SAND92-8225C ● UC-405
Unlimited Release
Printed January 1995

Supersedes SAND92-8225B, May 1993

The Change Tool for Changing Programs and Scripts

Version 2.04 of August 1993

(Submitted to *ACM Transactions on Mathematical Software*)

J. F. Grcar

Prepared by
Sandia National Laboratories
Albuquerque, New Mexico 87185 and Livermore, California 94551
for the United States Department of Energy
under Contract DE-ACO4-94AL85000

SAND92-8225C UC-405
Unlimited Release
Printed December 1993

Supersedes SAND92-8225B, May 1993

The Change Tool for Changing Programs and Scripts*

Version 2.04 of August 1993

Joseph F. Grcar

Sandia National Laboratories
Department 8745
Livermore, California 94551-0969 USA
(510) 294-2662
na.grcar@na-net.ornl.gov
sepp@california.sandia.gov

Abstract

Change is a computer program that interactively changes other programs. It allows one source text to move among many computers without proliferating versions for different compilers, precisions and the like. Change also modifies makefiles, shell scripts and programs embedded in makefiles and shell scripts. This is a guide to the program's use and a description of its operation.

* Submitted to *ACM Transactions on Mathematical Software.*

Acknowledgements

I thank R. J. Kee for promoting use of the change program. Thanks to F. M. Rupley for suggesting the extension to scripts and for finding version 1.0. Thanks also to C. L. Bisson, R. Ellis and D. Schneider for many discussions of tools, and to A. E. Lutz for reading the preliminary manuscript. Thanks to J. H. Bolstad for finding many errors, and to F. N. Fritsch and M. A. Saunders for suggesting some changes! Thanks finally to Prof. D. E. Knuth for one tool in particular, and to Blue Sky Research for Macintosh *Textures*.

Contents

1. The Need for Change

It is often necessary to change a piece of software frequently, but in a routine and predictable way. The software might offer different capabilities when debugging and testing, for example, or it might move between different computers when in general use.

Software migration alone forces many changes. Invariably, machines have different arithmetic precisions, file conventions and system services. The lack of standard interfaces between programming languages and operating systems guarantees portable software must change.

Software moves among computers three ways. *Selective distribution* moves software tailored to specific machines. Commercial software favors this method for its convenience and because it discourages redistribution. *Redundant distribution* moves software in multiple versions, en masse, with some version appropriate for each machine. *Modifiable distribution* moves software in one version that must be fit to every computer. This method

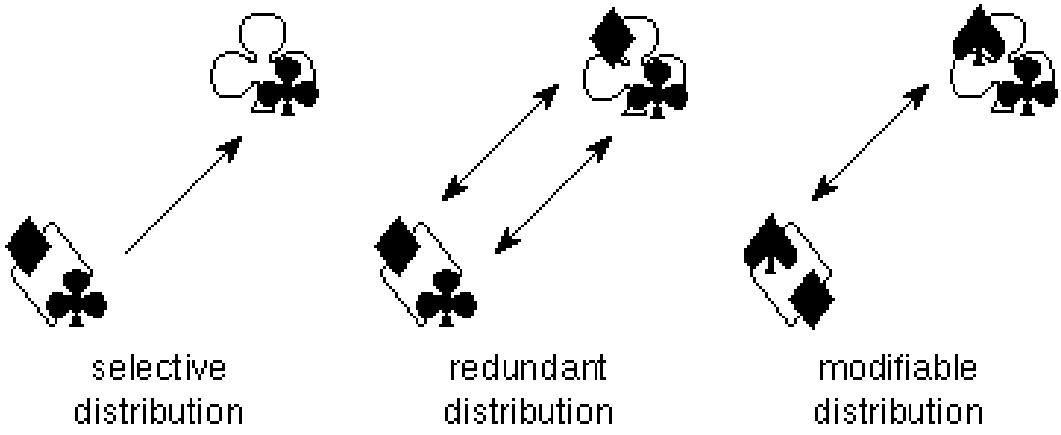

Figure 1. *Software moves among computers three ways (outlines represent computers, solid shapes represent software for computers of the same shape). Section 1 discusses this figure.*

For example, numerical software uses two distribution methods at present. Scientific computing requires sixty-four bit arithmetic, which is single precision on some machines and double precision on others. Thus, precision translators are needed to make numerical software portable. Translators exist on few machines, however, so numerical software must be distributed either selectively or redundantly. The lapack library [1] is distributed redundantly with each subroutine supplied in two precisions. The nag library [12], a commercial product, is distributed selectively.

That few machines have precision translators suggests existing programming tools do not make repetitive modifications routine. Some tools are not intended to make and undo simple changes easily. For example, these tools may impose inflexible managerial functions to coordinate large programming tasks. Many more tools are unreliable in the sense they are not available on all systems, or they are available but with arcane variations. In general, tools can be unwieldy even for experienced programmers, and they are unsuitable for occasional users unfamiliar with the tools or the software they modify.

Change is a simple programming tool that facilitates basic reprogramming. It manipulates programs in pre-approved ways anticipated by their authors. Change transforms

a given program to any of several versions. Each version contains information about all the others, so all versions can be recovered from any one. Change thus can make software portable, as in **Figure 1** (right). It can also modify software conveniently, as when exploring research alternatives. Change runs interactively on any computer independent of window environment (it doesn't use one) and flavor of operating system.

2. What Change Does

Change manipulates *change blocks*. These are small groups of fortran statements in program source files (or unix commands in makefiles and shell scripts).

```
C*****name of the change block
      |
      text of the change block
      |
C*****END  name of the change block
```

The fortran change block above is *active*; the block below is *inactive* because it has been "commented out."

```
C*****name of the change block
C      |
C      text of the inactive block
C      |
C*****END  name of the change block
```

Thus, a program might explain how to change its arithmetic precision by including the two change blocks below.

```
      PROGRAM MAIN
C*****DOUBLE PRECISION
C      IMPLICIT DOUBLE PRECISION (A - Z)
C*****END DOUBLE PRECISION
C*****SINGLE PRECISION
      IMPLICIT REAL (A - Z)
C*****END SINGLE PRECISION
      DOUBLE PRECISION DRAND
      REAL ETIME
```

Reversing the status of these change blocks reliably changes the program's precision. This can be done tediously using a text editor, or easily using the change tool. Change lists the blocks and asks which to change, as in **Figure 2**.

