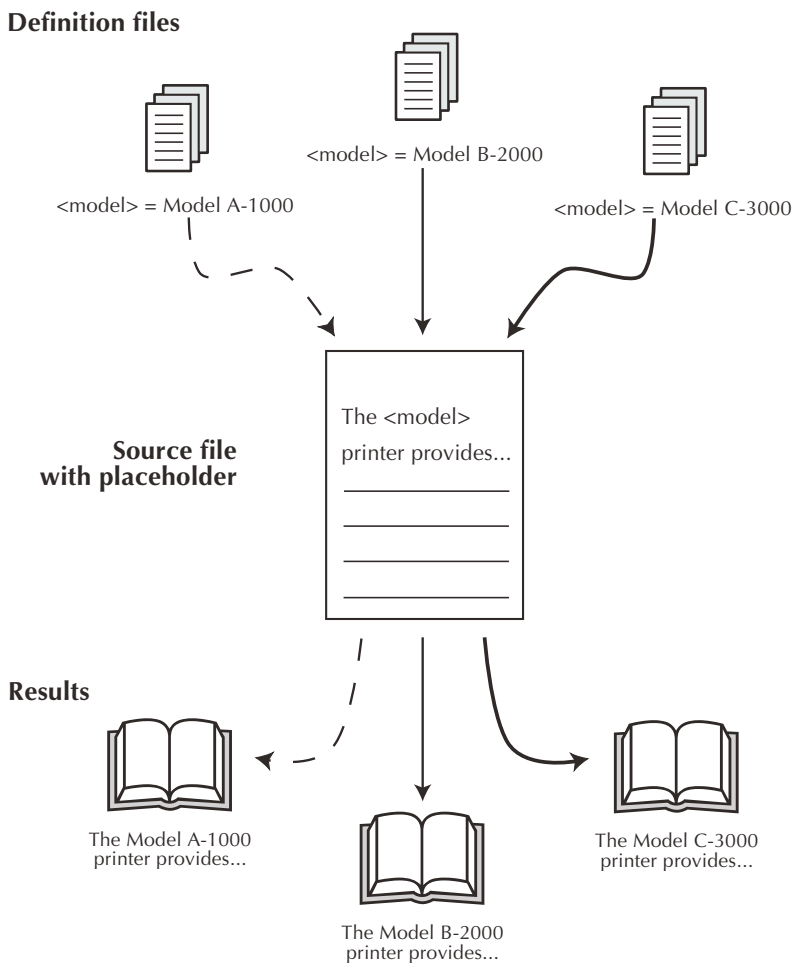

Figure 37: *Using placeholders*

Another factor to consider is how widely implemented the single-sourcing process will be—is it a small group of writers doing the conversions, or is it a solution that is implemented across a large department or company?

NOTE: As an entry-level writer, you probably won't be tasked with evaluating single-sourcing tools. In fact, you'll probably be told what tool to use—and exactly how to use it. However, if you're part of smaller department, you may find yourself in the position of choosing a tool and setting up the single-sourcing process.

Level of automation

Single-sourcing tools vary in how much they automate the process of creating online formats from document source files.

There are tools that automate a good portion of the conversion process but produce output that requires some post-processing clean up. These tools are fairly inexpensive, relatively easy to use, and do not require programming ability. In some cases, you have to save your source files to another format before importing them into the tool—which can remove some formatting and graphics that you have to put back in yourself.

Other tools that provide a higher degree of automation are a better choice. Often, these tools don't require you to reapply formatting or manually save your files to another format before conversion.

Some of these tools require a lot tagging and special formatting in the source files, and you need some low-level programming skills to set up the conversion process. Tools and technologies in this category include the following combinations:

- FrameMaker with MIF2GO
- FrameMaker or Microsoft Word with WebWorks Publisher

- Content based on Extensible Markup Language (XML) transformed into online formats with Extensible Stylesheet Language (XSL)[1]

These advanced tools and technologies provide the ability to set up totally automatic single sourcing. It is possible to build a process that requires writers to do little more than push a button to start conversion. Strict adherence to templates is required, and you may need the help of a consultant to create the templates and to set up the process. Figure 38 shows a single-sourcing workflow you can set up using advanced tools.

