Documentation teams traditionally have not written software or developed customized software tools. Many companies may consider those skills too specialized to relegate to writing groups, too costly, or do not see the advantages of automation. Many teams themselves just have never thought about programming, thinking such an effort may be out of scope or that the team does not have the proper skills. However, in this age of automation and convenient data access, there are significant gains to be made (such as better, cheaper, or faster in any sense of those words) by automation in support of documentation. To help you into this frontier, I've got a few suggestions based on my documentation management experience along with the experience from peers.

In the broadest sense, automation is the programmatic collecting, formatting, or checking of data. It aims to reduce the manual effort or replace repetitive tasks with predictable and repeatable applications or scripts. Automation can take many forms along a continuum, from the simplistic use of the Microsoft Word's F4 key to repeat the last command to the full development of a conventional software application. In practice, documentation automation tends to falls more on the simple end of that continuum. The development effort ranges widely too. You can pick the level of involvement that is right for your team. Certainly, groups with formal programming experience (such as those with programmer-writers) have an advantage but not a lock. Reliance on simple methods of automation such as macros, for instance, still has benefits. For example, procedures that are thought to be one time events, rarely are. If you perform a similar task twice, consider automating it or storing it as a macro (which stores the all the keyboard, mouse, and other input actions as a single keystroke).

Automation is not about writing process itself but rather it supports the process by getting or manipulating the information for you. The point is to get away from performing tasks manually. There are as many applications of automation as there are technical communicators and problems they have to overcome. As an example, assume a substantial amount of information is kept in your company's database. In this case say it's a long list of your product's error messages that needs to be included in your help file. There are four gains automation could provide.

- First, a simple Visual Basic application can retrieve that information. This is for situations in which the company does not have a front-end interface. Even if they do, often that interface does not support exactly what you need it to and as a result, the writer's workaround is still substantial. For example, what you end up with may be a long list, perhaps in a plain text file or as a spreadsheet. In this case, getting the information is may be only a small step in the overall process. Formatting still needs to be applied.
- Second, the application can format the returned data to fit your needs and it's likely this formatting is the most amount of time and the largest source of the quality control problems. The Visual Basic front end, in addition to retrieving the information, could format it with HTML, XML, or any other format after it's retrieved; an ASP script could format it as it's retrieved. A Visual Basic for Applications (VBA) could write it directly as a styled Word file. Regardless, if the database is large enough, this could be a significant savings and is a noticeable improvement over conventional hand coding. It also fulfills the better, cheaper, faster mantra. The text formatted through automation is usually a better and more consistent quality.

- Third, the process is repeatable. You can generate the same information as many times as needed, which makes it convenient for routine updates or repeating the process to get it formatting exactly right. Again, better, cheaper, faster.
- Fourth, by slightly modifying the code, the application itself could be reused for similar situations. This is the real advantage to automation since it can be applied to other situations easily. Although each group has its work practices, perhaps unique to itself, each group also tends to follow patterns and trends for themselves. Leveraging one solution to another problem is a highly effective way to customize solutions.

Automation is pervasive in the marketplace and many of us already have access to it. The most common example has to be Microsoft Word. Word is extensively customizable and programmable. It is customizable in that virtually all the front end features can either be moved, renamed, automated, or modified. Moving a menu item to a new location can hardly be called automation but it is an important step. Reducing the number of keystrokes or mouse movements to get to an action introduces the concept of easy and repeatable functions. Word is programmable in that all the inner workings are exposed to direct manipulation through code. Again, there is a continuum. At one end, you can use macros to encapsulate a series of steps or an entire complicated procedure and the user does not have to know programming or to look at code. Moving along the continuum, users can directly modify the macro's code, if only to change a file name or drive location. At the other end is the full programmatic involvement. Word has a secret life in that it has a complete development environment and debugger built into it. Microsoft went one step farther in that they supported these capabilities in all their Office applications. By exposing most of the internal features and functions programmatically, languages such as Visual Basic, Visual Basic for Applications (VBA), .NET languages (C#, J#), Javascript, and Java can all access these functions. This opened architecture can be seen most recently by their reclassifying Office from a tool suite to a "system."

For as popular as Word and Visual Basic are, they are by no means the only applications supporting automation. Most applications (including those from Macromedia, Adobe, and Microsoft) include an API (application programming interface), which is the set of functions that can be accessed programmatically. In turn, the applications often support others' APIs.

Although the thought of programming may seem out of scope and experience for writing groups, most teams can benefit from some level of automation, if the project is planned appropriately. The surest way to fail here is to over estimate your capabilities and over promise your results. In short, keep the projects as simple as necessary to get one task done. When planning these projects, keep the following issues in mind.

Solve One Problem At a Time

The scope of the solution is to address one and only one problem. Whether it's getting information from a database, checking the styles within a Word document, or generating pages in HTML, limit the scope. Don't combine solutions until you're more comfortable with automation concepts and implementation.

Don't Write Robust Tools

You’re writing scripts for a one-time, specific purpose and scope. Therefore, you don’t need extensive error checking or online help. A common time sink is trying to make the tool bulletproof. The audience may be equally limited, such as one specific writer or editor. In that case, you can specify exactly how to run the code or what parameters to enforce. Again, the point is to get a specific task done.

Tools have a limited lifespan

Because the tool is designed to do one specific task, it may have a short lifespan. That’s alright. Besides, the good and useful tools will evolve over time. The concept is that you’re reusing the code, and not the tools.

Reuse The Code, Not The Tools

For each new solution create a new tool. Don’t keep adding on to the same tool. This will only make it unusable for any purpose. Certainly base the new tool on an existing one that does a similar task but always start from a copy and modify it. Over time you will end up with a large collection of tools, applications, and scripts. But think of this collection not as a tool library but as an example library. Reuse portions of the code for other projects. For example, code that opens different files in a single directory or writes to an external file will likely be needed at other times. You will get speed and proficiency gains over time from familiarity of the code.

You understand your work process better than anyone else does

Identifying your unique needs and similarities among your processes expertly locates the areas to concentrate on. Since the project will be under the writing manager’s direction and approval, it is often easier, quicker, and more efficient than by soliciting outside assistance.

Understand what is possible with your writing tools

Visual Basic and VBA capabilities access virtually any aspect of a document, Word or otherwise. You may not use all the capabilities at first, but knowing what is possible will encourage you to push your solutions.

Keep Focused

When you want to hang a picture, a hammer is not a solution, a nail in the wall is. The same thing applies to automation. You will not be judged by the quality or the elegance of the tool; chances are, they may not even know about them. Rather, you will be judged on the document’s conformity to requirements. In other words automation is a means to an end, not the end itself.