NASA Contractor Report 187507

Transferable Output ASCII Data (TOAD) Editor Version 1.0 User's Guide

Bradford D. Bingel, Anne L. Shea, and Alicia S. Hofler

Computer Sciences Corporation
Applied Technology Division
Hampton, VA 23666-1379

Contract NAS1-19038

February 1991

National Aeronautics and
Space Administration

Langley Research Center
Hampton, Virginia 23665-5225

Preface

This document describes the Transferable Output ASCII Data (TOAD) Editor, production release 1.0. It is intended to serve as a tutorial for new users and as a reference source for all experienced users. All readers are urged to review the sample sessions in appendix A of this document. Readers not familiar with the TOAD format should refer to appendix B or to NASA Contractor Report 178361.

Because of the ongoing development of this package, the current production release may offer features in addition to those described in this document. When and if changes are made, every effort will be made to preserve existing capabilities.

This software was developed by Computer Sciences Corporation's Applied Technology Division under contract to the National Aeronautics and Space Administration's Langley Research Center, from late 1989 through late 1990. CSC directly supports this product only at Langley Research Center.

Table of Contents

PAGE ii INTENTIONALLY BLANK

PRECEDING PAGE BLANK NOT FILMED

Table of Contents - concluded

Section 1
Introduction

General Description and Purpose

The Transferable Output ASCII Data (TOAD) Editor is a software tool for manipulating the contents of TOAD files. It offers many of the advantages of a spreadsheet program (mathematical operations, row/column manipulations, cut/copy/paste, selective data extraction/replacement, macros) without the initial purchase cost or the need to transfer the data files to a PC or Macintosh. The Editor also offers many other features (such as statistical operators and unit conversion functions) designed for manipulating scientific/engineering data which are not available from many spreadsheet packages.

Features

The most beneficial features of the TOAD Editor are:

Directive Driven - Rather than working through a long series of menus, the user enters operational commands, which the Editor performs immediately.

Aliases - Most commands and keywords have multiple aliases and/or abbreviations. This significantly reduces the number of keystrokes necessary to perform a particular operation.

On-Line Help - A complete description of each command is available through an on-line help facility. Each description contains the command's purpose, its syntax, an explanation of its keywords and/or arguments, and a list of aliases. Additional information (such as what happens when an argument is omitted) is also provided.

General Undo - Any operation which changes the TOAD file can be "undone."

Directive Files - The user may create a long directive sequence within an external file, then have it executed by providing the file's name to the Editor. Further, a directive file may in turn execute another directive file, greatly simplifying extremely long or repetitive tasks.

Macros - A repetitive sequence of directives may be grouped into a macro, then executed by simply entering the macro's name. Although macros may be defined "live" during an editing session, many users find it more convenient to define their macros in a "startup" file, which is automatically read before each session.

Symbols - A symbol is a name (e.g., "pi") assigned a numeric value. When a symbol appears where a number is expected, the symbol's numeric equivalent is automatically substituted.

Targeting - If desired, almost all directives may be restricted to work upon a select data subset. This is especially useful when merging or extracting data.

Mathematical Operations - A variety of mathematical operations are provided for advanced data manipulations. Included are the four basic arithmetic functions (addition, subtraction, multiplication, and division) along with factorials, power and root functions, logarithm and exponential functions, trigonometrics, hyperbolic trigonometrics, and various statistical functions.

Conversion Functions - Many conversion functions are provided, including conversions for angles (degrees and radians), temperatures (Celsius, Kelvin, Rankine, and Fahrenheit), and various times, lengths, masses, volumes, velocities, pressures, and energy.

AutoSave - If the user attempts to end an editing session in which the file was altered but not saved, the Editor asks if it should be saved. As a result, it is almost impossible to "forget" to save revisions.

Directive History - A limited history of the directives used during an editing session is retained. This history may be reviewed at any time, and an individual directive may be selected by index number for reexecution.

Session File - A complete history of the interactive session is recorded on a session file. This file may be edited later and used as a directive file.

Error Messages - All error messages are written in plain English. Without going to extremes, every effort has been made to identify the problem as clearly as possible.

Portability - A critical factor in this product's design is portability. The Editor successfully executes under CONVEX, IRIS, SUN, and VAX host environments.

Limitations

The TOAD Editor reads and writes entire data files, not file fragments. Consequently, there is a limit on the volume of raw data which can be accommodated. The capacity of the Editor is 1,000 "columns" of data and 10,000 raw data cells. However, these capacities may change as the Editor is installed on various hosts.

Both limits are set once in a central part of the Editor and are easily modified. If and when a file exceeds either limit, a clear error message is written.

Associated Products

A companion package, the TOAD Gateway, allows the user to translate raw data files between TOAD and other data formats, such as the Standard Interface File (SIF), Program to Optimize Simulated Trajectories (POST), and a variety of PC- and Macintosh-based spreadsheet programs. Both the TOAD Editor and the TOAD Gateway are available and supported at NASA Langley Research Center.

Section 2
Concepts

TOAD files

A TOAD file contains tabular data stored in a specific format. It is convenient to think of "tabular" data as a row-by-column table, similar to a spreadsheet. Each column of data has an associated 15-character name, called a variable. For example, a TOAD file which contains the variables *deltacp*, *temp*, *x/c*, *2y/b*, *alpha*, and *Mach* presumably stores pressure and temperature data as a function of chord location, span location, angle of attack, and Mach number.

Again using the spreadsheet analogy, a TOAD data wart is equivalent to a "row" of data. Because a row of data may require more than one 80-character record within a TOAD file, the collection of records associated with a single spreadsheet "row" is commonly called a "wart."

A tadpole file is the generic name given to a file which stores a subset of data from a TOAD file. Most of the commands which work with external files assume the name *tadpole* if the file name is omitted. Although the name may be somewhat misleading, a true tadpole file conforms to all TOAD standards.

A TOAD file becomes active when it has been opened by the Editor for processing. Only one TOAD file may be active at any one time. All other TOAD files are inactive. An active file is not really a file at all -- merely the Editor's own representation of a file. As a result, changes made to the active file exist only within the Editor and will not exist as a disc file until you perform a save operation. For example, if you open file "test21" and delete some columns of data, only the Editor's version of the file is affected -- the original disc copy is untouched.

Commands vs. Directives

The distinction between a command and a directive is often confusing. In actual practice, the terms "command" and "directive" are often used interchangeably. Strictly defined, a command is a type of instruction given to the Editor. For example, opening a TOAD file for editing is accomplished via the *open* command. A directive is the actual instruction given to the Editor. Thus the directive

open test201

is but one example of how you might use the *open* command. Many more directives using the *open* command are possible. This concept is fully discussed in Section 3, "Directive Syntax."

For the remainder of this document, the term "command" is used when referring to an element within the Editor's vocabulary, and the term "directive" is used to describe the actual instruction the Editor reads, interprets, and executes.

Startup File

A startup file is used to submit a stream of directives to the Editor before the first `edit>` prompt appears. The most common use of a startup file is to create the desired macros, symbols, and environmental settings without entering them manually during each session. For example, a simple startup file might contain the directives

```
disable session
set page 23
set tolerance 5%

define pi = 3.1415926
define e = 2.7182818

macro tab1
tabulate alpha deltacp 2y/b .95 x/c .05
endmacro

macro tab2
tabulate temp x/c .05 alpha 15
endmacro

macro fix
convert alpha degrees2radians
convert temp rankine2kelvin
endmacro
```

This example startup file turns off the session recording file, resets the page length to 23 lines, resets the default tolerance to a relative 5 percent, defines the symbols *pi* and *e*, and creates three macros. This example also illustrates two useful and highly recommended techniques. First, notice that all commands and keywords are spelled out in full, rather than abbreviated. Second, blank lines group logical instruction sets. Both features significantly improve readability.

Using a startup file is optional. On UNIX systems the file must be called *startup* and it must exist in the local directory when the Editor is executed. An alternate method is to establish a file link called *startup* which points to the desired file. VAX/VMS requirements are the same except the file name or global definition is *startup.dat*.

Note

> The directives read from the startup file do not appear in the directive history or in the session file.

Directive History

Because it is often convenient to repeat previous directives, the Editor retains a limited history of directives entered during the editing session in a directive history. UNIX users should recognize this as the standard UNIX history mechanism and VAX/VMS users should recognize this as a command recall buffer. Its contents may be displayed with the directive

```
history
```

Currently, 20 directives are retained (this may increase with future releases). If fewer than 20 directives have been entered prior to a *history* directive, only those directives are displayed. If more than 20 directives have been entered, only the last 20 are displayed.

To reexecute a previously entered directive, enter its associated index. For example, if the desired directive appears in the history list as

71. tabulate alpha deltacp 2y/b .95 x/c .05

then it may be reexecuted by entering

71

Notice that the directive image appears in brackets

[tabulate alpha deltacp 2y/b .95 x/c .05]

to confirm which directive was selected. An alternative method is to use a relative reference. For example, entering

-1

requests that the most recent directive be repeated. Similarly, the directive

-4

asks that the fourth most recent directive be repeated.