Note, few program transformations can be performed reliably without information from the programmer. In the case of changing arithmetic precision for example, rote textual substitution would confuse the precisions needed by external routines such as drand( ) and etime( ). Substituting `REAL` for "`DOUBLE PRECISION`" or conversely, gives these routines the wrong precision and invalidates the program fragment above.

Thus, well-written programs typically contain comments suggesting changes, and over time they acquire the commented-out remnants of modifications. This practice of embedding reprogramming information in programs themselves is as old as programming languages. Change blocks merely formalize a time-honored custom. Though the blocks are accessible

```
CHANGE:  VERSION 2.04 OF AUGUST 1993 BY JOSEPH GRCAR.

         ENTER THE NAME OF THE ORIGINAL FILE.

  FILE?  main.f

         154 LINES

         STATUS OF THE CHANGE BLOCKS.

        ORIGINAL  COPY   NAME OF THE CHANGE BLOCK
         ------  ------  -------------------------
     1)    --      --    DOUBLE PRECISION
     2)  ACTIVE  ACTIVE  SINGLE PRECISION

         ENTER THE NUMBERS OF BLOCKS TO CHANGE,
         OR ENTER BLANK TO ACCEPT THINGS AS THEY ARE.

NUMBER?
```

Figure 2. *The start of a typical interactive change session. See Section 2.*

to anyone with a text editor, the mild standardization suggested here allows the blocks to be manipulated easily, by the change tool, without laboriously inspecting the program text.

In sum, change is a computer program that activates and de-activates a file's prearranged change blocks. What happens when the file is used subsequently depends on the software in the file. The software's documentation might explain what the change blocks do, or the blocks' names might explain their use, as in "`SINGLE PRECISION`."

3. Some Other Tools

A programming tool is a program that aids programming. More narrowly, it either analyzes or translates programs, or it manages libraries. Some programming tools are surveyed here to place change in context.

At the highest level of abstraction is *metatool*, which builds translation tools [6] [7] [8]. From descriptions of translations, it produces tools (realized as c programs) that translate texts with reasonable grammars into other texts such as program fragments. For example, metatool might write a macro expander, and it does write itself. Machine-written tools are said to be more easily produced, efficient and reliable than those written by hand. With software generators easily generated, they might find greater use in scientific computing [22]. Metatool is a commercial product for unix system v compatible systems.

Lex and *yacc* are tools that build specific kinds of translators, namely token readers and grammar parsers, respectively (realized as c programs again). The translators so obtained can serve as "front-end" input readers for larger programs. If those larger programs are translation tools themselves, then metatool provides back-ends. Lex and yacc are available on all unix systems.

S2d and *d2s* are specific translation tools (c programs) that convert single precision fortran programs to double precision and back. They parse fortran statements and select from tables the correct precisions of external names such as those in the linpack library [20]. The tools are distributed under Free Software Foundation license by nanet's netlib and by the University of Texas [4] [11] [19] [21] [28] [31].

Toolpack consists of many analysis and translation tools for fortran programs. The analysis tools check for adherence to language standards, and they can find programming

errors invisible to compilers. Toolpack's translation tools change arithmetic precision, modify program appearance, and transform control structures. Toolpack was sponsored by the United Kingdom and the United States, and can be had for the cost of distribution from the Numerical Algorithms Group [2] [9] [10].

Nagware consolidates the toolpack tools. It simplifies their use while enhancing their capabilities and performance. Nagware is a commercial product available for several computer systems [10].

Note, analysis and translation tools make work for other tools. They must be revised occasionally, for example to recognize the syntax of new fortran or the precisions of new names, as when [1] replaces [20]. Metatool could generate the translators and revise them as needed. Moreover, the change tool can make the minor changes required to adapt the tools to various computers.

The *c macro preprocessor* is a general text substitution tool that can be used to change fortran programs too [35]. Originally part of c language compilers, it operates independently as *cpp*, and has become the macro utility of choice for unix systems. Many unix clones therefore make the preprocessor directly available to other languages. Some fortran compilers automatically preprocess `.F` files [29], while others preprocess everything. Thus, invoking the Silicon Graphics compiler [5] with "`f77 -Ddouble main.f`" compiles main.f after the following transformation.

```
      PROGRAM MAIN
#if (double)
      IMPLICIT DOUBLE PRECISION (A - Z)
#else
      IMPLICIT REAL (A - Z)
#endif
```

⇓

```
      PROGRAM MAIN
      IMPLICIT DOUBLE PRECISION (A - Z)
```

Use of cpp is system dependent. "If you use the preprocessor you are committing yourself to unix. You will have trouble porting your code to non-unix systems" [29, p. 63].

The *sccs source code control system* [13] and the similar *rcs revision control system* maintain libraries of programs and other text files. They store versions in condensed form organized by parent-child relationships. Texts are protected against loss and meddling, and are easily compared and reproduced. "Sccs is functionally identical to rcs but has a completely different user interface" [29, p. 167]. The control systems exist in many versions (ironically), with differences particularly regarding the *make* utility [33]. Some form of control system can be found on any unix system.

The *update* utility also maintains text libraries. While sccs and rcs maintain texts modified elsewhere, update makes modifications itself. It thus can function as a text editor, and did, prior to the introduction of interactive editors [15]. Update is available on Control Data and Cray Research [14] machines.

