# AUTOMATIC WORD GLOSSARY GENERATION

By: Norman E. Smith, CDP
Science Applications International Corp.
800 Oak Ridge Turnpike
Oak Ridge, Tennessee 37831

I seem to be the only Forth programmer that thinks Forth is a good general purpose language. The computer shop I work in primarily uses Fortran-77 and a little C. Needless to say I have had a hard time getting Forth accepted. I did develop one small communications program on an IBM-PC in Forth. That experience taught me several things about trying to use Forth in a non-Forth environment. Management must be convinced that the system will be maintainable. The best way to do that is to properly document the system.

Another problem management seems to have with Forth is its reputation for being a 'write-only' language. Any language can be write-only, it's just easier to develop write-only code in Forth. I have seen plenty of write-only Fortran code, but some of the worst has been C. Let's take a look at a code segment in both Forth and C and see if Forth is really the write-only language.

Forth-
```
    : DO-SOMETHING OVER + 1 + SWAP DO I C@ 32 = IF I LEAVE
      THEN LOOP;
```
C-
```
    do-something(x,y)
    int *x,y;
    { int i,z;
    for(i=0;(((z=*x+i)!=32)&&(i<y));i++) return(i+*x);}
```
Both versions of 'do-something' perform the same function, and both are write-only code. Worse code can easily be written in either language! Both code segments are passed the starting address and maximum length of a null terminated string and return the address of the end of the string.

How do you take Forth out of the write-only realm? Proper documentation is the only cure I know of! Program documentation is the programmer's responsibility, more so with Forth than with other languages! I have seen a lot written about Forth coding style. As far as I am concerned, Forth is a low level language. Forth code should be commented as if it were assembler. That is almost every line should have a comment. This means that only one Forth statement should be written per line! Don't forget proper indentation. Finally, include the text that would go in the word glossary in the screen with the word. It always seems to take longer to write glossary entries after the fact. Besides, if you can't write a couple of sentences describing a word, then you don't know enough to code it!

The suggestions mentioned above that contribute toward good documentation seem to take less time during coding. They also help insure that you think the application through. Putting together a Program Maintenance Manual will be a lot easier if large portions are already written as a by-product of coding.

After completing the communications system, I found that generating the word glossary by hand took almost as long as writing the code. I had already written a Forth program to list the first line of each word definition along with its screen number. This word proved an invaluable tool during both development and testing. It turned out that I already had most of the other pieces to automatically generate word glossaries already written. A little additional code and the next glossary would be easy to generate. My automatic word glossary generator is really semi-automatic, but who's complaining.

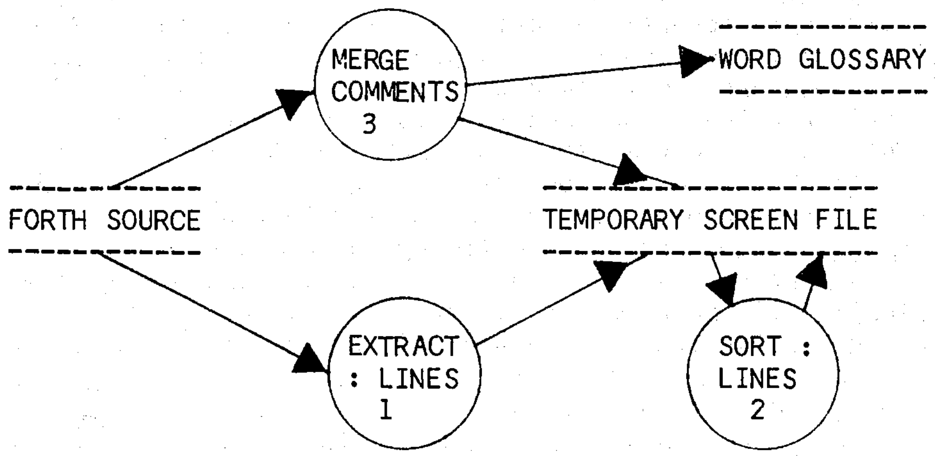
All of this talk of program documentation leads up to an application program that will make generating good Forth program documentation easier. My automatic word glossary generator does this. The word glossary is one of the most important sections of a Program Maintenance Manual for any Forth system. If you follow my coding style suggestions, you will have done about 80% of the work necessary to generate the word glossary.