Only directives which appear on a current history list may be referenced. Directives which "scroll off" the log cannot be referenced and must be reentered.

Directive Files

Lengthy or complex editing sessions are often difficult to perform when entering all of the necessary directives by hand. An alternate approach is to create a text file containing the desired directives, in the desired order, then have the Editor read it. Such a file is called a directive file. Many users create directive files when performing the same operations within a series of TOAD files. This significantly reduces the researcher's workload while allowing the TOAD files to be consistently edited.

Using a directive file interrupts the Editor's normal interactive dialog. That is, after a directive file is invoked the Editor accepts its instructions from that directive file, not from your keyboard. You regain control only after the entire directive file is read and processed. Of course, very long or very complex directive sequences will require a commensurate amount of processing time, which may create a noticeable delay.

A directive file may itself use another directive file which may in turn use another directive file, and so on. There is no limit on the number of levels, nor any limit on the number of files per level, which can be used within a directive file hierarchy. Repetitive calls to a directive file, even from within another directive file, are allowed. However, a directive file cannot call itself; that is, directive file recursion is not allowed.

For more information concerning directive files, please refer to Section 5, "Directive Files and Macros."

Macros

Creating a macro allows you to execute a sequence of directives whenever you enter that macro's name. For example, imagine converting ten columns' worth of data from Fahrenheit to Kelvin by entering the directive *convtemp* or creating five new columns of data by entering only *newcols*. Using a macro is very similar to using a directive file -- once invoked, the Editor accepts its instructions from the macro, not your keyboard, and returns control to you after the macro is completed. Also, a macro may itself use another macro which may in turn use yet another macro, and so on. Macros and directive files may be freely intermixed -- a macro may use a directive file which may use a macro which may use a directive file, and so on.

Macros offer one substantial advantage over directive files: arguments. Unlike a directive file, in which all commands and associated parameters are known, a macro may use arguments to alter the directives processed. Further, each argument may also be assigned a default value, permitting omitted arguments when invoking the macro. In effect, creating a macro actually creates a new, customized command.

For more information concerning macros, please refer to Section 5, "Directive Files and Macros."

Session File

The session file retains all directives read and processed during the course of an editing session (except those read from the startup file). This can be particularly useful when trying to reconstruct a directive sequence for the development of a directive file or macro. A session file also verifies that the directive files or macros perform the intended sequence of directives.

Note

The Editor does not write warning or error messages to the session file.

A session file is always created. On UNIX systems the session file is called *session* and is created in the local directory. You may reroute it to a different directory by creating a file link called *session*. The file is similar under VAX/VMS except the name is *session.dat*.

Targeting and Object Lists

Targeting is a technique which allows you to restrict the actions of most directives to a specific subset of the entire TOAD file. For example, perhaps you want to calculate pressure coefficients along a wing's leading edge or tabulate fuel consumption at a particular mission milestone. You may target only those data cells containing data along the leading edge or only those cells associated with the mission milestone, then perform the desired operation. Targeting is also useful when extracting and accepting raw data from external files. Depending on the target scheme used, a single cell, a partial row, a partial column, an entire row, an entire column, or the whole active file may be moved to and from an external file.

Targeting is accomplished through the use of an object list. An object list identifies the variables being targeted, and, where necessary, their target range. For example, targeting a wing's leading edge requires an object list with, at a minimum, the name of the airfoil chord location variable and its value associated with the leading edge. If other independent variables need to be controlled (such as angle of attack) they too must be included in the object list.

A full explanation of object lists will be presented later. For now, we'll only work with a few simple examples. Suppose we have a TOAD file with the following variables:

deltacp	pressure
temp	temperature
x/c	nondimensional chord location of the control point
2y/b	nondimensional semispan location of the control point
alpha	angle of attack (in degrees)

where dependent variables *deltacp* and *temp* are functions of independent variables *x/c*, *2y/b*, and *alpha*. Further, there are multiple values of *x/c* within each value of *2y/b*, and there are multiple values of *2y/b* within each value of *alpha*. Let's also assume that there are 10 chordwise control points along each spanwise station.

To tabulate pressure along a spanwise station (let's say 80% outboard of the wing root, or *2y/b*=.8) we enter the directive

tabulate deltacp x/c 2y/b .8

which reads "tabulate all values of *deltacp* and *x/c* when *2y/b* equals .8." If there was only one angle of attack we'd see ten values, just as we expect. If, however, there are 5 angles of attack on the file, we'd see 50 values (10 x 5). To avoid this, we should also specify controlling values for *alpha*, such as

tabulate deltacp x/c 2y/b .8 alpha 10

which reads "tabulate all values of *deltacp* and *x/c* when *2y/b* equals .8 and *alpha* is 10 degrees."

Using the same file, suppose we instead want to tabulate temperature and span location along the leading edge (*x/c*=.05) at an angle of attack (*alpha*) of 15 degrees. The directive is

tabulate temp 2y/b x/c .05 alpha 15

If we want to see pressure at the tip (*2y/b*=.95) leading edge (*x/c*=.05) as a function of angle of attack (*alpha*), we use the directive

tabulate deltacp alpha 2y/b .95 x/c .05

How do we know when we may use targeting and object lists? Let's look at the help text for command *tabulate* :

```
TABULATE  displays the targeted portion of the TOAD file.

syntax:   Tabulate  [object list]

          object list   see the help text for command Target
                        If omitted, the default target list
                        is assumed.

aliases:  tabul tab type typ ty print pri
```

It says that the *tabulate* command has no parameters and may contain an optional object list. Further, when the object list is omitted, the default target list is used in its place. The <u>default</u> target list serves as a backup specification whenever a <u>direct</u> object list isn't provided. So far we haven't used a default target list, only direct object lists. To illustrate how a default target list may be used, the previous example,

```
tabulate deltacp alpha 2y/b .95 x/c .05
```

may also be entered as

```
target deltacp alpha 2y/b .95 x/c .05
tabulate
```

Why use two directives? Using *target* creates a default target list which will be used by all subsequent directives when and if a direct object list is omitted. This is particularly useful when you are performing a series of manipulations on the same data subset -- set the default target once, then let subsequent directives assume that same data subset for their operations. For example, the directive sequence

```
target temp probe_id alpha 15 30
tabulate
convert temp rankine2kelvin
tabulate
```

establishes a default target, tabulates the data subset, converts from Rankine to Kelvin those temperatures associated with 15-30 degrees angle of attack, then retabulates the data subset. Without a default target the same process would require the directives

```
tabulate temp probe_id alpha 15 30
convert temp rankine2kelvin alpha 15 30
tabulate temp probe_id alpha 15 30
```

What happens when the two types of target lists are mixed? Consider the directive sequence

```
target deltacp 2y/b .85 .95 alpha
tabulate temp x/c .05 .15 alpha 15 20
tabulate
```

The direct object list within the first *tabulate* temporarily overrides the default object list. Therefore, the first *tabulate* report contains the variables *temp, x/c,* and *alpha*. The second *tabulate* report, using the default target list, contains the variables *deltacp, 2y/b,* and *alpha*.

Section 3
Directive Syntax

We've already used a few Editor directives in our previous examples but haven't gone into much detail as to how they are constructed, what rules govern their use, and how they may be manipulated to suit your individual needs. All readers who hope to use the Editor's full capabilities must have a complete understanding of the principles presented here.

Commands, Parameters, and Keywords

An Editor *directive* begins with a *command* which may be followed by one or more *parameters* and *keywords*. For example, the directive

```
sort Mach
```

uses the command *sort* and has the parameter *Mach* (this directive uses the data in *Mach* to control sorting the file).

Each individual item is separated from its neighbors using a comma or one or more blanks. This same directive could also be entered in any of the following forms:

```
sort,Mach
sort, Mach
sort ,Mach
sort , Mach
sort   Mach
```

Keywords are used to indicate specific actions within directives. For example the directive

```
convert temp rankine2kelvin
```

uses the command *convert*, the parameter *temp*, and the keyword *rankine2kelvin* (this directive converts temperatures from a Rankine scale to a Kelvin scale).

Parameter Type

Most Editor commands accept a variety of parameter types. For example, the *sqrt* command, which calculates square roots, can be in any of the following forms:

```
sqrt flowrate
sqrt maxarea
sqrt 12
sqrt 5*7
```

where *flowrate* might be a variable within the active TOAD file, *maxarea* a symbol, *12* a numeric value, and *5*7* a simple numeric expression. Obviously the type of parameter used should be appropriate for the command. For example, a negative value is inappropriate for a square root function and a fractional value is improper for a factorial (n!) operation.

Simple Expressions

At times it is convenient to express a numeric value as a simple expression. For example, ".333333" can be entered much faster, clearer, and more accurately as "1/3". Any of the four basic arithmetic functions (+,-,*,/) may be used once within a simple expression. Thus the expressions

```
1/3
12-4.3
5.6+1.234
9*25.5
```

are all valid. Do not use parentheses (), brackets [], or braces {}.