A comparison of these tools is inappropriate due to diversity of purpose, but clearly, simpler tools are more portable and easier to use. None of the tools above are simultaneously interactive, easily portable, readily available, and do quite what change does. Some tools are commercial products limited by cost to central computing centers. Many tools are in free distribution but exist mainly because they maintain unix operating systems. It is at least plausible these tools are not ideally suited to other programming. They may share syntax with the c language or assume c compilers, for example, though c is rare and sometimes

inappropriate in numerical computing [30]. In general, most tools either are not present on most computers (which are personal computers with limited resources and operator expertise), or they exist with variations that necessitate minor changes to programs and scripts. Thus, these tools do not satisfy, and may intensify, the need for basic reprogramming tools, such as change.

Table 1. *A comparison of tools. Section 3 discusses this table.*

tool	type	system	interactive	cost
change	translation	all	yes	0
cpp	translation	unix	no	0
lex/yacc	translation	unix	no	0
metatool	translation	unix	no	$
nagware	analysis/translation	some	no	$
s2d/d2s	translation	unix	no	0
sccs/rcs	library	unix	no	0
toolpack	analysis/translation	some	no	¢
update	library	some	no	$

(The remainder of this page is intentionally left blank.)

4. Using Change

The change program is self-explanatory. Usually, typing `change` or `CHANGE` runs the program, see Appendix 3. Change then prompts for information about the file to change and its change blocks.

Figure 3 displays a typical interactive session's three steps. In step 1, change asks for the file to change (LINES 3 TO 5), and it looks for the file's change blocks (LINE 7).

In step 2, change lists the blocks' names and shows their status, both in the original file and in the proposed copy (LINES 9 TO 16). In this list of names, parentheses after index numbers indicate change blocks in fortran programs (LINES 13 AND 14), while pound signs, #, indicate blocks in makefiles or shell scripts (LINES 15 AND 16). Change asks for some blocks to change (LINES 18 TO 21) and it displays the blocks' new status (LINES 23 TO 28). This dialogue may be repeated for any number of blocks and changes of mind (LINES 30 TO 45).

In step 3, change asks for a file to receive the changes (LINES 47 TO 52), and it writes the changed copy to this file (LINE 54). The original file and the changed copy always have the same number of lines.

5. Writing Change Blocks

Change recognizes change blocks in fortran programs and unix makefiles and shell scripts. The blocks have a definite syntax and must obey some rules.

Every block has a top line, a bottom line and some lines in between. The delimiting lines begin with a comment character (`C`, `c` or `*` for fortran, # for scripts) and five asterisks. The top line's remaining characters, from the seventh through the last non-blank, are the change block's name.

```
C*****OUTPUT TO main.out
      OPEN (FILE = 'main.out', UNIT = 6)
C     OTHERWISE, UNIT 6 IS STANDARD OUTPUT
C     AND IS REDIRECTABLE ON UNIX SYSTEMS.
C*****END OUTPUT TO main.out
```

The active, fortran change block above is named "`OUTPUT TO main.out`." Blanks and case are significant in the name, which may be blank. The bottom line matches the top line exactly, except `END` (or `End`, or `end`) and one blank lie between the five asterisks and the name.

A change block is inactive if all its lines begin with a comment character. Change deactivates a block by "commenting out" the lines in-between, including those that are comments already. It does this by prefixing a comment character, which for fortran change blocks is the one used in the delimiting lines.

```
C*****OUTPUT TO main.out
C      OPEN (FILE = 'main.out', UNIT = 6)
CC     OTHERWISE, UNIT 6 IS STANDARD OUTPUT
CC     AND IS REDIRECTABLE ON UNIX SYSTEMS.
C*****END OUTPUT TO main.out
```

Change activates a block by un-commenting the lines, that is, by discarding the leading comment character.

```
CHANGE:  VERSION 2.04 OF AUGUST 1993 BY JOSEPH GRCAR.

         ENTER THE NAME OF THE ORIGINAL FILE.

  FILE?  main.sh

         1729 LINES

         STATUS OF THE CHANGE BLOCKS.

        ORIGINAL  COPY   NAME OF THE CHANGE BLOCK
         ------  ------  -------------------------
    1)     --      --    PRECISION > DOUBLE
    2)  ACTIVE  ACTIVE   PRECISION > SINGLE
    3#  ACTIVE  ACTIVE   system > cray
    4#     --      --    system > sun

         ENTER THE NUMBERS OF BLOCKS TO CHANGE,
         OR ENTER BLANK TO ACCEPT THINGS AS THEY ARE.

NUMBER?  1 2

        ORIGINAL  COPY   NAME OF THE CHANGE BLOCK
         ------  ------  -------------------------
    1)     --    ACTIVE   PRECISION > DOUBLE
    2)  ACTIVE     --     PRECISION > SINGLE
    3#  ACTIVE  ACTIVE   system > cray
    4#     --      --    system > sun

         ENTER THE NUMBERS OF BLOCKS TO CHANGE,
         OR ENTER BLANK TO ACCEPT THINGS AS THEY ARE.

NUMBER?  3 4

        ORIGINAL  COPY   NAME OF THE CHANGE BLOCK
         ------  ------  -------------------------
    1)     --    ACTIVE   PRECISION > DOUBLE
    2)  ACTIVE     --     PRECISION > SINGLE
    3#  ACTIVE     --     system > cray
    4#     --    ACTIVE   system > sun

         ENTER THE NUMBERS OF BLOCKS TO CHANGE,
         OR ENTER BLANK TO ACCEPT THINGS AS THEY ARE.

NUMBER?

         ENTER THE NAME FOR THE COPY,
         OR ENTER BLANK TO QUIT.

         main.sh (NAME OF THE ORIGINAL FILE)

  FILE?  main.sh

         1729 LINES

         FINISH.
```

Figure 3. *A typical interactive change session. See Section 4.*

Change imposes a few rules for change block usage. Blocks may share names, but if one is active, then all with the same name must be active. Moreover, change blocks may neither nest nor overlap, so the construction below is invalid.

