Creating Automation Tools for Writing Teams: A Programmer's Perspective

shutterstock.com/ Aha-Soft

By ROBERT DELWOOD | *STC Senior Member*

TOOLS WILL ALWAYS be the weak link for writers. Not the high-level ones such as Flare or Framemaker, but low-level tools, ones that writers need to finish individual tasks. These tasks may be unique to each group or perhaps to each writer, from reformatting long lists of error codes, to collecting acronyms within a document, to special or conditional formatting on tables, or repeating onerous sequences—a representative list is impossible to create. Given the extreme diversity among writer's procedures, it is unlikely we will get a comprehensive tool suite. Neither the commercial sector nor the writing community at large provides these suites. That means we have to write them ourselves.

We've always been told that the goal is never far off. Many high-level tools have at least some support for this. Notably, Microsoft Word has a built in macro language (VBA), a macro recorder, a code editor, and complete programmatic access to its commands. These tools, though, are still very much in a programmer's realm, out of reach for many writers, and development groups rarely loan out programmers. But what if they did? What results are possible?

I was fortunate to be assigned to three writing groups at the NASA Johnson Space Center in Houston for a number of years. The mission was simple: to improve these teams' efficiency. I had considerable leeway in my methods, but mostly through the creation of custom tools and procedural changes. My qualifications seemed fitting. I was a programmer, writer, and programmer-writer, and thus familiar with writers and the writing process. The teams were grateful for the attention and willing to help although there were several conditions that had to be addressed.

Initial mistrust. There was serious mistrust about automation, with writers imagining that it would eliminate their jobs. The truth is, it's about freeing team members to do what they're good at. Start on a project they are comfortable with and earn their trust. Let them set requirements, provide frequent product updates, show progress, and offer options to alleviate those initial fears.

User education. Most team members are Word power users and capable of solving Word problems themselves. Automation doesn't minimize their skills. Quite the opposite; they still have to tell you to construct the documents and requirements. Over time, they'll learn what is possible with automation. The best ideas come from the team members themselves, but they have to know what can be automated and they have to be comfortable enough to ask. My favorite part of any project is when writers start asking, "Is this possible?" or "Can we do this?"

Bureaucracy. NASA is a government agency and despite the perception of being innovative, it's ultimately still a bureaucracy. Combined with other factors such as the reliance on international partners, interdepartmental coordination, and natural resistance to change, it was not always possible to make the right change. We had to work within what we could change. Don't be afraid to compromise or, conversely, overstep boundaries a little as needed. Remember, it's easier to keep a project that exists, rather than sell just an idea for a new one.

Development. Typical writer's automation projects tend not to be large or take a long time, perhaps two or three weeks. They have very specific goals, often just a single task. One project took less than two hours. In a way, it may not be much more than rapid prototyping. Working directly with the clients provides immediate feedback and shortens development time.

Once completed, you deploy projects in one of two ways. If the project is simple enough, it can be written as a series of macros and deployed through a common template. Otherwise, it will be deployed as a Windows application. This adds an installation step, but the application benefits from .NET features, making it more versatile. Web deployment was not an option for us because Microsoft Office then couldn't be used over an intranet.

What Can Be Automated?

I advocate that almost every team can use a programmer-writer. The team may not even know it, but after pointing out the opportunities, tools become indispensable. Think about your procedures and which ones are repeated, either immediately or periodically, such as before releases. Repetitious steps are obvious candidates for automation. Some are less obvious. For example, Flare truncates bookmark names to eight letters. As a result, sections such as "Installing System Drivers" and "Installing System Tools" display as "Installi" and "Installi2." This truncation is annoying and unhelpful. You can manually change these one by one, but an automated tool would fix all the occurrences in only a few seconds. You could also, for example, use a building block or snippet to create a

preformatted table, but over time the formatting might change. Instead, you could write a macro ensuring the format, from the table headers to the font and size for each cell, is correct. If you run that macro on dozens of tables before each release, the time and quality savings become significant.

Experienced programmer-writers recognize these opportunities. They can help point out what is and isn't automatable, and they often know procedural changes to accompany the automation.

The following are three representative but diverse projects.

TIFF-based Application

This is an example of controlling two applications, merging data from one to another. The first document is an enumerated list of parts and equipment going up to the space station. The restrictions were to create a computer-generated RFT file that couldn't be handled like a normal Word document; it had to be read only, and each page had to paste individually into a master, or target document, complete with table titles and body text provided by engineers. The conventional way of producing this was to save the list document as a single TIFF image file. Each page was then hand copied from Photoshop, pasted, and resized into the target. The list document was often more than 80 pages, and the handbook rated this procedure at 45 hours. In other words, it took one person more than a week of repetitious, error prone processing per iteration, and they may have had five to eight documents a year. In addition, a quality assurance (QA) team member checked each page, ensuring completeness and no duplicates.

