Time-Saving Debugger
Redefining Words

Phil Koopman, Jr.
FPO San Francisco, California

One of the biggest time wasters in writing large Forth programs is the compiling delay that occurs whenever a word defined near the beginning of the dictionary must be changed. The recompilation of the dictionary after a redefinition can often take several minutes for a large application. Numerous methods have been tried to reduce this delay, typically addressing methods to speed up the Forth compiler. Most methods involve large numbers of extra word definitions and significant changes to the dictionary structure of Forth.

This article discusses a simple method to eliminate the time-consuming recompile step after making a minor change. Only one screen of source code is used, and absolutely no changes to the Forth compiler or dictionary structure are required.

The Method

When a small bug is corrected (usually involving the definition of only one word) all that is needed to make the entire program correct is to compile the revised definition and somehow ensure that all references to the old definition are changed to reference the revised definition.

The first way that comes to mind to compile the new word is just to compile it to the end of the dictionary. This will mean that any new word defined will use the revised definition, but no previously defined words will do so. This method fails when the word being revised is used by any previously compiled word.

Another method might be to compile the revised definition directly into the memory used by the old definition in the dictionary. This eliminates all need to change words that use the revised word, since the location of the definition does not change. This method works fine unless the revised definition is larger than the original definition, and therefore will not fit into the dictionary in the old definition's spot.

The solution presented in screen 180 is a combination of the two above methods. The technique used is to define the revised word at the end of the dictionary, then modify the old definition so that it merely executes a jump to the revised definition.

How It Works

The redefinition process is in three steps: using **REDEFINE** to alert the system that a word is to be redefined, redefining the word and then using the word **PATCH** to actually put the new definition into effect.

REDEFINE alerts the system that a word is about to be redefined. It is used in the format

REDEFINE <name>
where <name> is the word name to be redefined. It saves the PFA of the old definition of <name> in the variable **PATCHADDR**.

The second step of the redefinition process is to define the revised definition of <name>. This is usually by means of a **LOAD**, but any means may be used. Note that no special words within the definition are needed, and no editing of screens is required.

The third step of the redefinition process is to use the word **PATCH** to actually patch the old definition to point to the revised definition. **PATCH** is used in the format

PATCH <name>

where <name> is the name of the revised definition. The name used with **PATCH** is usually the same as the name used with **REDEFINE**, but does not have to be. **PATCH** uses the factored word **MAKE-PATCH** to compile the run-time action word **<PATCH>** into the first cell of the old definition's parameter field. The PFA of the revised definition is stored in the second cell of the old definition's parameter field.

At run time, the word **<PATCH>** is executed by the old definition. **<PATCH>** removes the pointer to the old definition from the return stack and replaces it with a pointer to the parameter field of the revised definition. This ensures that any remaining words in the parameter field of the old definition are ignored, and that the return stack is not cluttered up with another return address (in case the revised word uses an unusual exiting technique or is otherwise expecting certain values on the return stack).

Screen 181 shows a very simple test that illustrates the ease of use of this redefinition facility. First, the words **ATEST**, **BTEST** and **TEST** are defined and used. Then the word **ATEST** is redefined and incorporated into the dictionary without having to redefine **BTEST** and **TEST**. Note that the return stack contents are identical for both the old and new versions of **ATEST**.

Limitations

First, this facility is not designed to be very "smart." It will be perfectly happy to crash if the exact sequence of **REDEFINE** ... **LOAD** ... **PATCH** is not properly used. Also, it only works for high-level definitions and will not work for code words, constants, variables or the like.

Another limitation is that the word redefined must have at least two cells in its parameter field. This means that a null definition cannot be redefined with this system (a null word is in the form : **NULL** ;). Only one word may be redefined in any **REDEFINE** ... **LOAD** ... **PATCH** sequence.

FORGETting the redefined word without also **FORGET**ting the original definition can cause the system to crash.

<PATCH> may have to be redefined on systems that pre-increment the IP instead of post-incrementing it. The change is simply:

```
:  <PATCH>
R>   2+
@   >R  ;
```
for a sixteen-bit cell size.

On the Bright Side

The words **REDEFINE** and **PATCH** provide a very quick way to change a word definition and examine its results without recompiling the whole dictionary. No change is made in the dictionary structure. When the application is fully debugged (or when a substantial number of bugs have been corrected), the application can be recompiled one time to clean it up and free the extra space used by word redefinitions. Additionally, multiple redefinitions of the same word can be written and quickly tested in the context of the entire system without recompiling.

Summary

The words **REDEFINE** and **PATCH** provide an extremely simple yet effective way to practically eliminate recompile time during debugging. This technique has provided substantial time savings while developing a hotel cash register/control system. The program was developed on ECS MVP/FORTH (a Forth–79 compatible system) running on a well-known 8088-based personal computer system.

Further investigations of redefining words might include greater safeguards against accidental misuse and simultaneous multiple-word redefinitions.

Editor's note: This technique is sometimes referred to as "hotpatching" and must be used very carefully, due to the problems that can arise when the source and object code versions of a program differ.

```
SCREEN #180
\ DEBUGGING PATCH WORDS         P. KOOPMAN JR.   28DEC84
DECIMAL               \ DEFINITIONS IN THE PUBLIC DOMAIN
VARIABLE PATCHADDR     \ PFA FOR REDEFINE, USED BY PATCH

: <PATCH>     ( -> )  ( ACCEPTS IN-LINE PFA FROM DICTIONARY )
  R> @    >R ;

: MAKE-PATCH    ( PFA-NEW   PFA-OLD   -> )
  ' <PATCH> CFA   OVER !      2+ !   ;

: REDEFINE    ( -> )    ( USAGE: REDEFINE name )
  [COMPILE] '   PATCHADDR !  ;

: PATCH     ( -> )       ( USAGE: PATCH name )
  [COMPILE] '    PATCHADDR @    MAKE-PATCH ;


SCREEN #181
\ TEST SCREEN FOR PATCHING        PJK    28DEC84    MVP FORTH
DECIMAL
: ATEST   ." TOP OF RETURN STACK="    R@ U.    CR ;
: BTEST   ." BTEST IS AN UNCHANGING WORD IN DEFINITION"  CR ;
: TEST    CR CR    ATEST    BTEST    CR ;

TEST

REDEFINE ATEST
: ATEST   ." THIS IS A REDEFINED ATEST. R@=" R@ U. CR ;
PATCH ATEST

TEST



```