invalid, as nesting is not allowed:

```
C*****OUTPUT TO main.out
      WRITE (6, 100) RADIUS
C*****DOUBLE
  100 FORMAT ('THE INPUT WAS ', D20.14)
C*****END DOUBLE
C*****SINGLE
C  100 FORMAT ('THE INPUT WAS ', E10.4)
C*****END SINGLE
C*****END OUTPUT TO main.out
```

A common use of change blocks is to change precision. Usually, a few blocks suffice because fortran is forgiving where precision is concerned. The language allows compilers to interpret the precisions of subexpressions and of generic, intrinsic functions as appropriate [3].

unnecessarily careful change of precision:

```
C*****DOUBLE
      AREA = 3.14D0 * RADIUS ** 2
C*****END DOUBLE
C*****SINGLE
      AREA = 3.14 * RADIUS ** 2
C*****END SINGLE
```

Beware fortran compilers will not convert constants passed to subroutines. The change blocks below are invalid because the basic linear algebra subroutine dscal [20] needs a double precision scale factor.

invalid, as `2.0` *must change precision too:*

```
C*****DOUBLE
      CALL DSCAL (N, 2.0, X, INCX)
C*****END DOUBLE
C*****SINGLE
      CALL SSCAL (N, 2.0, X, INCX)
C*****END SINGLE
```

Fortran change blocks are more flexible than unix shell script blocks because fortran statements can continue past comments. Thus when a single statement has several variants, the invariant text need not be repeated.

pieces of fortran statements can be changed:

```
C*****DOUBLE
      CALL DGEFA
C*****END DOUBLE
C*****SINGLE
      CALL SGEFA
C*****END SINGLE
     +   (A, LDA, N, IPVT, INFO)
```

Finally, change blocks frequently select alternatives, as between single and double precision, and then exactly one of several blocks should be active. In this case, blocks

should be given names that suggest commonality. For example, change blocks might be named "`PRECISION > SINGLE`" and "`PRECISION > DOUBLE`" rather than `SINGLE` and `DOUBLE` alone. Change alphabetizes names when it lists them, so appropriately named alternatives appear together.

blocks with similar uses should have similar names:

```
   ORIGINAL  COPY   NAME OF THE CHANGE BLOCK
    ------  ------  -------------------------
1)    --    ACTIVE  DOUBLE          bad choice of name
2)    --    ACTIVE  PRECISION > DOUBLE        good
3)  ACTIVE    --    PRECISION > SINGLE        good
4)  ACTIVE    --    SINGLE                     bad
```

6. Chemkin Change Blocks

The change tool supports the chemkin libraries [18] [25] [27], the twopnt boundary value problem solver [24], and several reacting flow simulators built on chemkin and twopnt, among them [16] [17] [23] [26] and [32]. The sun never sets on computers running this software, so change was developed to make the software easily field-adaptable to various machines.

Figure 4 lists some change block names in the fortran programs and shell scripts that comprise the chemkin libraries. Documentation for the libraries should be consulted for information about particular change blocks. The fortran change blocks concern changes of precision, idiosyncrasies of file manipulation, and methods of obtaining computing time. Change blocks in shell scripts additionally choose compiler options.

```
COMPUTING TIME > CRI (CRAY) UNICOS
COMPUTING TIME > DEC (VAX) VMS
COMPUTING TIME > IBM (RISC System/6000) AIX
COMPUTING TIME > SUN (SPARCstation) SunOS
COMPUTING TIME > generic unix etime
COMPUTING TIME > none
LIST MESSAGES > NO
LIST MESSAGES > YES
OPEN statements > unix
OPEN statements > vax/vms
PRECISION > DOUBLE
PRECISION > QUAD
PRECISION > SINGLE
compiler > cray
compiler > ibm risc 6000
compiler > sun, sgi
exponent range > +/-30
exponent range > +/-300
old Cray operating system's TREMAIN
output > 123 column
output > 80 column
```

Figure 4. *Some change block names in the chemkin libraries. See Section 6.*

Chemkin's shell scripts coordinate the many programs and files used by each reacting flow simulation. The scripts consolidate in one file everything an investigator might wish to alter. Thus the scripts encapsulate data for the chemical reactions, main programs for

running the simulations, makefiles for creating the executables, and instructions for storing the output, see **Figure 5**. These scripts must adapt to different fortran compilers and machine precisions. To that end, the scripts might employ conditional evaluation of shell scripts and makefiles, and conditional compilation of fortran programs (thus relying on three levels of system software to achieve portability). It has been found more convenient simply to change the scripts themselves, using the change tool [34].

```
# to execute: sh premix.sh name &                          1
#                                                          2
#///  makefile for premix and the chemkin libraries        3
#                                                          4
cat << EOF > makefile                                      5
#*****compiler > cray                                      6
COMPILE = cft77 -a static -emx                             7
LINK = segldr -f indef -o                                  8
#*****end compiler > cray                                  9
⋮
EOF                                                       10
#                                                         11
#///  main program                                        12
#                                                         13
cat << EOF > driver.f                                     14
      PROGRAM DRIVER                                      15
C*****precision > single                                  16
      IMPLICIT REAL (A-H, O-Z), INTEGER (I-N)             17
C*****end precision > single                              18
⋮
EOF                                                       19
#                                                         20
#///  reaction mechanism                                  21
#                                                         22
cat << EOF > mech                                         23
ELEMENTS                                                  24
H  O  AR                                                  25
⋮
EOF                                                       26
#                                                         27
#///  simulation data                                     28
#                                                         29
cat << EOF > premix.in                                    30
PRES  0.0329 (atmospheres)                                31
FLRT  4.63E-3 (g/cm**2-sec)                               32
⋮
EOF                                                       33
#                                                         34
#///  make the executables and run them                   35
#                                                         36
⋮
make premix.e                                             37
premix.e < premix.in > premix.out                         38
ENDSH                                                     39
```

Figure 5. *Excerpts from a shell script with an embedded makefile and fortran program, both containing change blocks* (LINES 6 TO 9, AND LINES 16 TO 18). *See Section 6.*

Appendix 1. Change in Detail

The change program modifies files in three steps. This appendix explains some details of what change does with files and lines at each step.