Aliases

Aliases are alternative or abbreviated names for the same item. Commands and keywords generally have a number of aliases. Variables rarely, if ever, have aliases. For example, the directive

```
convert temp rankine2kelvin
```

could also be entered as

```
con temp r2k
```

The command *convert* has been replaced with the alias *con*, and the keyword *rankine2kelvin* has been replaced with the alias *r2k*. Notice that the variable name, *temp*, was not aliased. The on-line help facility provides a list of aliases available for each command.

Omitted Parameters

At times you may wish to omit a parameter. For example, the directive

```
scan test report
```

scans the TOAD file *test* and writes the resulting information to file *report*. If you omit the report file *report*, the directive becomes

```
scan test
```

in which case the Editor displays the report to your screen. Similarly, if you omit the TOAD file *test* the directive becomes

```
scan ,, report
```

in which case the Editor assumes the active TOAD file. In fact, the directives

```
scan,,report
scan ,, report
scan, ,report
scan , , report
```

are all equivalent. Two consecutive commas are the only way to indicate an omitted parameter or keyword.

Quotation Marks

Suppose we have a TOAD file containing the variables

```
node
inner temp
outer temp
```

How would we tabulate the inner and outer temperatures as a function of the node ID? The directive

tabulate node inner temp outer temp

asks to tabulate the variables *node*, *inner*, *temp*, *outer*, and *temp*, which isn't correct. The solution is to use quotation marks to indicate embedded blanks within a single item. For example, the directive

tabulate node 'inner temp' 'outer temp'

or

tabulate node "inner temp" "outer temp"

or

tabulate node 'inner temp' "outer temp"

asks to tabulate the variables *node*, *inner temp*, and *outer temp*, which is correct. However, the directive

tabulate node 'inner temp" "outer temp'

or

tabulate node "inner temp' 'outer temp"

improperly mixes single and double quotation marks.

Assumed Commands

Sequences of the same command may be streamlined using the "assumed command" feature. For example, the directives

```
define  pi=3.14159
define  e=2.71828
define  c=2.99793E8
```

could also be written as

```
define  pi=3.14159
  "     e=2.71828
  "     c=2.99793E8
```

Continuations

Some directives may be too long to fit within a single 80-character entry and must be continued on another line. The continuation character is the ampersand (&). For example, the directive

```
target deltacp temp x/c 2y/b .95 alpha 10
```

could also be entered as

```
target deltacp temp &
x/c 2y/b .95 alpha 10
```

or as

```
target &
deltacp temp &
x/c &
2y/b .95 &
alpha 10
```

Terminating an entry with the continuation character allows you to provide the remainder of the directive on subsequent entry lines. The Editor responds by replacing the regular `edit>` prompt with the `..edit>` prompt. Up to 800 characters (including any embedded blanks) may be entered for a single directive, spread over as many entry lines as you wish.

The continuation feature is commonly used to arrange complex directives more clearly. Our previous example directive

```
target deltacp temp x/c 2y/b .95 alpha 10
```

creates a target object list containing the dependent variables *deltacp* and *temp*, as controlled by the independent variables *x/c* (all values), *2y/b* (at 95% span), and *alpha* (at 10 degrees angle of attack). Using continuations, we might rewrite it as

```
target deltacp temp &
   x/c &
   2y/b .95 &
   alpha 10
```

which many users find easier to read.

Comments

Any entry which begins with a pound sign (#) or exclamation point (!) is assumed to be a comment and is not processed. This provides a way of including notes inside a startup file, directive file, macro, or session file. For example, our previous example startup file

```
disable session
set page 23
set tolerance 5%

define pi = 3.1415926
```

```
define e = 2.7182818

macro tab1
tabulate alpha deltacp 2y/b .95 x/c .05
endmacro

macro tab2
tabulate temp x/c .05 alpha 15
endmacro

macro fix
convert alpha degrees2radians
convert temp rankine2kelvin
endmacro
```

might be clearer if we added some comments:

```
#  TOAD Editor startup file
#
#  turn off the session recorder
#
      disable session
#
#  set the screen size and default tolerance
#
      set page 23
      set tolerance 5%
#
#  define the mathematical constants pi and e
#
      define pi = 3.1415926
      define e = 2.7182818
#
#  create two tabulation macros and one conversion macro
#
      macro tab1
      tabulate alpha deltacp 2y/b .95 x/c .05
      endmacro
#
      macro tab2
      tabulate temp x/c .05 alpha 15
      endmacro
#
      macro fix
      convert alpha degrees2radians
      convert temp rankine2kelvin
      endmacro
#
#  end of startup file
```

All comments, except those within the startup file, are passed to the session file, allowing explanations to be inserted during a long or complicated editing session.

Summary of Special Characters

The Editor reserves many characters for special purposes. They are:

or ! — When either is the first character in a directive, the entire entry is assumed to be a comment.

, — A general separator. For example, commas separate arguments within a directive or numeric values within a wart id list. Two consecutive commas indicate an omitted item.

[blank] — Also a general separator. Like commas, blanks may also be used to separate items within a list. Unlike commas, however, the number of consecutive blanks between items is insignificant. When blanks and commas are intermixed, the commas take precedence.

' — A single quotation mark is commonly used to enclose a variable name which contains an embedded blank. For example, the variable name

test panel

would normally be interpreted as two names, *test* and *panel*. Using single quotes

'test panel'

preserves the embedded blank.

" — Double quotation marks can always substitute for single quotation marks. For example, the previous variable name *test panel* could also be specified as

"test panel"

Double quotes can also be used to clarify names with an embedded single quote used as an apostrophe. For example,

RPM's

could be clarified as

"RPM's"

Finally, double quotes also indicate an assumed command. For example, the directive sequence

```
define  def1  1000
define  def2  2000
define  def3  3000
```

could also be written as

```
define  def1  1000
  "     def2  2000
  "     def3  3000
```

& The default continuation character. When the last character in a non-comment entry, an ampersand is interpreted as a continuation mark and the next entry is appended. For example, the directive

```
tabulate press temp port [1,20] model 34 run 1025
```

could be broken up into the sequence

```
tabulate press temp &
   port [1,20] &
   model 34 &
   run 1025
```

The continuation character may be changed via the *set* command.

$ The default macro character, discussed in Section 5, "Directive Files and Macros." It too may be changed via the *set* command.

Interactive Use

This section introduces most of the Editor's commands. Organized as a tutorial, it begins with the simpler ones and builds up to the more complex ones. If you are a new user and wish to learn all of the Editor's features we urge you to skim this entire section and try out new commands as they pique your curiosity. If you are an experienced user you may find the detailed information and recommendations useful.

Most of the examples in this section do not use aliases for commands or keywords. This is done to improve clarity. In reality, aliases are frequently used and have no adverse effect upon the Editor's performance or the TOAD files' contents. Likewise, the examples may not demonstrate the best way of performing a particular manipulation; many were fabricated solely for the purpose of illustrating how a specific command might be used.

Finally, remember that the terminal "screen" is actually the standard output device you have assigned. If you are working interactively the standard output device would indeed be your terminal screen. If you redirect your output it will go to a file rather than to the screen.

Files

The user is responsible for ensuring that any requested TOAD files are available to the Editor. This normally requires that you have at least "read" permission. You will need "write" permission for those TOAD files you plan to create or rewrite.

Execution

How the Editor is executed depends entirely upon the host operating system. Through the use of procedures, global definitions, or shell scripts, most installations require only that you enter

toaded

to start execution. Regardless of the host operating system, the following welcome banner appears:

```
-----------------------------------

  T O A D   F i l e   E d i t o r

    Release 1.0    October 1990

-----------------------------------
```

The release number and date will change as new versions of the Editor are installed.

To stop execution, enter the directive

end

or any of its aliases,

stop
halt
exit
exi
quit
qui
q

On-Line Help

Help is readily available. At the `edit>` prompt enter

h

or, if you prefer,

help

and a list of all commands appears. If the list stops without the `edit>` prompt, you're probably at a page break -- just press the return key to keep going. Or, if you'd rather cancel the list, enter *q* and press the return key.

At the conclusion of this help list, or anytime you're at an `edit>` prompt, you can find out more by entering the directive

h *command*

where *command* is the name of the command you want to know more about. For starters, you might inquire about *help* itself by entering

h help

to verify that *h* is indeed an alias for *help*.

The help facility was the very first module installed in the Editor and to this day it remains the best source for quick, up-to-date information. You are urged to use the on-line help facility for most of your needs and to refer to this document for those occasions when the help facility is inadequate.

Environmentals

There are a number of items which, while not directly affecting the contents of any TOAD file, do control aspects of the interactive dialog. Because they affect only the Editor's environment they are called *environmentals*. The Editor initializes all environmentals to their default settings. You may change any environmental via the *set* command. The general form is:

set *environmental value*

where *environmental* is the keyword identifying the environmental being changed and *value* is its new value or state. Similarly, you may display any environmental's current setting via the *show* command. Its general form is:

show *environmental*

where *environmental* is the keyword identifying the environmental being displayed. For on-line assistance with either the *set* or *show* command, or to see a list of the aliases for any environmental keyword, use the on-line help facility:

h set
h show

There are three types of environmentals: numeric, text, and toggle. Numeric environmentals contain whole or fractional constants (e.g., the default tolerance). Text environmentals contain a single character (e.g., the directive continuation character). Toggle environmentals are turned "on" or "off" (e.g., the AutoSave protection toggle). Each group of environmentals is individually presented below.