Another tool that offers a higher degree of automation is AuthorIT. Unlike WebWorks Publisher and MIF2GO, which convert source files created in another program, writers create the source files in AuthorIT and store them in a database that is part of the application. (The database also provides a way to control who has access to particular content and a way to track versions of a document.)

Using the product's interface, writers assemble a document from chunks of stored content and then specify what type of output they want—Word documents for printed documentation, HTML for web sites, and so on.

1. XML is a specification for storing structured information in a text format. XSL is a technology that transforms content based on XML into formats such as HTML and HTML Help. Chapter 14, "Structured authoring with XML—the next big thing," describes structured information, XML, and XSL.

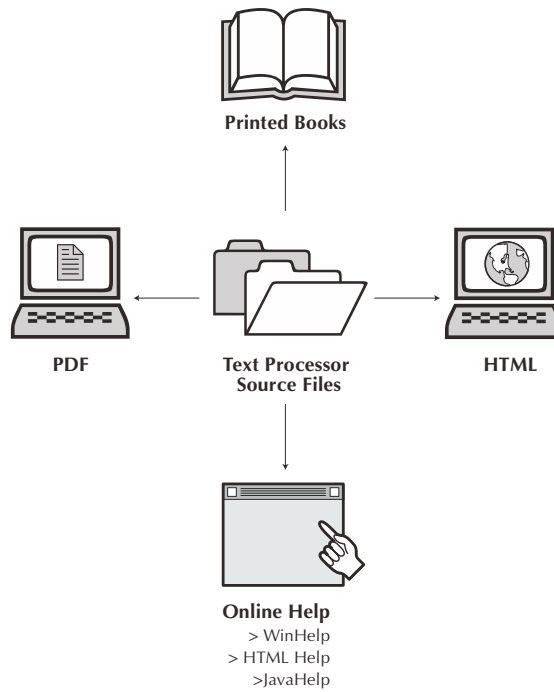

Figure 38: *Single-sourcing workflow*

The conversion process is push-button with AuthorIT, but someone must first set up the formatting standards for the content (just as you do in templates for WebWorks Publisher and MIF2GO, but less scripting is involved). Also, if your department uses AuthorIT, someone must handle the administration of the database (determining users' access to content, and so on).

In general, more automation requires higher software costs and a greater degree of complexity in initial setup, but a highly automated process requires writers to spend significantly less time converting files. So, when evaluating the cost of a single-sourcing process, don't focus on just the price of licensing the software. Be sure to also consider the following:

- Any start-up costs, particularly if you need help from a consultant in establishing the process and being trained on how to use it
- The time (and money) it takes authors to convert documents each time there is a documentation release

Level of implementation

Another factor affecting the cost and complexity of single sourcing is the level of implementation. Some smaller departments may handle conversion by using the applications described in the previous section; staffers use applications installed on their computers to create online output. The output is then stored on their machines or on a shared server. The content is distributed to users on CDs or as a help system integrated with the documented application itself. The output may also be stored on the company's web site so users can access it.

Larger companies, however, may set up what's called an enterprise-level solution. The cost of software and implementation can skyrocket to $50,000–$100,000 or more; the software can be highly customized versions of

existing single-sourcing applications or even custom-designed applications. Enterprise-level environments usually include significant amounts of networking and database storage capabilities. For example, writers working in different locations generate content that is stored in the same database.

These large-scale solutions are appropriate only for very large organizations, and their implementation requires the services of a consultant—heavy-duty programming and database skills are necessary. These custom-designed environments can provide repositories for information and the ability to output to many media, including on-demand distribution via the Internet. For example, a company's web site can be connected to a database housing the online output. The database displays specific content based on the search terms typed by a user.