Technology

Property

Technology—Property Editor: Gerald J. Hoenig, 8495 Caney Creek Landing, Alpharetta, GA 30005, ghoenig@mindspring.com.
Guest Editor: Gary Whittington, AEGON USA Realty Advisors, Inc., 4333 Edgewood Road N.E., Cedar Rapids, IA 53499-5221, gwhittin@Aegonusa.com.

Technology—Property provides information on current technology and microcomputer software of interest in the real property area. The editors of *Probate & Property* welcome information and suggestions from readers.

Wasted Days and Wasted Nights: Confessions of a Homeschooled Template Developer

When I started automating templates for a commercial mortgage lender using the LexisNexis HotDocs program, I had little idea that I would end up speaking a foreign language of computation operators, variable names, and the like.

Back in the Late Pleistocene era of HotDocs 1.0, our loan origination operation was relatively small, with production volume in the low hundreds of millions. The lender used local law firms as closing counsel, and each firm maintained its own forms. With HotDocs 1.0, my goal was simply to automate a commercial loan commitment letter, so that turnaround time for legal input and review could be reduced. I accomplished this by creating a template to create fairly complete loan commitment letters based on the user's loan information. These loan commitments rarely omitted significant terms because the template acted as a complete checklist to elicit all the deal-specific information from the user. Outside counsel would use these commitment letters to draft the loan documents tailored to their jurisdictions. The detailed information incorporated in the automated letter cut down on attorney negotiations and resulted in more consistent loan documents.

Having succeeded in creating a successful commitment letter template, I succumbed to HotDocs subtle appeal to my latent megalomania. From my single template with a user-friendly interface for incorporating a few merged variables and rules, I started to envision more: a complete loan documentation system driven from a single set of templates incorporating all deal data and local law requirements. The documents would be free of nasty "one size fits all" boilerplate. They would assemble themselves to fit a particular loan using a single set of facts concerning the particulars of a loan.

And so, lacking favorite TV shows and having both a screened porch and a laptop, I started slipping out onto the porch on summer nights to play around with the document templates. Ten years and about a billion mouse clicks later, I find myself in possession of a fair amount of experience in developing templates that are on the complex end of the HotDocs template spectrum.

When rules of universal application are identified correctly, such a system can indeed permit those who do not know all of the rules to produce document drafts, in which much institutional knowledge is imbedded. A complex template system like my loan document package can serve as a single repository for the intellectual contributions of loan officers, lawyers, and other real estate professionals. It can be a focal point for documentation expertise and knowledge. There are, however, many ways to go astray. In the hope that some of the borrowers and others, whom I have annoyed over the years by rejecting their proposed changes, will accept this column as a gesture of goodwill, I will describe in this article the most imprudent and wasteful mistakes I made in developing the loan document templates. Those who use HotDocs may save some time. Those who do not may vicariously enjoy my suffering.

Blunder Number One: In the Swamp of Perfectionism

A template developer can easily identify tasks that are important to the production of letter-perfect documents. The hard part is deciding which tasks are worth the automation effort. A good example is the automation of notarial acknowledgements. Because recorded documents must contain an appropriate form of acknowledgement acceptable to a county recorder, how convenient it would be if all documents in all states included an automated acknowledgement! Perhaps some have accomplished this, but I

found myself devoting inordinate resources to a task easily completed by local counsel slapping on a form of acknowledgement appropriate to the entity, parties, and jurisdiction. Although the automation task seems simple on its face, the formal requirements of recorders, the multiplicity of entity types, and the various capacities of signatories led me to develop an inordinately complex sub-template that I eventually abandoned because of the need to devote attention to more pressing priorities. (A sub-template is a HotDocs feature that permits a form to be developed for insertion into master templates.)

Automating signature blocks is a similarly time-consuming task. There are many possibilities to address because of the variations inherent in the multiple capacities of intervening entities between the borrower and the signatory. My automation produced frequently flawed results and soaked up too much of my time. It usually takes less time to manually cut and paste the correct signature block into the loan documents than it takes to input the necessary information for HotDocs to create the signature block.

Blunder Number Two: Drafting by Programmers

An initial question is who will program the automated template. Either a subject matter expert can learn HotDocs programming or a programmer knowledgeable in HotDocs can automate documents based on markups provided by the subject matter expert.

Initially, our information technology experts proposed to do all HotDocs programming. It quickly became apparent that this approach would be impractical, at least for us. The first reason was our organizational circumstances. We were growing by acquisition at a rapid pace in those days, and it was obvious to me that programming resources (and thus, the project) would always be at risk. The next time an acquisition occurred, all programmers would be assigned to integrate our information technology systems with those of the acquired company. A second reason was the delay and inefficiency resulting from the programmers' heavy workloads and the many meetings and communications with them. It became apparent that it was most practical for me to do the automation programming. Even though this resulted in many mistakes a trained programmer would have avoided, the automation actually happened, the templates worked, and many programming mistakes were corrected later with the help of consultants. A timely return on the investment of my time was key to maintaining both management and user support for my further document automation efforts. A fellow user is in the best position to know how to make other users' lives easier through automation.