1. read the original file and search for change blocks
2. prompt for blocks to change
3. write the final copy

In step 1, change imposes an unforgiving limit of 200 characters per line. That is, if lines in the original file have more than 200 characters, then change won't notice, and will produce a changed copy without the excess characters. The change program can be reprogrammed to accept longer lines (edit the program to increase the parameter `MAXS` and to increase the declared lengths of some strings).

Change can be made to insist files have no more than 72 characters per line (see Appendix 3 for instructions on setting change's change blocks). In this case, if a file has nonblanks beyond position 72 (and through position 200), then change will notice and won't proceed. Seventy-two is the line length read by fortran standard compilers [3].

In step 2, change accepts any number of blocks' numbers on the same line. Blanks must separate the numbers. Change ignores not-numbers and out-of-range values without comment.

In step 3, change tries to protect the original file from damage.* It won't alter the file unless the final copy is given the same name, see **Figure 6**. As a precaution against loss of data therefore, change can be made to insist the original file and the final copy have different names (see Appendix 3 again for setting change's change blocks).

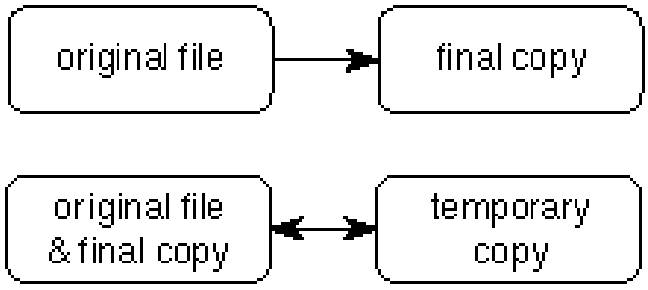

Figure 6. *Change uses two files even when the original and the final copy are the same. In the later case, the original is rewritten, not reopened or recreated. The temporary copy is a formatted, scratch file.*

Note, it is safer to allow change to rewrite original files. Catastrophes most likely occur through unforeseen errors made when copying or moving by hand, as in the following example. Some computers give files "attributes" for line termination, printer options and the like. These attributes can depend on the program that creates the file, so an original file (created by a text editor) and an entirely new final copy (created by change) might have different attributes. In this case, long after the original file has been discarded through normal use, it might be found the new copy has been damaged through misuse due to unexpected

* Software that's free comes with no guarantee.

attributes. Change may be able to rewrite old files without changing their attributes, and if so, then rewriting the original file would be the safer course.

In any case, change deletes trailing blanks from lines in the final copy. An entirely blank line has no characters at all.

Appendix 2. Error Messages

Change complains when things go wrong. It anticipates errors, so nothing is lost and the original file is still intact, see Appendix 1. Error messages concern both file problems and the syntax of change blocks. Complaints are most likely the first time a file is changed when typographical errors are fresh. After a file's change blocks have been established, complaints are rare and usually are of three kinds.

(1) The change blocks may be inconsistent, that is, among blocks with the same name, some may be active and some not. This may occur when several files have been joined whose change blocks have the same names but different settings. The separate files must be changed and reassembled, or the assembled file must be changed by hand, using a text editor.

(2) The lines in the file may be too long, that is, change may expect shorter lines. This usually occurs when mixing fortran programs with unix scripts. Standard fortran limits lines to 72 characters (so it is convenient to restrict change to 72 characters too), but unix does not limit characters between newlines. If change expects 72, but a file has been prepared with no limit in mind, then change won't read the file. See Appendix 3 to remove the 72 character limit.

(3) Some errors are due to file troubles. Most likely, these occur when working in foreign directories or with files too heavily "protected." Such errors are computer and vendor dependent, so the fortran language explains them by means of system-supplied status codes. Specifically, change reports values given to `IOSTAT` [3, Sec. 12.7]. The reference manuals for specific computer systems must be consulted for explanations of these error codes.

Below are all the error messages written by change. Some suggestions for correcting errors follow each message.

Messages

```
ERROR.  A CHANGE BLOCK HAS NO LINES BETWEEN THE TOP AND
        BOTTOM LINE.

4521:  C*****GEORGE WASHINGTON
4522:  C*****END GEORGE WASHINGTON
```

A change block must contain some lines between its first and last delimiters.

```
ERROR.  A CHANGE BLOCK NAME IS NOT RECOGNIZED.

4528:  C*****ABRAHAM LINCOLN
```

While re-reading the original file (or the temporary copy), change encounters a name it does not remember from the first reading. This error should not occur.

```
ERROR.  AFTER 8701 LINES SUCCESSFULLY READ,
        A FILE READ FAILS WITH I/O STATUS 43.
```

Change can't read a line from a file, which might be either the original file or the temporary copy. This error should not occur.

```
ERROR.  FILE CREATE FAILS WITH I/O STATUS 43.
```

Change can't open the new file intended to receive the final copy. The file name may be improper, or the system may not give permission to open files in the current directory.

```
ERROR.  IN THE ACTIVE CHANGE BLOCK WHICH BEGINS ON LINE 4521,
        LINE NUMBER 4528 IS LONGER THAN 71 CHARACTERS.

4521:  C*****GEORGE WASHINGTON
4528:       +  (/13X, ERROR.  IN THE ACTIVE CHANGE BLOCK WHICH...
```

Change inactivates a change block by prefixing a comment character to each interior line. When change limits lines to either 72 or 200 characters, active blocks must have no more than 71 or 199 characters per line. Either the too long line must be shortened or split, or change must be made to accept longer lines.

