

Developing a Single-Sourced Online Help System

Keith Vicek, Phil Menzies, André Evans

*The definition of single sourcing continues to broaden in scope since its first mention in *The Society of Technical Communication's 46th Annual Conference* publication. As a result, it is becoming increasingly difficult for technical communicators to understand what single source means and, more importantly, choose a definition of single sourcing that correlates with their specific task. One "type" of single sourcing involves reusing information for multiple products. Several developers at IBM have produced a single-source online help system. Unlike other single-sourcing methods that require a significant investment and a high degree of technical experience, these methods are inexpensive and require a moderate, yet creative, technical aptitude.*

A DEFINITION OVERVIEW

Content vs. Format. Before looking at specific definitions of single sourcing, perhaps it is better to view single sourcing within the context of an overarching understanding. In a 2000 article, Stewart Florsheim analyzes the various definitions associated with single sourcing and groups these definitions into two categories — format single sourcing and content single sourcing. He explains the aim of format single sourcing is to reuse information between alternative media, whereas content single sourcing is geared more toward generating and designing content for a particular medium.

Single Source vs. Output. One year before Florsheim's distinction between content and format single sourcing, Mark Baker wrote that the single source used to generate content should actually be derived from a different source than any of the expected outputs.

Defining a Single Source. In a series of collaborative articles, Dawn Stevens, Ann Rockley, and JoAnn Hackos developed two of the first definitions concerning single sourcing, which are somewhat less rigorous than Baker's definition. Though Rockley and Hackos agree that the source for containing content is typically a database, their definition for single sourcing is forgiving, as it is simply concerned with containing information in a single source. Unlike Baker's definition, Rockley and Hackos do not discriminate between the source and the output.

In addition, these three contributors are the first to expound on the definition by introducing guidelines and scenarios to assist technical communicators in evaluating a variety of situations where single sourcing is useful. Moreover, they address designing a single-source

approach and maintaining the solution. In this way, it can be said that they have contributed the most to this newly formed body of knowledge by attempting to explain its purpose and therein its value.

Help Files, Manuals, and Multiple Versions. In a February 2001 article, Philip Butland attempts to synthesize a single source definition in a two-part introduction to the topic. The first definition, which he writes is the most widely used, deals with generating manuals and help files from the same source. The second definition takes a more broad-brush approach and deals with using content "to produce multiple versions in any medium." An example of the latter is to reuse content found across multiple products.

Definition Summary. As the single-source landscape begins to mature and change, so does its definition, illuminating its viability as a credible body of knowledge while at the same time exposing its gaps. Although the past four years have identified unmovable characteristics exhibited by single sourcing, the details are volatile. These small shifts make it increasingly difficult for technical communicators to understand the various aspects of single sourcing and determine which aspects pertain to their particular needs. Therefore, it is difficult to implement this important approach.

Those interested in developing a single-sourced online help system for multiple products are better served to understand the various definitions that exist and combine pieces of those definitions in order to situate a single-source approach within the genre. For example, Florsheim's content single sourcing is extremely useful for this task as it deals with choosing a particular medium — in this case, a help system. One approach to single sourcing an online help system is to store files in a type of "electronic library," which correlates with Baker's contention that the source should be kept separate from the output. However, in the case of generating information for multiple products that have different "flavors," sometimes the output *is* the single source because each version of the help system serves as a source, or a base, for each subsequent version. In this case, definitions set forth by Stevens, Rockley, and Hackos are most appropriate. Finally, Butland's broader definition allows for common information to be shared between similar products.

So, a neat definition and tidy solution explaining how to develop a single-sourced online help system for multiple products does not exist. Answers do exist in the larger picture, however. In a strange way, sifting through the

data is exciting as it substantiates the idea that single sourcing, once seen as a fad, is emerging in different directions, taking root, and evolving in ways once unimaginable.

TEN EASY STEPS

It is important to note that the purpose of the following steps is to provide a guideline or framework by which other practitioners can develop their own single-source approach.

Step 1: Identifying Topics and Procedures. More than likely, the help system is designed to work in conjunction with other types of documentation that accompany the product. The documentation should not compete with each other. Instead, the different types of documentation, including the help system, should work together to form a complete information package, each developed with the audience in mind.