Two numeric environmentals are available: page length and the default tolerance. **Page length** is the number of text lines displayed before a page break occurs. For example, without a page length, if your screen had a capacity of 24 lines, and a tabulate directive created 50 lines, you'd watch the first 26 lines scroll right off the screen. Using the default page length of 20, the tabulation breaks every 20 lines, then waits for your signal. Pressing the return key continues the tabulation -- entering *q* and then pressing the return key stops the tabulation (but not the Editor).

To change the page length, enter

set page *n*

where *n* is the number of lines your screen can handle. Zero, negative, or fractional page lengths are not accepted. Batch users may prefer to remove the page size limit (and avoid unexpected prompts to continue with a display) by using the directive

set page unlimited
or
set page nolimit

To display the current page length (whether you altered it or not), use the directive

show page

The **default tolerance** is used whenever an item within a targeting object list omits a tolerance (more on this later). There are two types of tolerances: absolute and relative. An absolute tolerance is an unvarying quantity. For example, the specification "10 plus or minus 5" creates the interval [5,15]. A relative tolerance varies according to its target value. For example, the specification "10 plus or minus 5%" creates the markedly different interval [9.5,10.5]. Initially, the Editor establishes the default tolerance to be relative, at 1%.

To declare a new absolute tolerance, enter the directive

set tolerance *value*

where *value* is the new default tolerance. Zero or negative tolerances are not accepted. To declare a new relative tolerance, the directive is

set tolerance *value* **%**

Note

The only difference between declaring an absolute tolerance and declaring a relative tolerance is the inclusion of the percent sign (%).

To display the default tolerance and its type (whether you changed it or not), use the directive

show tolerance

Two text environmentals are available: the continuation character and the macro character. The **continuation character** is used to mark the end of an entry which continues on the next line. It is initially set to an ampersand (&). For example, the entries

target deltacp &
x/c 2y/b .94

are interpreted as the single directive

target deltacp x/c 2y/b .94

because the continuation character (&) appears at the end of the first entry. You may change the continuation character to any other character by entering

set contchar *x*

where *x* is the new continuation character. To restore the original continuation character, enter

set contchar &

or, more simply,

restore contchar

which assumes that you want to restore the previous continuation character.

The **macro character** is used to mark dynamic variables within a macro definition. It is initially set to a dollar sign ($). A full discussion of using the macro character is presented in Section 5, "Directive Files and Macros." For now, our concern is how it can be changed using the *set* command:

set macrochar *x*

where *x* is the new macro character. To restore the original macro character, enter

set macrochar $

or, more simply,

restore macrochar

which assumes that you want to restore the previous continuation character.

There are nine toggle environmentals: MacroEcho, OverWrite, AutoSave, EntryEcho, ShoWartList, InfoMess, Session, Expand, and History. In general, each may be turned on (enabled) by entering

set *toggle* **on**

or

set *toggle* **yes**

or

set *toggle* **true**

or

enable *toggle*

where *toggle* is the keyword identifying the toggle you want changed. Similarly, any may be turned off (disabled) by entering

set *toggle* **off**

or

set *toggle* **no**

or

set *toggle* **false**

or

disable *toggle*

The current state of any toggle may be displayed by entering

show *toggle*

and the current states of all toggle environmentals may be displayed by entering

show toggles

or

show indicators

or

show states

or

show flags

or

show switches

Each toggle's purpose is discussed in the following paragraphs.

Toggle **MacroEcho** controls whether directives are echoed during the execution of a user-defined macro. When enabled, each directive in the macro's script is displayed, in brackets, as it is performed during the macro's execution. When disabled, no such information is provided. It is initially enabled.

Note

Enabling the MacroEcho toggle automatically enables the OverWrite and AutoSave toggles.

Toggle **OverWrite** controls whether you are prompted for a confirmation when you ask to overwrite an existing external file. When enabled, the prompt

```
This request will overwrite the original contents of
an existing file.  Do you really want it performed ?
```

appears whenever a directive attempts to overwrite an existing file. Entering "yes" tells the Editor to go ahead and overwrite the file. Entering "no" instructs it not to perform that directive. When the OverWrite toggle is disabled, no such prompt appears and external files are overwritten without warning. It is initially enabled.

Note

The OverWrite toggle is automatically enabled when the MacroEcho toggle is enabled.

Toggle **AutoSave** controls the built-in safety feature which keeps you from inadvertently stopping the Editor without saving changes made to the active TOAD file. When enabled, the prompt

```
The active TOAD file's contents have not been saved.
Do you really want your last command performed ?
```

appears whenever you attempt to end an editing session in which you've altered the active TOAD file's contents without first saving your changes to an external TOAD file. Entering "yes" instructs the Editor to go ahead and end the session. Entering "no" keeps the editing session active, giving you a chance to save the changes. When the AutoSave toggle is disabled, no such prompt appears. It is initially enabled.

Note

The AutoSave toggle is automatically enabled when the MacroEcho toggle is enabled.

Toggle **EntryEcho** controls whether or not directives are echoed as they are read. When enabled, every directive accepted, whether from the keyboard or from a directive file, is echoed back to your screen (or whatever you have assigned as the standard output device). This is above and beyond the normal echoing provided by the host operating system. When the EntryEcho toggle is disabled, the directives are not echoed. It is initially disabled.

Helpful Hints

When the EntryEcho toggle is enabled and you enter a directive interactively, the operating system echoes the directive as you type it and the Editor echoes it after you press the RETURN key, in effect echoing the directive twice. We suggest leaving the EntryEcho toggle disabled during an interactive editing session.

The toggle is, however, particularly useful when using directive files. Under normal circumstances, the Editor displays little if any progress information after you've started executing the contents of a directive file. If, at the beginning of the directive file, you enable the EntryEcho toggle, each directive is displayed as it is executed, providing a live report of the Editor's progress. We highly recommend this practice, and offer the following as a pattern for all of your directive files:

Enable entryecho
directive
directive
.
.
directive
Disable entryecho

Additional information regarding directive files can be found in Section 5, "Directive Files and Macros."

Toggle **ShoWartList** controls the format of wart ID target list reports. When enabled, a full wart ID list is displayed in response to a *show target* directive. When disabled, a full list may or may not appear, depending upon its size. A more detailed description of this toggle is presented with the *target* command, described later in this section. This toggle is initially disabled.

Toggle **InfoMess** controls whether or not informative messages are written after select operations. For example, with the InfoMess toggle enabled, a *tabulate* command displays the requested data wart subsets and then tells you how many were displayed. Similarly, mathematical commands, such as *divide*, perform their operations and then tell you how many data warts were changed and how many improper operations (e.g., dividing by zero or finding the square root of a negative value) were attempted. When the InfoMess toggle is disabled, no such messages appear. This toggle is initially enabled.

Toggle **Session** controls the "door" for the session file. When enabled, all directives interpreted are written to the session file ("open door"). When disabled, no directives are routed to the session file ("closed door"). The Session toggle is initially enabled.

As an illustration, the directives

set page 23
set tolerance 5%
disable session
define pi = 3.1415926
define e = 2.7182818
macro tab1
tabulate alpha deltacp 2y/b .95 x/c .05
endmacro
enable session
macro tab2
tabulate temp x/c .05 alpha 15
endmacro

create the session f le

set page 23

```
set tolerance 5%
disable session
macro tab2
tabulate temp x/c .05 alpha 15
endmacro
```

Toggle **Expand** controls whether or not directives executed as a result of using a macro appear in the session file. A full discussion of this toggle is presented in Section 5, "Directive Files and Macros." It is initially disabled.

Toggle **History** is very similar to the Session toggle. When enabled, interpreted directives are written to the directive history. When disabled, the directive history remains idle. In other words, if you're entering a series of directives which you may later want to repeat via the directive history, the History toggle should be enabled. If, on the other hand, you're entering a series of directives which you'd rather not have displace the current contents of the directive history, the History toggle should be disabled. Initially, the History toggle is enabled.

Symbols

There are times when you may wish to use a session variable or symbol to represent numerical data. For example, accurate values for pi and e are troublesome if they must be entered whenever needed. Instead, you may create a symbol, which is automatically replaced with its numeric equivalent. For example, to create a symbol for pi, enter

```
define pi 3.1415926
```

or

```
define pi = 3.1415926
```

Then, when the symbol *pi* appears where a number is expected, it is automatically converted. As another example, the directives

```
define epsilon = .001
set tolerance epsilon
```

create the symbol *epsilon* and then use its value to set the default tolerance. Similarly, the directives

```
define angle 20
target 2y/b .95 alpha angle
```

create the symbol *angle* and then use its value to create a target object list of 2y/b=.95 and alpha=20.