See Appendix 3 to choose between 72, 72/73, and 200 characters per line. For the 72/73 choice, inactive change blocks may have 73 characters, so active ones may have 72. The change program can be reprogrammed to accept even longer lines (increase the parameter MAXS and the declared lengths of some strings).

```
ERROR.  IN THE INACTIVE CHANGE BLOCK WHICH BEGINS ON LINE 4521,
        LINE NUMBER 4528 IS LONGER THAN 73 CHARACTERS.

4521:  C*****GEORGE WASHINGTON
4528:       +  (/13X, ERROR.  IN THE ACTIVE CHANGE BLOCK WHICH...
```

If this error occurs, then change has been configured to accept 72/73 character lines. That is, most lines are limited to 72 characters, but since change inactivates a change block by prefixing a comment character to each interior line, inactive blocks may have 73 characters per line. Either the long line must be shortened or split, or change must be made to accept longer lines. See Appendix 3 to choose 200 characters per line.

```
ERROR.  INQUIRE FAILS WITH I/O STATUS 43.
```

Change can't obtain information from the computer system about the original file or the file intended to receive the final copy (it should be clear from the context which file is meant). The file name may be improper. Otherwise, this error should not occur.

```
ERROR.  LINE NUMBER 8701 HAS OVER 72 CHARACTERS.
```

Change has been made to insist lines have no more than 72 characters, and it has found a longer line. The line in the original file must be shortened or split, or the change program must be changed to accept longer lines, see Appendix 3.

```
ERROR.  OPEN FAILS WITH I/O STATUS 43.
```

Change can't open the original file or another existing file intended to receive the final copy (it should be clear from the context which file is meant). The file may be protected against use or it may be inappropriate in some way, for example, it may not be a text file.

```
ERROR.  READ FAILS WITH I/O STATUS 43.
```

Change can't read interactive input from the terminal, that is, from the special fortran unit sometimes called standard input [3, Sec. 12.9.2]. This error should not occur.

```
ERROR.  SCRATCH OPEN FAILS WITH I/O STATUS 43.
```

Change can't open the scratch file for the temporary copy. This error should not occur.

```
ERROR.  THE CHANGE BLOCKS ARE INCONSISTENT.  AMONG
        BLOCKS WITH THE NAME BELOW, THE FIRST IS INACTIVE,
        BUT THE BLOCK BEGINNING ON THE LINE BELOW IS ACTIVE.

4521:  C*****GEORGE WASHINGTON
```

This error usually occurs when many files have been concatenated to one. Change blocks may share names, but if one is active, then all with the same name must be active.

```
ERROR.  THE COPY MAY NOT OVERWRITE THE ORIGINAL FILE.
```

Either a different file name must be chosen, or the change program must be changed to permit rewriting the original file. See the discussions of files in Appendix 1 and of change blocks in Appendix 3.

```
ERROR.  THE FILE DOES NOT EXIST.
```

There is no original file with the name given. The file name may be misspelled, or the file may be in a different directory.

```
ERROR.  THE FILE ENDS IN THE MIDDLE OF A CHANGE BLOCK.
```

Either the file ends prematurely, or a change block delimiter has been mistyped and not recognized.

```
ERROR.  THE FILE IS OPEN TO SOMEONE ELSE.
```

Some other program is using either the original file or another old file intended to receive the final copy (it should be clear from the context which file is meant). Change cannot use files simultaneously with another program.

```
ERROR.  THE LINE AT THE BOTTOM OF A CHANGE BLOCK
        DOES NOT SAY "END".

4521:  C*****GEORGE WASHINGTON
4528:  C*****ABRAHAM LINCOLN
```

In the example above, the bottom line that pairs with line 4521 is

```
C*****END GEORGE WASHINGTON
```

This line has been either mistyped or omitted. If mistyped, it could be either the second line shown in the message, or some earlier line not even recognized as a change block delimiter. (Since line 4528 appears to be the correctly formed top line of the next change block, the typo or omission probably occurs somewhere between the two lines shown.)

```
ERROR.  THE LINES AT THE TOP AND BOTTOM OF A CHANGE
        BLOCK HAVE DIFFERENT COMMENT MARKS.

4521:  #*****GEORGE WASHINGTON
4528:  C*****END GEORGE WASHINGTON
```

The first characters must match: `C`, `c` or `*` for fortran change blocks, # for makefile and shell script change blocks.

```
ERROR.  THE LINES AT THE TOP AND BOTTOM OF A CHANGE
        BLOCK HAVE DIFFERENT NAMES.

4521:  C*****GEORGE WASHINGTON
4528:  C*****END GEORGE WASHNIGTON
```

The names must match. This is usually a typographical error.

```
ERROR.  THERE ARE OVER 100 CHANGE BLOCK NAMES.
```

A file may contain any number of change blocks, but change assumes there are at most 100 distinct names. The change program can be reprogrammed to remove this restriction (increase the TMAX parameter).

Appendix 3. Software Notes

Change is distributed both separately and with the chemkin libraries (contact the author for copies). It is a fortran program which must be prepared before it can be used. At least, change must be compiled and loaded. At best, some provision might be made to use the program easily.

The change program and a few subroutines are distributed in a single file called CHANGE.FOR, change.f, CHANGE.204 (to indicate the version number) or some combination of these. An executable image should be made from this file, by compiling and loading, in the manner customary for the computer system at hand. **Table 2** lists commands that have been found appropriate for several computers. There results a file called CHANGE.EXE or change that can be run, in the directory where it resides, by typing the command "RUN CHANGE" or change, as the computer system prefers.

If the change program is used frequently, then it is most conveniently used regardless of directory by typing CHANGE or change as though it were a system command. To accomplish this on unix systems, it suffices to place the executable file in the bin subdirectory of the user's home directory (and if necessary, add that directory to the list of directories the shell searches for commands). On vax/vms systems, a symbol must be defined by the command

```
$ CHANGE :== RUN [directory.extensions...]CHANGE.EXE
```

in which $ is the system prompt and *directory.extension* is the directory that contains the executable file (presumably, the user's home directory plus any subdirectory extensions). This command should be added to the user's LOGIN.COM file. Some combination of these devices (moving the executable file to a special directory and defining special names) may be appropriate on other systems.