The best way to accomplish this segmentation of information is to perform a task analysis on the complete product. Then, once all appropriate tasks have been identified, separate the tasks into groups that fit the appropriate medium and audience requirements. Most of your segmentation will be based on Legal requirements, safety requirements, and just plain common sense. For example, some countries require that warranties be in hardcopy format. Information about the initial setup of a computer is best put in hardcopy format, either a Setup Guide or Setup Sheet, as opposed to an online presentation that the user cannot access until the computer is already set up. Any redundant information that you include should be by design based on user needs.

Once you have grouped the tasks by medium, it is time to start structuring your help system. When structuring a help system for multiple products, it is important that you mentally separate the task from the process whereby the task is accomplished; then, identify which tasks are common for all products and which tasks are unique to specific products. Use an outline to arrange your tasks in a hierarchical order. In most cases, the task itself will become the topic heading, so refine the task names so they reflect a consistent presentation.

Now you can expand your outline to take into account specific procedures and variations. If any tasks are unique to certain products, make a notation in the outline of the product names. If tasks are common, but the procedures for accomplishing the tasks vary by product, make a notation in the outline of how many “flavors” of a task need to be documented. This outline will become the skeleton of your help system.

Step 2: Structuring Modules. Your enhanced outline has all of the information required to structure

modules. Modules can be as large as a major topic or as small as a specific procedure, depending on the level of granularity you want to achieve. This framework for developing an online help system turns each information module into an HTML file and then groups HTML files into compiled help modules, or CHM files. By increasing the number of CHM files that make up the help system, you can increase the amount of information that is reused and decrease the amount of time required to modify and test the help system. The concept is to be able to “unplug” modules, replace modules with different “flavor” modules, or add new modules without affecting the basic structure of the help system.

There are three key concepts about CHM files to keep in mind: (1) The table of contents entries and index entries travel with the individual CHM file; therefore, same-named CHM files can be plugged in or unplugged without recompiling. (2) Not every CHM file defined in a project needs to be present; therefore a “light” version of the help system can be created by removing specific CHM files, or placeholders can be added for future expansion or updates. (3) External programs that are capable of detecting certain hardware or software features can move CHM files into and out of specific folders to create custom help systems that match a specific computer configuration. With this in mind, you can determine the granularity required for your help system and assign CHM names and HTML file names to each item in your outline.

Step 3: Designing the Database. This approach uses a database in an unconventional way in that the database is not used to store individual HTML files. Instead, it is used to track information about the various versions of the help files that have already been developed and exist in a library or are being developed and will be placed in a library.

Note: This step can be done in parallel with either of the earlier steps.

The key to building a truly modular and flexible help system is to create a basic structure that does not change even though the information does change. In order to accomplish this implementation, CHM names and HTML file names must remain stable, but some method of differentiating the various “flavors” of same-named HTML files must be implemented. In our case, we used an alphanumeric alias that we refer to as an identification code. This identification code appears at the bottom-right corner of each HTML file, is recorded in a master database that contains information about each HTML file, and is also used in the master library as the name of the folder that contains the HTML file. The identification code is in the format of X-XXXX-XX-XX-XX, where:

- The first character identifies the master library where the files are stored.

- The second grouping (4 characters) is the topic-identification code.
- The third grouping (2 characters) identifies the version level. Based on the products being supported, there might be several versions of a file. For example, if some products run diagnostics from a CD, while others run diagnostics from the hard disk, two flavors of the “Running diagnostic programs” topic need to be developed. The file names remain the same, and the four-digit topic identification code remain the same, but the flavor code changes.
- The fourth grouping (2 characters) identifies the revision level. For example, if a phone number or external URL changed, and the file remained compatible for other projects, the revision level is incremented.
- The fifth grouping (2 or 3 characters) identifies the language (ENG, FR, GR, IT, and so on).

You can use any off-the-shelf database program. We used Lotus Approach and defined fields for master library, topic identification code, version, revision, file name, CHM name, where used, general description, version/revision unique description, projects where used, button launches, linked to, linked from, open defects, closed defects, and languages. The database enables us to assign and maintain unique identification codes for each HTML file, track defects, and identify the differences among same-named files.

Step 4: Creating the Roadmap. The roadmap is an essential part of our development process. It is the next logical step after creating your enhanced outline. Not only does it act as a guide for the writers, it also provides a status and history, and is also critical in the testing phase. We set up our roadmap in a table format using a word processing program. The following describes each column in the table:

- **CHM name:** This column contains full names of the CHM files.
- **File name:** This column contains full names of the HHC, HHK, and HTM files used in the CHM.
- **Title:** This column contains full titles of each HTM file.
- **Previous product identification codes:** This column has content only if you are basing a new help system off an existing help system. This column contains the identification codes for each HTM file used in the previous help system.
- **New product identification codes:** This column contains the new identification codes for each HTM file used in the new help system. If a topic is reused without change, the same number appears in this column as in the "Previous product identification" column. If a topic from the previous product help system is

modified for the new help system, only the version or revision code changes. If a new topic is added, a completely new identification code is used. If a topic from a previous help system is not used, “NA” is placed in this column.