To change the value of a symbol, use the *redefine* command. For example, the directive

```
redefine alpha 30
```

assigns a new value to the symbol *alpha*. Although *redefine* is designed to assign new values to existing symbols, it also creates new symbols. For example, the directive

```
redefine mach .6
```

assigns the value .6 to the symbol *mach*. If *mach* already existed, it is reassigned. If *mach* did not already exist, it is created.

To display all existing symbols, enter

show symbols

To display the value of any symbol, use the directive

show symbol *symbol*

where *symbol* is the name of the symbol to be displayed.

To rename an existing symbol, use the directive

rensymbol *old_name new_name*

where *old_name* is the name of the symbol being renamed and *new_name* is its new name. Both parameters are required -- the Editor cannot make any assumptions if either or both are omitted. In addition, *old_name* must be an existing symbol; *new_name* cannot be an existing symbol, the name of an active file variable, or a numeric image (e.g., "4").

To delete an existing symbol, use the directive

delsymbol *symbol*

where *symbol* is the name of the symbol to be deleted.

Warnings

When used to set other variables (such as the default tolerance or target angle of attack, as shown above), the symbol is converted to a numeric constant. Therefore, setting the default tolerance to symbol *epsilon* actually sets it to the value .001, not to the symbol *epsilon*. Subsequent changes to the symbol *epsilon* will not affect the retained value for the default tolerance.

You cannot create a symbol whose name duplicates one of the active file's variable names. However, you can create a symbol before opening a TOAD file containing a variable with the same name. This could create severe problems when using target object lists. When this situation occurs the Editor writes a warning to your screen but does not remove the conflicting symbol definition. We strongly urge you to either avoid this situation altogether or, at the very least, change the symbol's name or the variable's name within the active TOAD file.

File Operations

A number of commands are available for manipulating entire files. The ***open*** command initiates a TOAD file for editing. The ***save*** command writes the active file back to disc. The ***close*** command aborts the editing session without retaining any changes. Commands ***scan, report,*** and ***menu*** provide information about TOAD files. Each of these commands is individually presented.

You must open a TOAD file before any editing operations can be performed. To open a TOAD file, use the directive

open *file*

where *file* is the name of the TOAD file being opened. If your TOAD file contains variable names which are center- or right-justified, *open* automatically converts them to left-justified. If any of the file's variable names are blank, duplicates of each other, match existing symbol names, or could be mistaken for a numeric image, a brief warning message appears.

The *open* command creates an active TOAD file. As explained in Section 2, "Concepts," an active TOAD file really isn't a file at all, merely the Editor's internal representation of a file. Thus *open* creates a copy of the TOAD within the Editor and it is this copy which is subsequently affected, not the original disc file. For this reason, if changes made to the active TOAD file are to be retained you must specifically request that it be written back to a disc file. To save the active TOAD file, enter the directive

save

or

save *file*

where *file* is the name of the file being written. If the file name is omitted, the TOAD file originally opened is assumed. If the named file does not exist, it is created. If the file does exist, the message

```
This request will overwrite the original contents of
an existing file.  Do you really want it performed ?
```

may appear, depending upon the state of the OverWrite protection toggle. Answering "yes" instructs the Editor to overwrite the file. Entering "no" instructs it to ignore the previous *save* command.

Helpful Hints

Periodic saves are highly recommended during editing sessions. The associated disc file is kept updated and a recent backup file is available in case you make a severe change by mistake.

If you make such periodic backups to the same external file, the above message about overwriting an existing file appears repeatedly. To disable the built-in safety feature which issues this warning, use the directive

disable overwrite

For more information concerning the external file overwrite protection, refer to the discussion of the OverWrite toggle on page 21.

Warning

Once a TOAD file is rewritten, its original contents are lost and cannot be recovered. If the original contents of a TOAD file are to be retained for future use, we strongly recommend you first make a copy of the file, before executing the Editor.

Under certain circumstances, such as immediately after a catastrophic error, it may be advantageous to abandon the current active file and start anew. To do this, enter the directive

close

Unlike *save*, the *close* command does not retain the changes made to the active file. It should only be used to abandon the current active file. If the file was altered during the session, the message

```
The active TOAD file's contents have not been saved.
Do you really want your last command performed ?
```

may appear, depending upon the state of the AutoSave protection toggle. Answering "yes" instructs the Editor to go ahead and abandon the current active file. Entering "no" instructs it to ignore the *close* command and retain the current active file.

Once a TOAD file has been opened, you may ask for a descriptive report by entering

scan

which creates a report in the form

```
This TOAD file contains 6 variables:

    mach            cldes           planform
    x/c             2y/b            deltacp

. . . and has a total of 150 data warts.
```

This report indicates that the variable order within this TOAD file is *mach*, *cldes*, *planform*, *x/c*, *2y/b*, and *deltacp*, listed from left to right. Command *scan* can also be used on external TOAD files by entering

scan *tfile*

where *tfile* is the name of the external TOAD file to be scanned. In addition, you may ask the resulting report to be written to a file, rather than to your screen. The directive is

scan *tfile rfile*

where *tfile* is the name of the external TOAD file to be scanned and *rfile* is the name of the external file in which to write the report. If you want to scan the active file, and write the report to an external file, you must omit the TOAD file name, as illustrated in the directive

scan,,report

Notice that the omitted parameter is marked with two consecutive commas.

The *scan* command and the concept of the active file are closely related. For example, if you open a TOAD file called "test21" and then enter the directives

scan
scan test21

two identical reports are displayed. However, if you then delete a column from the active file, then reenter the same directives, two different reports are displayed. Why? Because only the active file has been altered, not the corresponding disc file. If you subsequently save the active file and then reenter the directives the reports would again match, although both would lack one of the TOAD file's original variables.

In addition to *scan*, there are two other commands which provide similar information. The ***report*** command displays the number of variables and the number of data warts within the active TOAD file, but does not list the variable names. For example, the directive

report

creates the report

```
This TOAD file has

     6 variables
     150 DATA warts
```

It is useful when you do not wish to see a long list of variable names. Like *scan*, command *report* allows you to send the report to an external file, rather than to your screen. For example,

report *rfile*

where *rfile* is the name of the external file in which to write the report. If omitted, the report is displayed on your screen.

Command ***menu*** lists the variable names available within the active TOAD file, but does not display the number of data warts. Its only form is

menu

which creates a report in the form

```
This TOAD file contains 6 variables:

     mach                cldes               planform
     x/c                 2y/b                deltacp
```

Targeting and Using Object Lists

There are two ways to target data subsets: directly and by default. A *direct* target is defined and used whenever you use a directive with an explicit object list. A *default* target is defined using the *target* command and is retained and used whenever a directive omits its object list.

The object list syntax, whether in a direct or default target definition, is the same. For example, in order to display the values for pressure (*deltacp*) at an angle of attack of 20 degrees (*alpha*=20) and a wing semispan location of 95% (*2y/b*=.95), we can enter

tabulate deltacp alpha 20 2y/b .95 *direct targeting*

or

target deltacp alpha 20 2y/b .95 *default targeting*
tabulate

There are two formats for object lists: selection criteria and wart ID lists. A selection criteria object list qualifies a data wart only when its numeric contents meet some predefined guidelines. For example, a data wart qualifies when its value for angle of attack falls between 10 and 30 degrees. A wart ID object list selects data warts solely on the basis of their position within the TOAD file (e.g., the first six warts), regardless of their numeric contents.

Each form provides unique advantages. The selection criteria format performs the intended targeting even after data warts are sorted, removed, or inserted within the active file. It is sometimes called a *dynamic* target because it locates the qualifying data warts regardless of how many exist or where they are positioned within the file. On the other hand, the wart ID list format is much easier to use, assuming you already know the exact positions of the warts to be targeted. It is often called a *static* target because it does not vary as the active file's contents are reorganized. Each format is individually presented.

A selection criteria object list can be broken down into its subordinate items:

[*data specification*] [*data specification*] . . .

Each *data specification* can be further broken down into:

variable [*filter specification*]

where *variable* is one of the available TOAD file variables. The *filter specification* can be further broken down into:

minimum *maximum*

or

[*minimum, maximum*]

or

[*minimum, maximum*)

or

(*minimum, maximum*]

or

(*minimum, maximum*)

or

target_value [/ *tolerance* [%]]

If the *filter specification* is omitted, the entire column of data corresponding to the named variable is selected. For example, the object list

alpha

selects all values of variable *alpha*, which is expected since no filter was established.

There are two ways to establish a numeric range. Using a minimum and maximum value pair creates the interval directly. For example, the object list

alpha 10 30

or

alpha [10,30]

creates the interval [10,30], read "all values of alpha greater than or equal to ten, and less than or equal to thirty." Similarly, the object list

alpha [10,30)

creates the interval [10,30), read "all values of alpha greater than or equal to ten, and less than thirty."

alpha (10,30]

creates the interval (10,30], read "all values of alpha greater than ten, and less than or equal to thirty."

alpha (10,30)

creates the interval (10,30), read "all values of alpha greater than ten and less than thirty."