This was an ideal automation project. The steps were well defined, there were no exceptions to the business logic, and it had a repetitive nature. Deployed as a Windows application, the team member started by selecting the list document, and placing the cursor in the target document where the first image was to be placed. The application would then image the list document. Because it was an RTF file, page breaks determined individual images. It generated each page as a separate TIFF file. When the images were complete, each file was opened automatically, and the image was copied and pasted at the insertion point in the target. The application also automatically resized each image for the available page, determining page dimensions and allowing for titles. The insertion point automatically advanced to the next page and the process repeated. The document completed in less than 15 minutes. After validating the process for a few weeks, they dropped the quality review requirement, saving additional time.

Acronyms

The most onerous requirement was that each document needed an acronym list. This list included identifying acronyms, cataloging them (complete with the definition), spelling them out on first occurrence, and providing a revision table to show newly added, deleted, or changed terms. Desktop guides rated this task as 32 hours per document, although it often took longer, sometimes more than 40 hours. Over time, the number of flights to the space station increased from 3 to over 15 a year, and bureaucracies being what they are, this meant that the documentation increased at a disproportionately high rate. Therefore, it was unreasonable to expect this process to remain a manual effort.

The difficulty started by just identifying what an acronym was. It could actually be anything from a three-letter acronym (USA or ARS [Air Revitalization System]), to terms (Log [Logistics], ATT [Attitude]), or almost anything else (rack loc [rack location], Rqmt [Requirement], or *U.S. Lab*). The approach had been simply looking at each page, usually printed, and using the reviewer's skill to spot acronyms. It was easy to assume the Find dialog could have been used, but reviewers still had to copy and paste each term, and the list approached 2,500 terms, double that if you included the definition. And that only covered the known terms. Engineers could introduce new terms without notification. There was also spelling out the term on the first occurrence, which became increasingly frustrating since material was often moved. The definitions had slight variations that had to be corrected (Heat Rejection Subsystem/Heat Rejection System, or Subelement/Sub-Element). As a result, automation had to be used, not only for quality control but, more critically, to keep up with the increasing workload.

The automation occurred in three steps because each step required verification and validation. The first step found and listed all the acronyms. Ultimately, we defined three ways to find them. What was called the "master list" was a complete term listing of 2,500 terms. It continues to grow and is controlled by select book managers. The application searches for these terms explicitly first. Then it searches the existing document's acronym list, assuming that those terms were already correct. Finally, we created a catchall using Word's internal misspelling dictionary. This identified all other terms, assuming an acronym shows up as a misspelling. For that list, we reviewed each term, adding the acronyms as needed. We managed those terms using Word's dictionary and exclusion dictionary (such as for people's names like *Sufferdini*). Some acronyms were also legitimate words (like *temp* for temporary, or *He* for Helium). Unless those terms were in the master list or the document's current acronym list, they had to be found manually. Regardless, we achieved nearly 100% accuracy.

The second step produced the acronym list as Word table, complete with revision tracks and style formatting applied. Upon approval, it was this table that was pasted into the target document.

The third step created a first occurrences report that team members used to verify that the term was spelled correctly and fully on first use.

What Is VBA?

VBA is the programming language Visual Basic for Applications. It is form of Basic and is widely held to be an easy language. It reads like English and can be programmed with less difficulty than other languages, like C or C#. VBA is not a modern language like Microsoft .NET, being a predecessor to VB.NET, and is limited in some advanced functions.

However, VBA has the advantages of still being a versatile language and, most notably, it's built into all Microsoft Office applications, along with a code editor. This implies two things. First, it supports Microsoft's OLE Automation, which means it interacts with all Office applications. Second, it is an interpreted language rather than compiled. VBA has to be hosted inside an application, typically Office applications like Word, and can't be used as a standalone application or as an *.exe file.

You don't need to buy a compiler or editor for it, because it's fully incorporated into Office applications automatically. The macro tool—the one that lets you record macros as a sequence of commands—is also a VBA code generator. For example, after recording a macro, the code will be VBA. You can then edit that code as original programming.