- **Description of change:** This column provides a description of the changes required for each HTM, HHK, or HHC file. If an HTM file is picked up from an existing help system that does not require change, put "NO CHANGE" in this column. A date is also helpful whenever text in this column is changed or updated. A "Done: MM/DD Writer's name" entry is also helpful when the modifications are made to help you track what is complete, when the work was completed, and by whom. If a topic from a previous help system will not be used, "Not used by this product" is placed in this column.
- **Component owner:** This column contains the name of the programmer, engineer, or other contact who is responsible for providing or reviewing the content. This information is used in several ways. First, it provides a contact with whom the writer needs to work to develop the content. Next, when the help system or portions of the help system are sent out for review, it helps the reviewers identify those topics for which they are responsible. After a topic is reviewed and approved by the responsible parties, put a date in this field next to each name.

We also incorporated a color-coding scheme to help identify changes and unused topics. Any row that contains a change is blue. Any row that contains an unused topic is green. Any notes to the writers or developers are in red.

Step 5: Forming the “War Room.”

Communication is the key in a team environment and there is no substitute for proximity. If possible, arrange for an open area (war room) where all team members are within speaking distance of each other. When developing a brand new help system, impromptu decisions need to be made on the fly with buy in from all team members. If setting up a physical war room is impossible, you can set up a virtual war room by deciding on the best methods of communicating information (daily meetings, instant messenger, phone conferences, chat room, and so on).

Set up a shared disk where all team members can store their files, and set up a skeleton file structure that meets your needs. If you intend to use a database for control, make sure the database is ready for data input. Make sure you have adequate security in place to protect the files against unauthorized access. Put a backup scheme in place for the shared disk and make sure you use virus-protection software on each workstation and on the shared disk. Make sure all workstations have the same

version of whatever software you select to do your development. In many cases, files created in a later version cannot be opened or modified with an earlier version.

When the team is ready to begin work, take the time to gather the team members together for an orientation. Make sure all members understand their roles, the tools to be used, and how files are to be stored on the shared disk. Show a sample topic file that contains as many elements as possible and make sure all members understand the conventions to be used (color, font formats, style sheets, file-name conventions, numbering conventions (aliases), and so on). Once everyone is on the same page, you are ready to begin.

Step 6: Creating the Help Files. In order to ensure consistency among the help files and to avoid problems with fonts in non-English languages, we use a cascaded style sheet to define fonts, lists, background, and other characteristics. Depending on the resources available, you can split up the development of the individual HTML files to any number of team members. Depending on your help-authoring tool you might have to clean up some code manually in a text editor program. Our team used the RoboHELP HTML help-authoring tool for the initial development of the help files, but had to manually clean up the HTML code produced by RoboHELP to meet IBM translation-center standards. Our team used Microsoft HTML Workshop for the final compiling.

During development, our team used a "phantom CHM" approach to allow for variable information and future pluggable updates. The compiled HTML architecture enabled us to define more CHM files in the "Merge Files" section of the HHP file than we were actually using. And, by using a master HHC file for all CHMs, we were able to build in hooks to the HHC and HHK files from the "phantom CHMs." The end result enabled us to build a help system where certain modules could be added or removed during installation based on detection code, or updated at a later date with new information without disturbing the integrity of the help system.

Step 7: Editing. One of our team members is an editor who also possesses compiled HTML development experience. All team members funneled their respective help files through the editor, who is able to spot inconsistencies or problems and correct them real time, enhance indexes, and ensure that the individual HHC files merged correctly to make up the master table of contents. Being in a physical "war room" environment is a big advantage because the editor is able to work with the individual writers to get answers about content and also disseminate information about inconsistencies so the writers could address these in any future work or work in progress. The editor is also responsible for final compiling.

Step 8: Making the TOC and Index.

1. Making the TOC. Because same-named CHMs would constitute all help systems, a master TOC in the form of an HHC file enabled us to include or exclude topics without affecting the rest of the help system. The master HHC includes only "calls" to other CHMs. If one of those CHMs is not present in the final project file, the beauty of compiled help becomes evident: the files present ignore the missing file and the remaining files are presented in the contents.