Either the minimum or maximum value can be replaced with a star (*), which indicates "no limit." For example, the filter specification

*** 40**

or

[*,40]

or

(*,40]

reads "all values less than or equal to forty." Similarly, the filter specification

20 *

or

[20,*]

or

[20,*)

reads "all values greater than or equal to twenty." In addition, the filter specification

*** ***

is allowed, but because it has the same effect as omitting the specification entirely, it is not recommended. That is, the object list

alpha * * 2y/b * *

is equivalent to the object list

alpha 2y/b

Using a target value and optional tolerance offers more flexibility. In general, the selection range is created as the target value, plus or minus the tolerance.

There are two very different types of tolerances: absolute and relative. An *absolute* tolerance is an unvarying quantity. For example, the filter specification

10 / 5

reads "ten plus or minus 5," which creates the interval [5,15].

Note

We strongly recommend using a space before and after the slash ("/") character when specifying a target value / tolerance pair. Without any spaces, the filter specification

10/5

is interpreted as the simple numeric expression "ten divided by five," or the value 2, creating a very different interval.

A *relative* tolerance varies according to its target value. For example, the filter specification

10 / 5%

reads "ten plus or minus five percent of ten," which creates the markedly different interval [9.5,10.5]. Initially, the Editor establishes the default tolerance to be relative, at 1%. However, you are free to change both the tolerance value and its type using the *set* command, as previously described.

If the tolerance is omitted, the interval is defined as the target value plus or minus the default tolerance. For example, the directives

```
set tol 5
target alpha 10
```

create a default target of *alpha* within the interval [5,15]. By comparison, the directives

```
set tol 5%
target alpha 10
```

create the default target of *alpha* within the interval [9.5,10.5]. Notice, however, that the directives

```
set tol 5
target alpha 10
set tol 5%
```

still results in an *alpha* interval of [5,15]. Why? Because the *target* command established the interval [5,15] before the default tolerance was changed -- therefore changing the default tolerance had no effect upon the established interval for *alpha.*

All types of filter specifications may be freely intermixed. For example, the object list

```
alpha 10 2y/b .75 .95 x/c deltacp
```

selects *alpha* at ten degrees, span locations 75% through 95%, and all values of *x/c* and *deltacp.* In use, the directives

```
target alpha 10 2y/b .75 .95 x/c deltacp
tabulate
add deltacp pzero
tabulate
```

set a default target list, tabulate *2y/b, x/c,* and *deltacp* which meet the targeting criteria, adds the value of symbol *pzero* to *deltacp* within the targeted subset, and retabulates. As another example, the object list

2y/b .85 x/c .05 alpha -20 * deltacp

selects span location 85%, chord location 5%, all values of *alpha* greater than or equal to -20 degrees, and all values of *deltacp*. In use, the directives

```
tabulate 2y/b .85 x/c .05 alpha -20 * deltacp
tabulate 2y/b .95 x/c .05 alpha -20 * deltacp
```

tabulate leading edge (chord location 5%) pressure as a function of angle of attack, at span locations 85% and 95%, respectively.

A wart ID object list can be broken down into its subordinate items:

warts mix mix

where *warts* is a wart ID list and *mix* is either a wart ID list or a variable name. Each wart ID list has the form

i[T*j*[B*k*]]

or

i[t*j*[b*k*]]

where *i* is the beginning wart index, *j* is the ending wart index, and *k* is the increment factor. For example,

13

identifies wart #13;

1t13

identifies warts 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, and 13; and

1t13b6

identifies warts 1, 7, and 13. Reversed lists may also be created. For example,

13t1b6

or

13t1b-6

yields warts 13, 7, and 1, in that order.

The wart ID lists are prepared exactly as specified, even if duplicate or overlapping ID's result. For example, the object list

1t7 7t10 8t12

creates the wart ID list 1, 2, 3, 4, 5, 6, 7, 7, 8, 9, 10, 8, 9, 10, 11, and 12. However, a negative wart ID (e.g., "-5") or a wart ID exceeding the number of data warts in the active file is not accepted.

Note

In the previous example the wart ID's 7, 8, 9, and 10 are specified twice. Does that mean these warts will be used twice? No. A target object list only "qualifies" or "disqualifies" each data wart. Thus, a data wart specified twice in an object list is treated like a data wart specified only once -- it is considered "qualified."

When variable names are not used within a wart ID object list, all columns of data are targeted. If any names are used, only the columns associated with those variables are targeted. Thus the object list

1t61b6

targets eleven data warts and all columns, while the object list

1t31b6 deltacp x/c 2y/b

targets six data warts and three columns.

Wart ID object lists are somewhat order-independent. That is, the object list

1t6 deltacp 31t36 x/c 61t66 2y/b

creates the exact same target scheme as the object list

1t6 31t36 61t66 deltacp x/c 2y/b

or

1t6 deltacp x/c 2y/b 31t36 61t66

However, among wart lists and among variable names order is significant, as the object list

1t6 31t36 61t66 deltacp x/c 2y/b

creates a different target scheme than either

1t6 31t36 61t66 2y/b x/c deltacp

or

61t66 31t36 1t6 2y/b x/c deltacp

When you open a new TOAD file for editing, the default target list is initially set to **all**, as if you entered the directive

target all

This effectively qualifies all data contained on the active TOAD file for any subsequent operations. For example, the directives

open test21
convert alpha r2d

perform the same function as the directives

open test21

convert alpha r2d all

or the directives

open test21
target all
convert alpha r2d

To display the current default target, use the directive

show target

An example of a selection criteria default target scheme is

Variable	Interval	Value	Tol
deltacp	all	-	-
x/c	all	-	-
2y/b	[.9000,.9800]	-	-
alpha	-	10	1%

where all of variables *deltacp* and *x/c* qualify, *2y/b* qualifies only over the interval [.9,.98], and *alpha* qualifies when it equals 10 degrees, plus or minus 1%.

Wart ID list target schemes appear as

```
Wart ID list:

     1, 2, 3, 4, 5, 6

Variables:   all
```

where the first six warts are targeted, including all variables. Another form is

```
Wart ID list:

     3, 4, 5, 6, 7

Variables:

     deltacp             x/c                2y/b
```

where the third through seventh warts are targeted, including only variables *deltacp*, *x/c*, and *2y/b*. If long series of warts are targeted the report may appear as

```
Wart ID list:

     1t145b6, 6t150b6

Variables:

     deltacp             x/c                2y/b
```

which uses the abbreviated notation *1t145b6* and *6t150b6* rather than create the full list. You can control this in either of two ways. First, enabling the ShoWartList toggle forces the full list to appear, regardless of its length (obviously, this is not recommended for extremely long wart ID lists). Second, you can use the keywords *full* or *brief* within the *show target* directive. For example,

show target full

asks for a full wart ID listing, such as

```
Wart ID list:

     1, 2, 3, 4, 5, 6

Variables:   all
```

regardless of how long a full listing might be. The directive

show target brief

asks for an abbreviated wart ID listing, such as

```
Wart ID list:

     1t145b6, 6t150b6

Variables:

     deltacp              x/c                 2y/b
```

regardless of how short a full listing might be.

Note

Toggle ShoWartList and the keywords *full* and *brief* affect only how wart ID list target schemes are displayed and have no effect upon how selection criteria target scheme reports are displayed.

Tabulating

There are two commands which display the contents of the active TOAD file: *tabulate* and *stats*. Command ***tabulate*** is used to display selected subsets of raw data. Command ***stats*** provides a statistical profile of a selected variable. Each command is presented individually.

Raw Data

Command ***tabulate*** displays selected sets of raw data. Its form is

tabulate [*object list*]

where the optional *object list* selects the desired data subset. If omitted, the current default target list is assumed.

The object list provides two types of information: the data subset selected and those variables to be displayed. For example, using the example file *toad1*, the directive

tabulate 2y/b .9 .94 x/c deltacp

creates the report

wart #	2y/b	x/c	deltacp
133	0.900000	0.416667E-01	6.09007
134	0.900000	0.208333	3.02826
135	0.900000	0.375000	2.12340
136	0.900000	0.541667	1.60278
137	0.900000	0.708333	1.17190
138	0.900000	0.875000	0.711813
139	0.940000	0.416667E-01	7.38284
140	0.940000	0.208333	3.76906
141	0.940000	0.375000	2.71414
142	0.940000	0.541667	2.03536
143	0.940000	0.708333	1.50435
144	0.940000	0.875000	0.937932

which tabulates pressure (*deltacp*) across all airfoil chord locations (*x/c*) at a 90-94% (.90-.94) wing semispan location (*2y/b*). This same report could also be generated using the directive

tabulate 133t144 2y/b x/c deltacp

Because of screen width limitations, only four columns of data can be displayed. If you ask to tabulate more than four columns of data (directly or via the default target list) only the first four appear.