The change program itself has eight change blocks. The first two choose whether to use a common extension to the fortran language that improves the appearance of interactive sessions.

```
1) $ EDIT DESCRIPTOR > NO  (FORTRAN 77 STANDARD)
2) $ EDIT DESCRIPTOR > YES (RECOMMENDED)
```

Exactly one of these change blocks should be active.

The third change block chooses whether the final copy can overwrite the original file. This is recommended because it is often desired to replace the original, and files are most likely lost through errors when copying or moving by hand.

```
3) COPY MAY OVERWRITE ORIGINAL (RECOMMENDED)
```

Table 2. *Commands for compiling and running the change program on a variety of computers. Appendix 3 discusses this table.*

machine	system, and compiler	commands
apple macintosh IIfx	system 7.0.1, absoft fortran 3.1.2	*rename change.204 to change.f* f77 change.f –o change *double click on the change icon, or type* change *in the mpw window*
cray ymp 8/264	unicos 6.1.5a, fortran 5.0.2.12	mv change.204 change.f cf77 change.f –o change change
dec vax 8700	vms 5.2, fortran 5.3-50	rename change.204 change.for fortran change link change run change
ibm 486 pc clone	ms dos 5.0, lahey fortran f77l-em/32	rename change.204 change.for f77l3 change 386link change –stub runb change
ibm ps 2/80	pc dos 4.0, microway ndp fortran-386	rename change.204 change.f mf386 change.f –bind change
ibm risc 6000/530	aix 3.2, xl fortran 02.02.0100.0003	mv change.204 change.f xlf change.f –o change change
sgi 4d/380vgx	irix 4.0.1, fortran 3.4.1	mv change.204 change.f f77 change.f –o change change
sun sparc 1	sun os 4.1.1, fortran 1.4	mv change.204 change.f f77 change.f –o change change

Appendix 1 discusses this matter and also the next three change blocks.

The fourth through the sixth change blocks choose whether and how change enforces a limit of 72 characters per line.

```
4) LINE LENGTH > 72 CHARACTERS (FORTRAN 77 STANDARD)
5) LINE LENGTH > 72 OR 73
6) LINE LENGTH > AS LARGE AS POSSIBLE
```

Exactly one of these blocks should be active. The fourth block limits all lines to 72 characters, which effectively limits lines inside active change blocks to 71 (because lines inside blocks gain a character when made inactive). The fifth block allows inactive change block lines to have 73 characters, so active ones can have 72. The sixth block limits all lines to 200 characters, which limits lines inside active change blocks to 199.

The last two change blocks choose whether change pauses before finishing. This prevents some window-based systems from closing the program window too hastily.

```
7) PAUSE AT FINISH > NO  (RECOMMENDED)
8) PAUSE AT FINISH > YES (FOR SOME WINDOW ENVIRONMENTS)
```

To change the choices for any of these blocks, run the change program on itself (or edit its change blocks by hand), and rebuild the executable.

Appendix 4. Change History

This appendix chronicles the changes made to change by version number and date. Version 2 is the first with separate documentation (this manual).

`1.00` January 1986.

`1.01` January/February 1986.

`1.02` February 1986. Install change blocks for cray/ctss and vax/vms.

`1.03` March 1987. Echo the original file name before prompting for the final copy.

`1.04` August 1987. Alter the prompt for the name of the final copy.

`1.05` October 1990. (1) Make many changes to improve error checking and reporting. (2) Alter change blocks for use with unix systems.

`1.06` July 1991. (1) Replace the two-pass loop structure with straight-line code. (2) Add use of a temporary copy in a scratch file. (3) Allow the final copy to overwrite the original file.

`1.07` September 1991. (1) Correct integer typing of variable `IN`. (2) Remove the initial rewinding of the original file to accommodate sgi/unix.

`1.08` October 1991. Add the ability to change makefiles and shell scripts.

`1.09` October 1991. Allow 132 character lines.

`1.10` February 1992. Install a change block to remove `C` from otherwise blank lines.

`2.00` August 1992. (1) Revise change 1.09, allowing 200 character lines. (2) Reverse change 1.10. (3) Make many other changes conforming to the manual.

`2.01` August 1992. Correct declarations in subroutine squeez.

`2.02` August 1992. (1) Alter some error messages. (2) Remove the external statement from program change to accommodate microway ndp fortran.

`2.03` April 1993. (1) Remove assigned go-to's from some error messages. (2) Allow 72/73 character lines.

2.04 August 1993. (1) Remove an unused array from the program. (2) Correct sorting of block names whereby the block type can be lost. (3) Correct handling of unrecognized names to echo the name in the error message and to copy the line with the name.

2.05 March 1995. (1) Correct error 9106 about unavailable copy files. (2) Correct error 9208 about inconsistent change blocks. (3) Rewrite the variable declarations.

References

[1] E. Anderson and others, *LAPACK Users' Guide*, Society for Industrial and Applied Mathematics, Philadelphia, Pennsylvania, 1992.

[2] Anonymous, *Announcing Toolpack 1 Release 2.5*, Numerical Algorithms Group Inc., Downers Grove, Illinois, undated.

[3] Anonymous, *ANSI X3.9-1978 American National Standard Programming Language FORTRAN*, American National Standards Institute, New York, 1978.

[4] Anonymous, *GNU General Public License (Version 2)*, Free Software Foundation, Cambridge, Massachusetts, June 1991.

[5] Anonymous, *IRIS-4D Series Compiler Guide Version 1.0*, Silicon Graphics, Mountain View, California, 1987.

[6] Anonymous, *MetaTool Specification-Driven-Tool Builder Reference Manual (Release 1)*, AT&T, 1990.

[7] Anonymous, *MetaTool Specification-Driven-Tool Builder System Overview (Release 1)*, AT&T, 1990.