The master TOC, or HHC, can be constructed in a couple ways. The master HHC file can point to the first level topics only and the first level TOCs can point to subtopics developed as CHMs or the Master HHC can provide the structure for first-, second-, and third-level topics and subtopics as well. Again, the level of granularity will help dictate the HHC style. The key, of course, is using similarly structured material and same-named CHM and HHC files, which enable developers to plug in and unplug whole sections cleanly and simply.

Because any CHM or "subCHM" might be removed, depending on the documentation requirements of the product, interCHM links must be carefully considered and tracked for later testing. Although CHMs are added and removed with relative ease, links to material that has been removed can remain and lead to "page not found" errors.

2. Making the index. Creating an index for a unified volume for which the manuscript is already prepared can be a daunting task. Creating an index to several hundred HTML topics that are divided among twenty CHMs, while the topics and CHMs are being developed, is a challenge indeed. The tools for building CHMs (RoboHELP HTML and HTML Help Workshop, for example), enable developers to create index entries at the topic level as topics are being created, which has a couple of negative consequences: Index entries without consistency or appropriate subordination proliferate; and different developers might apply various styles in preparing the index on the fly.

Knowing the behavior of index, or HHK files, enables a master index to be developed following topic development. Build the master index according to main and subordinate entries, and then use the tools to point to the topics that are included in the help system. The key concept here is this: You can have all the index entries specified in a unified index file; however, only those entries that culminate in a call to an HTML file will appear in a given index.

Step 9: Compiling. Compiling successfully twenty projects that call and point to each other properly is a culmination of maintaining topic file names, CHM names, HHC, and HHK file names. The key is knowing that each CHM must respond to two HHCs or contents files: the internal, or "chapter" contents file and the

project, or master, contents files. Both must be specified and available for the CHMs to appear seamlessly as a single, unified help system. Furthermore, all window definitions must be set exactly the same, otherwise a user who moves from one CHM to another will experience errors and multiple open windows. It is essential that every project share the same master HHC file, which gives the whole help system the overall structure. If a single CHM has a downlevel or different master HHC from the CHMs, unexpected errors that are difficult to trace will ensue.

Step 10: Testing for Functionality. Help systems have always been problematic to test. First, how can you be sure you have the correct version of topics? Second, how can you be sure that each topic has been fully checked for links, application launches, and Web launches? Our solution involves using the full-text search capability built into the compiled HTML help system in conjunction with the roadmap created during development and the identification codes contained in the HTML files. Because the roadmap contains a complete listing of all HTML files used in the help system, both by name and identification code, the roadmap is used as a guide to systematically locate each HTML file and verify the version, revision level, and language.

The testers use the built-in search feature to search for the 4-digit topic-identification code instead of trying to use a table of contents that might not have every topic listed. This is especially useful when testing help systems that are not in your native language. Once the topic is located, the tester scrolls to the bottom of the topic, verifies the language, and compares the version and revision level to the roadmap. The testers then perform quick visual inspection to ensure there are no formatting problems. Finally, the testers systematically click each functional object (link, launch button, or Web link) to verify each function is working correctly. After clicking a functional object, the tester returns to the topic being tested and moves to the next functional object. When all functional objects have been tested, the tester moves on to the next topic listed in the roadmap. All discrepancies are recorded (including the identification number and a description of the failure) and returned to the developers for investigation and correction. After making corrections, a regression test is performed on the topics that contained discrepancies.

TOOLS USED

Details on using RoboHELP HTML and HTML Help Workshop are beyond the scope of this paper. However there are numerous tools that enabled us to automate and streamline procedures. Correcting hundreds—even thousands of HTML files—processing and viewing twenty CHMs per language can mean a lot of pointing

and clicking unless you get serious about recording your actions in simple tools: batch files, an external search-and-replace utility, and a utility that gathers and saves as text filenames within a folder.

- As stated earlier, it is essential that each internal CHM share the same project-level HHC. Pointing, clicking, and pasting a recently changed HHC into hundreds of CHM project folders is tedious and error prone. Employ the talent of the computer to do the tedious and repetitive. A batch file rapidly and completely can distribute a changed master HHC to all constituent files.
- HTML Workshop provides batch-file commands, that enable developers to compile numerous projects with a double-click. Furthermore, after compiling, batch files can copy the resulting CHMs into another folder so that you can view the associated CHMS.
- A file utility called Tree Print facilitates capturing file listings and importing listings into batch files with the precision that such files demand. Combining Tree Print output with spreadsheets is an effective way to write numerous batch files by copying paths and file names into fields. After the paths and file names are pasted into their respected columns, copy the spreadsheet into a text file, remove the unnecessary space, and you have a precision batch file, made with few keystrokes.
- Batch files can also move hundreds of files from one location to another with precision and speed. Such capability enables moving hundreds of translated files into appropriate project files with confidence.
- An external search-and-replace utility enables writers and developers to make changes to links and launch buttons even after hundreds of files have been developed.