In the above example *2y/b* and *x/c* appear to be independent variables and *deltacp* appears to be a dependent variable. When possible, *tabulate* attempts to simply the resulting report by eliminating those variables which you request to remain constant (usually independent variables). For example, the directive

tabulate 2y/b .9 x/c deltacp

creates the slightly different report

wart #	x/c	deltacp
133	0.416667E-01	6.09007
134	0.208333	3.02826
135	0.375000	2.12340
136	0.541667	1.60278
137	0.708333	1.17190
138	0.875000	0.711813

Notice that, while the numbers are the same, the variable *2y/b* doesn't appear in the report. Why? The directive specifically requests that *2y/b* remain at .9 -- *tabulate* therefore assumes the value for *2y/b* is already known and there's no need to repeat it in the report.

Helpful Hint

When you want to display more than four columns of data, consider "windowing" the active file. A simple windowing technique is illustrated in the following dialog (the bold entries indicate user input):

```
edit> open  test
edit> scan

     This TOAD file contains 6 variables:

          col1                col2                col3
          col4                col5                col6
          col7                col8

     . . . and has a total of 40 data warts.

edit> tabulate  1t20  col1  col2  col3  col4
edit> tabulate  1t20  col5  col6  col7  col8
edit> tabulate  21t40  col1  col2  col3  col4
edit> tabulate  21t40  col5  col6  col7  col8
```

which creates four "windows," each four columns wide and 20 rows long, displaying the entire contents of the active TOAD file.

Statistical Profile

Command ***stats*** displays a statistical profile of the selected data set. Its form is

stats *variable* [*object list*]

where *variable* is the name of the variable to be profiled, and the optional *object list* selects the desired data subset. If the object list is omitted, the current default target list is assumed.

The report created by *stats* displays basic statistics, including

- frequency count
- sum
- range
- minimum and maximum values
- mean and unbiased variance
- biased standard deviation and standard error

For example, using the example file *toad1*, the directive

stats deltacp all

profiles all occurrences of *deltacp* and creates the report

```
Frequency Count:        150
Sum:                  213.588
Range:                10.2899
```

```
Minimum:              0.292267
Maximum:              10.5822
Mean:                 1.42392
Variance:             2.07422        (unbiased)
Standard Dev:         1.44504        (biased)
Standard Error:       0.117987       (biased)
```

The directive

stats deltacp 2y/b .9

profiles *deltacp* only at a 90% (.9) wing semispan location (*2y/b*), which creates the report

```
Frequency Count:            6
Sum:                  14.7282
Range:                5.37825
Minimum:              0.711813
Maximum:              6.09007
Mean:                 2.45470
Variance:             3.17727        (unbiased)
Standard Dev:         1.95262        (biased)
Standard Error:       0.797154       (biased)
```

Command *stats* is particularly useful when determining which variables are constant and which are not. For example, the directive

stats mach

creates the report

```
Frequency Count:          150
Sum:                  90.0000
Range:                    0.
Minimum:              0.600000
Maximum:              0.600000
Mean:                 0.600000
Variance:                 0.         (unbiased)
Standard Dev:             0.         (biased)
Standard Error:           0.         (biased)
```

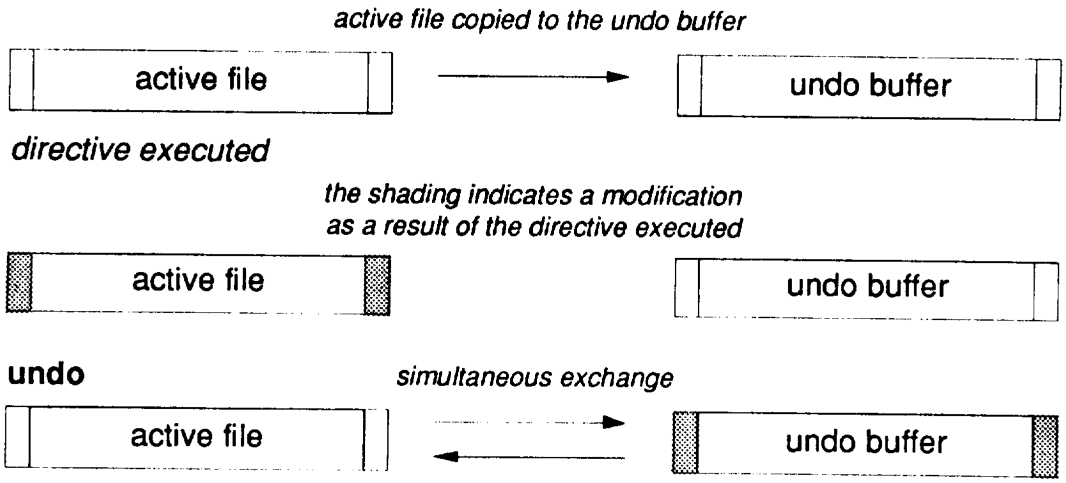
The mean of .6 and range of zero indicate that variable *mach* is constant at .6 throughout this file.

The Undo Command

Up to this point we've discussed commands which don't change the contents of an active TOAD file. Beginning with the next subsection, *Moving Data*, we will be presenting commands which have the potential of making substantial changes to the active file. But before we begin with these commands you should be aware of ***undo***.

The *undo* command allows you to restore the active file back to the state it was in immediately before the most recent directive which changed it. That is, *undo* removes the effects of the last directive which changed the active file. For example, if you open a TOAD file, then add 100 to the contents of a column of data, you've changed the active file. An *undo* would restore the active file's contents back to what they were before the add operation.

The mechanics of an *undo* are simple. The Editor copies the active file to the undo buffer when it receives a directive which looks like it would change the active file. Then it performs the directive. A subsequent *undo* simply exchanges the contents of the active file and the undo buffer. A graphic portrayal of this sequence is



We want to make four points clear. First, only a directive which changes the active file can be undone (virtually none of the commands discussed so far change the active file). For example, suppose you open a TOAD file, establish a new target (using command *target*), add a constant to a column of data (command *add*), tabulate the results (command *tabulate*), then use *undo*. The most recent command used is *tabulate*, but the most recent command which changed the active file is *add*. Therefore the *undo* command undoes the effect of *add*, not *tabulate*. (This makes intuitive sense when you consider what it means to undo a *tabulate* or *scan*.)

Second, only the most recent operation which changed the active file can be undone. For example, suppose you open a TOAD file, establish a new target, add a constant to a column of data, subtract a constant from another column of data, tabulate the result, then use *undo*. Which operation is undone? Both the add and subtract operations changed the active file, but the subtract operation is the more recent and therefore it is undone.

Third, using two consecutive *undo* commands does NOT undo the previous two operations. Because *undo* simply exchanges the contents of the active file and the undo buffer, the second *undo* undoes the effects of the first *undo*. This has two ramifications: 1) *undo* does not correct a mistake made many operations ago (again, only the most recent operation which changed the active file can be undone); and 2) *undo* itself can be undone.

Fourth, only the active file is restored to its previous state. Environmentals, symbols, and macros are not affected by *undo*. Further, changes made to the default target object list cannot be revoked via the *undo* command.

Moving Data

Copying from One Column to Another

Command ***copy*** moves data from one column to another. Its form is

> **copy** *variable1* *variable2* [*object list*]

where *variable1* identifies the source column, *variable2* identifies the destination column, and the optional *object list* selects the desired data subset. Both variables must exist within the active file and each must be unique. If the object list is omitted, the current default target list is assumed.

To illustrate how *copy* may be used, consider the following dialog:

```
edit> open  test
edit> scan

        This TOAD file contains 3 variables:

            x                   y
            z

        . . . and has a total of 5 data warts.

edit> tabulate

  wart #        x              y              z

    1        101.000        201.000        301.000
    2        102.000        202.000        302.000
    3        103.000        203.000        303.000
    4        104.000        204.000        304.000
    5        105.000        205.000        305.000

edit> copy  x  y
edit> tabulate

  wart #        x              y              z

    1        101.000        101.000        301.000
    2        102.000        102.000        302.000
    3        103.000        103.000        303.000
    4        104.000        104.000        304.000
    5        105.000        105.000        305.000

edit> copy  z  x  2t4
edit> tabulate

  wart #        x              y              z

    1        101.000        101.000        301.000
    2        302.000        102.000        302.000
    3        303.000        103.000        303.000
```

4	304.000	104.000	304.000
5	105.000	105.000	305.000

Warnings

The original contents of the column selected to receive the data are overwritten.

It is your responsibility to ensure that data from the source column is appropriate for the destination column. Improper use of the *copy* command can create worthless or misleading TOAD files.

Sorting

Command ***sort*** performs either an ascending or descending sort of the selected data subset. Its form is

sort *variable* *order* [*object list*]

where *variable* is the column of data on which the active file is sorted, *order* is "ascend" ("a") or "descend" ("d") to indicate the type of sort, and the optional *object list* selects the desired data subset. If *order* is omitted, an ascending sort is performed. If the object list is omitted, the current default target list is assumed.