Creating a Macro

An important automation aspect of Word is its ability to create and run macros. Macros are the ability to record a sequence of commands, and then play them back in the same sequence. This has important time saving and quality implications in that you can record a detailed and complicated sequence and play them back with a single keystroke. To get started:

1. Open Microsoft Word.
2. Click **View | Macros | Record Macro**. The **Record Macro** dialog displays.
3. You may choose to rename it from *Macro1*.
4. Select **Document1** from **Store macro in** drop down.
5. Click **OK**. The mouse icon changes to include a cassette, indicating you're recording.
6. Perform your commands. In this example, we'll format a table.
   a. Type "Macro Example." Highlight it and style it as Heading 1.
   b. Place the cursor at the end of the line and enter Return.
   c. Select **Insert | Table | Insert Table**, sweeping a 2x2 table.
   d. Enter text in each of the four cells. You will need to tab to the next cell rather than using the mouse.
   e. Select both cells of the first row. You'll have to use the keyboard arrow keys to move the cursor.
   f. Select any of a different font, size, color, or style.
7. Click **View | Macros | Stop Recording**.
8. Delete everything in the document.
9. Click **View | Macros | View Macros**.
10. Double click the macro name. The macro runs and the table is placed exactly as you entered it.

The new automation took between 20 minutes to an hour, depending on the complexity of the document, with a turnaround time reduced to less than four hours overall. Not all the changes were through software. One important process modification was to move this entire procedure from the quality review team to the book managers, as they could fix term anomalies quickly. This level of accuracy and speed is not possible manually. It had to be an automated tool. Neither could have been written as a series of macros. For instance, because of the sheer number of internal data iterations and algorithms, it had to be optimized for speed. It also required accessing Word's internal structures.

The Importance of Being Embedded

The traditional programming approach keeps the programmer separate from the client. However, in programming for writers, it is critical to embed programmers with the team. Merely sitting with them has advantages, such as overhearing conversations, learning jargon, observing tasks, and noting how things are being done. For example, in one instance I noticed a team member repeatedly hitting the same key for more than a minute. She was transferring files and her application didn't accept long names, so she had to skip those. Furthermore, there were at least 5,000 files in the archive, so that task would have taken an excessive amount of time, including manually changing the names to fix them. By intervening, we created an ad hoc application in less than thirty minutes that checked all the names directly in the Windows depository and corrected the long ones. A day-long task took less than two hours (with testing) and could be used by other team members. Even though she was an experienced team member who had worked with me before, she didn't realize that the task could be automated at the Windows level.

An extreme example happened later when I was transferred to become part of a team—a meeting support team that organizes routine formal management and engineering meetings, provides transcriptions and voice recordings, and manages agendas and invitation lists. Due to their transparency, they had been overlooked for technology and process improvements. The mission was essentially process improvement, changing procedures as needed, and writing efficiency tools. "Nothing is off the table," said my supervisor. This was truly an exciting and evolutionary step. In the nine-month assignment, the first three months were to learn to do the job. After that, changes could be made. And the insight that period provided was revolutionary. In all, we added 14 tools. They ranged from converting email forms to meeting logs, automatically adding and modifying agenda items, and queuing long lists of presentations (from PowerPoint, Word, and Excel) into a single PDF and print job. Procedures were changed, too, such as pooling team members to help with other meetings based on their availability, to something as simple as combining Windows locations into a single directory.

Failed Projects: Table Number 3

Not all projects, even those considered good automation ones, succeed. Table Number 3 in one of our documents was a notable failure. It was a standard table, in this case, vehicle launch, docking, and return information for all the flights (manned and supply missions) to the space station during a given period. The problems started when we discovered the business logic (that is, the rules applied to formatting and notes presentation) changed after each release. This was partly due, ironically, to our success, because managers wanted more business logic. But it was largely because the rules changed outright, as flight engineers redefined them. In addition to writing new rules, the existing business logic became almost incomprehensively complex, including exceptions with exceptions to the exceptions. Due to the time and effort required to maintain it, management decided to revert the automation to manual formatting. The code was rolled back to an earlier phase of creating and formatting the table for its initial use.

Summary

Automation is an extremely powerful tool and can be applied in innumerable ways. For one writing team, the document turnaround time dropped from the contractually required twenty days to just three. In addition to saving time and money, these tools reduce stress. Tools development doesn't have to be complex. Word encourages creating macros just by recording your steps. However, adding a programmer can unlock additional features, such all of .NET, regular expressions, and internal Word capabilities. It is unlikely that you would be assigned your own programmer, though, so consider approaching the IT department with a specific request for a single feature. As a compromise, suggest building complex tasks progressively. A small investment from a programmer could save the writing team literally weeks each year.

ROBERT DELWOOD *is a programmer, writer, and programmer-writer formerly with NASA's Johnson Space Center and Microsoft. With more than 18 years' experience, he has written and documented topics from Windows kernel-level device drivers and speech recognition APIs/SDKs for Microsoft, to help desk procedures and application manuals for the military. He specializes in Microsoft Office automation with VB/VBA and .NET VSTO. He's authored several books, the most recent one, a college-level textbook,* The Secret Life of Word *(*http://xmlpress.net/publications/word-secrets*/) about Word's automation for technical writers, non-programmers, knowledge workers, or anyone who wants to do more tasks quickly with Word.*