LESSONS LEARNED

Lesson 1: One large benefit in designing an online system this way is that the developer has the ability to build the help system in several languages. The identification code found in each HTML file is invaluable in this respect. Naturally, each language requires a high degree of testing, ensuring that the links and launches go to the correct destination. In short, each language requires a fair amount of time to develop and test.

The process works well as long as the product is announced at staged intervals throughout the world, or in waves. It becomes significantly problematic to have all of the languages built and test when the product is announced throughout the world simultaneously. This requires sending files to translation as soon as the developmental cycle is over, leaving a small margin for error. Further, when the files return from translation, it is important to establish a seamless integration and

compiling process to eliminate errors in the foreign language, resulting in less time spent fixing errors.

Lesson 2: Although we would like to say that the entire process as documented was established before we started our first project, it was not. Even with careful planning, we ran into unforeseen problems. The process as documented evolved over a period of time, during the development of several projects. In some areas, the process was a result of trial and error. In other areas, the process was a result of new requirements, better familiarity with the tools, or just a new idea or approach. It is important to take an adaptive approach when implementing a process. Keep track of any shortcomings and refine or change your process in a timely fashion.

Lesson 3: Do not take shortcuts in the up-front planning. Take the time to do a task analysis and understand your audience. Develop an outline and expand it into a CHM/HTML hierarchical roadmap that your team members can follow. Use every bit of knowledge that you have for designing good hardcopy information and apply it to your help system; then, examine it again. Segmentation of information and structural hierarchy are even more critical in online information than hardcopy.

Lesson 4: If you have multiple entry points into your help system from an external source, maintain a master alphabetical list of all CHM files that are used in your help system so your team members can copy and paste it into the Merge Files section of their respective HHP files. This eliminates problems with topics not found when your help system is opened by an external source.

Lesson 5: If you will be working with people in other countries, understand the tool availability in those countries. For example, RoboHelp typically has a several month lag between the time a new version is available in the United States and when a Japanese version will be available in Japan. Some other Help development tools might not ever be made available in some countries or languages. In some cases, it might be smarter not to use a brand new release of a product.

REFERENCES

- (1) Baker, Mark. "Designing an Information Set for Single Sourcing." *Society for Technical Communication Proceedings*, 46th Annual Conference publication, 1999, p. 374
- (2) Butland, Philip. "Introduction to Single Sourcing." *Society for Technical Communication Intercom*, February 2001, pp. 22-27

- (3) Florsheim, Stewart. "Designing for Single Sourcing." *Society for Technical Communication Proceedings*, 47th Annual Conference publication, 2000, p. 83
- (4) Rockley, Ann and Dawn Stevens. "Creating Single-Sourced Information Products." *Society for Technical Communication Proceedings*, 46th Annual Conference publication, 1999, pp. 267-269
- (5) Rockley, Ann and JoAnn Hackos. "Designing Single Source Materials." *Society for Technical Communication Proceedings*, 46th Annual Conference publication, 1999, pp. 53-55

Keith Vicek
Senior Software Engineer
IBM
3039 Corwallis Road
Research Triangle Park, NC, 27709
919-363-0615

Keith Vicek is a Senior Software Engineer at IBM in Research Triangle Park, NC. Keith joined IBM in 1976 and has focused on hardcopy and online information development since 1983.

Phil Menzies
Staff Software Engineer
IBM
3039 Corwallis Road
Research Triangle Park, NC, 27709
919-543-0245

Phil Menzies is a staff software engineer at IBM in Research Triangle Park, NC. Prior to IBM, he worked for many years in textbook publishing and started his own electronic book-production company

André Evans
Software Engineer
IBM
3039 Corwallis Road
Research Triangle Park, NC, 27709
919-543-2536

André Evans is a software engineer at IBM in Research Triangle Park, NC. He is the past-president of the Southwest Missouri State University STC Chapter where he received a Master of Arts in Writing. Currently, he is a member of the STC Carolina Chapter.