Before I discuss the automatic word glossary generator, I want to mention another common documentation mechanism that is widely used in the Forth community, shadow screens. I do not care for them. They do have their place. Shadow screens can be a big help for after the fact documentation, but I find them almost useless for documentation as you code. There are two forms of shadow screens that I know of. The first and most popular is to split the screen file in half, with the first half being code and the last half being the shadows for documentation. The second shadow screen method is to use every other screen as the shadow. I like the second method better because the task of keeping the code and its shadow in synch is easier if you must insert additional screens in the middle of a section of code. This assumes you actually keep the shadows up as you code. Several of the public domain Forths make extensive use of shadow screens as their primary form of documentation. None has what acceptable shadow screen support. It may be easy to switch between the code and its shadow, but they make no provision for displaying both at the same time. The automatic word glossary generator method gets around the problems that are inherent with shadow screens and lets me see the code as I write its word glossary entry.

The automatic word glossary generator exists today primarily because I used portions of other code in a program toolbox approach to build large portions of the final program. Otherwise, I probably would have never gotten around to actually writing the program. The toolbox approach to software development has long been used in the Unix/C world with much success. Personal experience leads me to believe heavily in the validity of the software toolbox. Forth happens to be a very good environment for developing toolboxes. I have built up a collection of words I carry from Forth to Forth and program to program over the years.

The primary reasons that Forth is so good for constructing program toolboxes is that programs are constructed as relatively simple building blocks, and the interface between words is easy to define and test. Once a word is tested, it can be depended upon in other words.

The design of the automatic word glossary generator is relatively simple. It consists of three phases and has one input file, one work file, and one output file. Figure 1 is the data flow diagram for the automatic word glossary generator.



AUTOMATIC WORD GLOSSARY GENERATOR DATA FLOW DIAGRAM
FIGURE 1

The automatic word glossary is divided into three phases and uses three files as shown in Figure 1. The input Forth screen file is passed to extract the word definitions. This first phase, EXTRACT : LINES, examines each line looking for a ':' in column 1. The entire line, along with the screen number is written to the temporary screen file each time a ':' is found. One simplifying assumption was made in the word definition search; only column 1 is checked for the ':'. This is the accepted convention; besides I was writing the program for myself and I always follow this convention.

The second phase, SORT : LINES, sorts the records in the temporary screen file. A simple bubble sort is used. Sorting speed is acceptable because the records are fixed length of 64 characters. So, there are 16 records to each physical screen. This large number of logical records helps cut down I/O time a great deal. The first fifteen characters are used as the sort key. The length of the sort key is a parameter and can easily be changed. I have found 15 characters adequate. The simplicity of a bubble sort and direct access nature of screen files allow the file to be sorted completely within itself.

The final phase, MERGE COMMENTS, reads the temporary screen file, and uses the screen number stored in the record as an index into the original screen source file. The ':' line from the temporary file is written to the word glossary file. The source screen is searched for a ';S' starting in column 1. If a ';S' is found, the rest of the screen is copied to the word glossary file. This procedure continues until all records in the temporary file have been processed.

The final output is the beginnings of a word glossary with most of the hard stuff done. All of the words are in sort order. The words you took time to describe during coding have their glossary entry already done. The text file will require at least some editing even if you documented every word, because no attempt is made to separate descriptions in the case that there are multiple words and descriptions per screen. This means that lines that do not apply will have to be deleted, but it is a lot easier to delete a few lines than to have to write them.

I took the program toolbox approach to the overall development of the automatic word glossary generator. Much of the code for the final program already existed, either stand alone or as part of some larger program. Building program was a matter of merging all of the parts, deleting unused code, and adding some new code to tie all the pieces together. Code already existed for the following: 1) listing ':' lines within a group of screens, 2) sorting records in a screen file, 3) logical files within a screen file, and 4) an abbreviated string package.

Several support words were necessary to build the source program. First, I needed to convert several screen files into normal ASCII text files so they could be manipulated by operating systems utilities. I wrote UNLOAD to perform this function. It turns out that the space compression because of the eliminated whitespace was so significant that I now keep all of my source code UNLOADed when I am not working on it. After the existing source code was all ULNOADed, I appended the files together and started modifying the source with my favorite text editor, MicroEMACS. There was no built in utility to load text files into screen files so I wrote FLOAD. The method I followed of UNLOADing, editing the text file, and FLOADing the source made the whole process of combining portions of multiple screen files much easier than if I had tried to work completely within Forth screen files. It is difficult to do a lot of cut and pasting with most Forth editors, but is very simple with MicroEMACS.

There are a couple of enhancements that might be desirable and could be made with very little trouble. The list of ':' lines could be written to a text file both before and after being sorted, and formatting commands for your favorite text formatter would be useful. Was the effort worth it? Yes! After using the automatic word glossary generator, I was amazed at how easy it is to produce quality Forth program documentation. I wonder why the major Forth vendors don't include documentation aids of this type. Documentation can go a long way toward helping management accept Forth as a viable general purpose programming language.