[8] Anonymous, *MetaTool Specification-Driven-Tool Builder User Manual (Release 1)*, AT&T, 1990.

[9] Anonymous, *NAG Toolpack/1 (Release 2) Distribution Service Contents Summary*, Numerical Algorithms Group Inc., Downers Grove, Illinois, undated.

[10] Anonymous, *NAGWare f77 Tools*, The Numerical Algorithms Group Ltd., Oxford, United Kingdom, 1991.

[11] Anonymous, *Netlib Index*. This netlib guide can be obtained by mailing the message "`send index`" to either `netlib@ornl.gov` or `netlib@research.att.com`.

[12] Anonymous, *The NAG Fortran Library Introductory Guide, Mark 15*, The Numerical Algorithms Group Ltd., Oxford, United Kingdom, 1991.

[13] Anonymous, *UNIX in a Nutshell Berkeley Edition*, O'Reilly & Associates, Sebastopol, California, 1990.

[14] Anonymous, *UPDATE Reference Manual*, Cray Research, Mendota Heights, Minnesota, 1990.

[15] C. L. Bisson, *private communication*, Sandia National Laboratories, Livermore, California.

[16] M. E. Coltrin and R. J. Kee, *CRESLAF (Version 4.0): A Fortran Program for Modeling Laminar, Chemically Reacting, Boundary-Layer Flow in Cylindrical or Planar Channels*, Sandia National Laboratories Report SAND93-0478, Livermore, California, April 1993.

[17] M. E. Coltrin, R. J. Kee, G. H. Evans, E. Meeks, F. M. Rupley and J. F. Grcar, *SPIN (Version 3.83): A Fortran Program for Modeling One-Dimensional Rotating-Disk / Stagnation-Flow*

Chemical Vapor Deposition Reactors, Sandia National Laboratories Report SAND91-8003, Livermore, California, August 1991.

[18] M. E. Coltrin, R. J. Kee, and F. M. Rupley, *SURFACE CHEMKIN (Version 4.00): A Fortran Package for Analyzing Heterogeneous Chemical Kinetics at a Solid-Surface–Gas-Phase Interface*, Sandia National Laboratories Report SAND90-8003B, Livermore, California, July 1991.

[19] J. Dongarra, *NA-NET Help File*, Oak Ridge National Laboratory, Oak Ridge, Tennessee, undated. This nanet guide can be obtained by mailing the message "`help`" to `na.help@na-net.ornl.gov`.

[20] J. Dongarra, J. R. Bunch, C. B. Moler and G. W. Stewart, *LINPACK Users' Guide*, Society for Industrial and Applied Mathematics, Philadelphia, Pennsylvania, 1978.

[21] J. Dongarra and B. Rosener, *NA-NET Numerical Analysis Net*, Oak Ridge National Laboratory Report ORNL/TM-11986, Oak Ridge, Tennessee, December, 1991.

[22] R. Ellis, *private communication*, Sematech, Austin, Texas.

[23] P. Glarborg, R. J. Kee, J. F. Grcar and J. A. Miller, *PSR: A Fortran Program for Modeling Well-Stirred Reactors*, Sandia National Laboratories Report SAND86-8209, Livermore, California, February 1991.

[24] J. F. Grcar, *The Twopnt Program for Boundary Value Problems*, Sandia National Laboratories Report SAND91-8230, Livermore, California, February 1992.

[25] R. J. Kee, G. Dixon-Lewis, J. Warnatz, M. E. Coltrin and J. A. Miller, *A Fortran Computer Code Package for the Evaluation of Gas-Phase Multicomponent Transport Properties*, Sandia National Laboratories Report SAND86-8246, Livermore, California, 1986. Reprinted November, 1988.

[26] R. J. Kee, J. F. Grcar, M. D. Smooke and J. A. Miller, *A Fortran Program for Modeling Steady Laminar One-Dimensional Premixed Flames*, Sandia National Laboratories Report SAND85-8240, Livermore, California, December 1985.

[27] R. J. Kee, F. M. Rupley and J. A. Miller, *Chemkin-II: A Fortran Chemical Kinetics Package for the Analysis of Gas-Phase Chemical Kinetics*, Sandia National Laboratories Report SAND89-8009, Livermore, California, September 1989.

[28] M. Kent, *The Numerical Analysis Net (NA-NET)*, Institut für Informatik Report, Eidgenössische Technische Hochschule, Zurich, January, 1988.

[29] M. Loukides, *UNIX for FORTRAN Programmers*, O'Reilly & Associates, Sebastopol, California, 1991.

[30] T. MacDonald, "C for Numerical Computing," *The Journal of Supercomputing*, v. 5 n. 1 (June 1991), 31–48.

[31] J. Meyering, *private communication*, Department of Computer Science, University of Texas, Austin, Texas. Get `tmp/f-s2d-1.1.11.tar.Z` by anonymous ftp from `cs.utexas.edu`.

[32] H. K. Moffat, P. Glarborg, R. J. Kee, J. F. Grcar and J. A. Miller, *SURFACE PSR: A Fortran Program for Modeling Well-Stirred Reactors with Gas and Surface Reactions,* Sandia National Laboratories Report SAND91-8001, Livermore, California, May 1991.

[33] A. Oram and S. Talbott, *Managing Projects with make,* O'Reilly & Associates, Sebastopol, California, 1991.

[34] F. M. Rupley, *private communication*, Sandia National Laboratories, Livermore, California.

[35] D. Schneider, *private communication*, Center for Supercomputing Research and Development, University of Illinois, Urbana, Illinois.

UNLIMITED RELEASE

SECOND DISTRIBUTION

8745 J. F. Grcar (1)
8745 F. M. Rupley (200)

8535 Publications for OSTI (15)
8535 Publications/Technical Library Processes, 7141
7141 Technical Library Processes Division (3)
8524-2 Central Technical Files (3)

SUBSEQUENT DISTRIBUTION (January 1995)

8745 F. M. Rupley (200)