It seems to me that it is far easier for a practitioner with substantive expertise to learn the automation process and how to troubleshoot programming problems than for a programmer to obtain the substantive information necessary to create the templates. Consultants can help bridge the gap for some, but for us, direct automation by subject matter experts intimately familiar with the desired output has proven more successful. A key to replicating my approach in a private law firm would be management's willingness to compensate the expert's time spent on automation in lieu of billable hours.

Blunder Number Three: The Grand Plan

The vision of a grand plan easily overcomes common sense, resulting in an effort to do too much too fast. I fell prey to the lure of the grand plan in a variety of ways. One way has been to shear off the project into a development ("Beta") version that becomes too cumbersome to implement in the context of an organization that must produce documents continuously. At times I cloned our template set into a development version and attempted a series of changes that led me into more and more difficult and complex development. The result has been delay in implementing relatively simple and needed changes. An alternative approach that seems preferable in hindsight is to work more incrementally. Not only does this approach prevent publication delays, but it also simplifies the problems encountered in testing. Identifying the most pressing problems by taking a hard look at document output is also much better than allowing pet peeves of little practical significance to consume too much development time.

Blunder Number Four: Taking Too Much Drafting Responsibility Away from Users

I learned the hard way that users need tools to solve the inevitable problems in the documents resulting from the automated template. Prairie dogs always have an escape route and so should template users. This can be accomplished by giving users access to an organized library of document provisions that can be accessed from the same library they use to assemble templates. This approach makes possible an improved first draft for a user, who may well know more than the developer about what he or she wants the document to say. It also provides a source of alternative provisions for use as transactions change through negotiation or because of flaws in the initial data set used to assemble the documents.

Blunder Number Five: Encouraging a Slave Mentality

The logic underlying complex templates can be opaque to the average user, resulting in the false assumption that the resulting documents are error free and stifling the user's critical thought process and creative contributions. Initially, I contributed to this problem by introducing too many hard computations that prevented users from adapting their answers to the transaction. They had the frustrating experience of correcting wrong answers, only to find that their efforts were futile as hidden computations overrode their mouse clicks. Improvements in HotDocs now permit better control of the interface through the use of dialog variables. These variables give the developer the ability to change from mandatory to

proposed answers from the user and to allow the user to preview the results of his or her answers. For example, if a user indicates the borrower is organized as a special purpose entity, then they would see that covenants to operate the borrower in this fashion will be included automatically, unless the variable introducing such covenants is overridden by the user. My advice is to use the HotDocs default operator to introduce the developer's assumed answer and to provide explanatory "resource panes" for variables, so that the answers can be changed by the user when appropriate. This method of populating variables with computed default answers provides transparency to the users, informing them of the thinking behind assembly options, and invites them to contribute their frequently superior understanding of the terminology needed.

Blunder Number Six: Forgetting the 80/20 Rule

Early in the development process, while working on a particular deal, I identified improvements and attempted to implement them for that deal, rather than finishing the document manually and turning to the automation task afterward. The result was my imposing on the users the task of reassembling documents over and over as I attempted to perfect the template to solve the problem. Over time, I learned that it was best not to hold a transaction hostage like this. It is preferable to make changes to documents manually, complete the job at hand, and then turn to the solution of the problem that the particular assembly brought to light. This decompresses the process of fixing the templates and fosters goodwill with users.

Blunder Number Seven: Too Far Ahead of the Technology Curve with Web-based Assembly of Complex Templates

The HotDocs desktop product has produced excellent results for my company. The use of a master component file has proven indispensable. Each HotDocs template has its own component file: a set of text, true/false, multiple choice, dialog, and other variables, which are essentially either field or chunks of computer code. If multiple templates relate to a single transaction such as a loan, all of the templates can be made to point to a single "master" component file, which functions as a kind of glossary of fields and options for use as a hyper-language to write forms for the transaction. This valuable tool permits the developer to write a single set of variables that can be used in many related templates. The developer need not "skinny down" the component file of a particular template to make it run efficiently; the template assembles quickly in the desktop product even though the component file includes numerous unused variables.

Initial versions of server-driven, on-line products have produced slow assembly and disappointing results for such multi-template transactions. Although we have successfully developed a front-end for an SQL database, its utility has been limited to pushing and pulling data into and out of the database. The utility of such an application may be considerable for some, but our users have preferred to stick to the desktop product for ordinary document assembly. In hindsight, it would have been advisable to wait for the technology to mature before attempting to run HotDocs on a web server. I remain convinced that the day of web server document assembly has not yet come.

Blunder Number Eight: Too Few Documents

Perhaps the most common mistake is to automate a template that will be used only rarely. This is a judgment call on prioritization. I continue to learn by trial and error that it is easy to spend too much time on an incidental template and too little on templates that are used every day. The cure for this is to remain a user oneself. By keeping in the mainstream of document production, your document automation can be informed by actual deals and document problems rather than guesswork or secondhand information.

Conclusion

On balance, we have overcome most of my mistakes and are successfully using HotDocs as the primary means of mass producing reasonably high-quality first drafts of commercial mortgage loan documentation. In the process, I have learned that there is quite a bit more to this process than the mastery of the software. Organizational, human resource, and interpersonal challenges abound. It is hoped this confessional will inform fellow developers of a few of the most easily avoidable mistakes. ■