When a variable is sorted it is used as a guide to reorder the entire TOAD file or the selected data subset. To illustrate this, using the example file *toad1*, consider the following dialog:

```
edit> open  toad1
edit> target  2y/b  .86  .94  x/c  deltacp
edit> tabulate
```

wart #	2y/b	x/c	deltacp
127	0.860000	0.416667E-01	5.30779
128	0.860000	0.208333	2.59059
129	0.860000	0.375000	1.82227
130	0.860000	0.541667	1.36944
131	0.860000	0.708333	0.998986
132	0.860000	0.875000	0.604424
133	0.900000	0.416667E-01	6.09007
134	0.900000	0.208333	3.02826
135	0.900000	0.375000	2.12340
136	0.900000	0.541667	1.60278
137	0.900000	0.708333	1.17190
138	0.900000	0.875000	0.711813
139	0.940000	0.416667E-01	7.38284
140	0.940000	0.208333	3.76906
141	0.940000	0.375000	2.71414
142	0.940000	0.541667	2.03536
143	0.940000	0.708333	1.50435
144	0.940000	0.875000	0.937932

```
edit> sort x/c
edit> tabulate

  wart #        2y/b            x/c            deltacp

   127        0.860000      0.416667E-01    5.30779
   128        0.900000      0.416667E-01    6.09007
   129        0.940000      0.416667E-01    7.38284
   130        0.860000      0.208333        2.59059
   131        0.900000      0.208333        3.02826
   132        0.940000      0.208333        3.76906
   133        0.860000      0.375000        1.82227
   134        0.900000      0.375000        2.12340
   135        0.940000      0.375000        2.71414
   136        0.860000      0.541667        1.36944
   137        0.900000      0.541667        1.60278
   138        0.940000      0.541667        2.03536
   139        0.860000      0.708333       0.998986
   140        0.900000      0.708333        1.17190
   141        0.940000      0.708333        1.50435
   142        0.860000      0.875000       0.604424
   143        0.900000      0.875000       0.711813
   144        0.940000      0.875000       0.937932

edit>
```

Notice that the associated values for *2y/b* and *deltacp* are moved along with *x/c* as it is sorted. In reality, entire data warts are moved according to how the guide variable is sorted, whether or not the other variables have been targeted (sorting only the guide variable, without carrying along the remainder of the data wart, would create a useless or misleading file). The target object list, therefore, serves only to restrict the vertical (row-wise) scope of the data to be sorted, and has no effect on the horizontal (column-wise) scope of the data.

Note

The *sort* command erases the default target list only if it uses the wart ID list format. Selection criteria target schemes are not affected by *sort*.

Exchanging Data Between Columns

Command ***exchange*** swaps columns of data. Its form is

exchange *variable1 variable2* [*object list*]

where *variable1* and *variable2* identify the two columns of data to be exchanged and the optional *object list* selects the desired data subset. Both variables must exist within the active file and each must be unique. If the object list is omitted, the current default target list is assumed.

Variable names are exchanged only when the entire data set is targeted (object list *all*). Otherwise, only the targeted data are exchanged and the variable names remain unaltered. To illustrate this, consider the following dialog:

```
edit> open  test
edit> scan

        This TOAD file contains 3 variables:

            x                   y
            press

        . . . and has a total of 6 data warts.

edit> tabulate

  wart #        x             y            press

    1        0.600000      0.250000      7.38284
    2        0.600000      0.750000      3.76906
    3        0.700000      0.250000      2.71414
    4        0.700000      0.750000      2.03536
    5        0.800000      0.250000      1.50435
    6        0.800000      0.750000     0.937932

edit> exchange  x  y
edit> tabulate

  wart #        y             x            press

    1        0.250000      0.600000      7.38284
    2        0.750000      0.600000      3.76906
    3        0.250000      0.700000      2.71414
    4        0.750000      0.700000      2.03536
    5        0.250000      0.800000      1.50435
    6        0.750000      0.800000     0.937932

edit> exchange  x  y  x  .7
edit> tabulate

  wart #        y             x            press

    1        0.250000      0.600000      7.38284
    2        0.750000      0.600000      3.76906
    3        0.700000      0.250000      2.71414
    4        0.700000      0.750000      2.03536
    5        0.250000      0.800000      1.50435
    6        0.750000      0.800000     0.937932
```

Warning

The second *exchange* directive in the previous dialog illustrates a situation in which data for the two independent variables (*x* and *y*) are improperly exchanged. Because the *exchange* command makes no assumptions regarding the active file's structure, it is up to you to ensure that all exchanges are proper and suitable for the file being edited. Note that the first *exchange* directive in the previous dialog is proper.

Commands *exchange* and *sort* are often used together to alter the hierarchy of the file's independent variables. For example, consider the following dialog:

```
edit> open  test
edit> scan

          This TOAD file contains 3 variables:

               x                  y
               press

          . . . and has a total of 6 data warts.

edit> tabulate

  wart #        x               y              press

    1        0.600000       0.250000         7.38284
    2        0.600000       0.750000         3.76906
    3        0.700000       0.250000         2.71414
    4        0.700000       0.750000         2.03536
    5        0.800000       0.250000         1.50435
    6        0.800000       0.750000        0.937932

edit> sort  y
edit> tabulate

  wart #        x               y              press

    1        0.600000       0.250000         7.38284
    2        0.700000       0.250000         2.71414
    3        0.800000       0.250000         1.50435
    4        0.600000       0.750000         3.76906
    5        0.700000       0.750000         2.03536
    6        0.800000       0.750000        0.937932

edit> exchange  x  y
edit> tabulate

  wart #        y               x              press

    1        0.250000       0.600000         7.38284
    2        0.250000       0.700000         2.71414
    3        0.250000       0.800000         1.50435
    4        0.750000       0.600000         3.76906
    5        0.750000       0.700000         2.03536
    6        0.750000       0.800000        0.937932

edit>
```

Originally, variable *x* was the outermost independent, and variable *y* its subordinate, with *press* as the dependent variable. After this sequence of directives, variable *y* is now the outermost independent, and variable *x* its subordinate, a complete reversal!

In addition, the order of variables may be significant. For example, when the TOAD Gateway translates TOAD files into POST (Program to Optimize Simulated Trajectories) files, it assumes the first variable to be dependent and all remaining variables to be independent, with the outermost independent as the last variable. In order to "condition" the above file for the Gateway's POST translator we need one more operation:

```
edit> exchange  y  press
edit> tabulate

  wart #       press          x            y

    1        7.38284      0.600000     0.250000
    2        2.71414      0.700000     0.250000
    3        1.50435      0.800000     0.250000
    4        3.76906      0.600000     0.750000
    5        2.03536      0.700000     0.750000
    6       0.937932      0.800000     0.750000
```

Now the file is properly conditioned for translation into the POST format.

Replacing Data

Changing a Variable's Name

Command ***rename*** allows you to change a variable name within the active TOAD file. Its form is

rename *old_name new_name*

where *old_name* is the name of the existing variable being changed and *new_name* is its new name. Both parameters are required -- the Editor cannot make any assumptions if either or both are omitted. In addition, *old_name* must be an existing variable, and *new_name* cannot be an existing variable or symbol name (commands *scan* and *menu* display a list of the active file's existing variable names; *show symbols* displays a list of the current symbols).

Using the example file *toad1*, the directive

rename deltacp pressure

changes the variable name *deltacp* to *pressure*, as illustrated in the following dialog:

```
edit> open  toad1
edit> menu

        This TOAD file contains 6 variables:

              mach                cldes               planform
              x/c                 2y/b                deltacp

edit> rename  deltacp  pressure
edit> menu
```

```
This TOAD file contains 6 variables:

        mach            cldes           planform
        x/c             2y/b            pressure

edit>
```

Erasing Data

Command ***clear*** "erases" data cells (i.e., sets them to zero) without removing data rows or columns. Its form is

clear *variable* [*object list*]

where *variable* is the name of the variable to be cleared and the optional *object list* selects the desired data subset. If the object list is omitted, the current default target list is assumed.

Using the example file *toad1*, the directive

clear deltacp 2y/b .94

substitutes zero for pressure data only at a 94% (.94) wing semispan location (*2y/b*), as illustrated in the following dialog:

```
edit> open  toad1
edit> target  2y/b  [.90,.98]  x/c  deltacp
edit> tabulate

 wart #      2y/b           x/c           deltacp

  133      0.900000     0.416667E-01     6.09007
  134      0.900000     0.208333         3.02826
  135      0.900000     0.375000         2.12340
  136      0.900000     0.541667         1.60278
  137      0.900000     0.708333         1.17190
  138      0.900000     0.875000         0.711813
  139      0.940000     0.416667E-01     7.38284
  140      0.940000     0.208333         3.76906
  141      0.940000     0.375000         2.71414
  142      0.940000     0.541667         2.03536
  143      0.940000     0.708333         1.50435
  144      0.940000     0.875000         0.937932
  145      0.980000     0.416667E-01     10.5822
  146      0.980000     0.208333         5.61437
  147      0.980000     0.375000         4.43053
  148      0.980000     0.541667         3.78733
  149      0.980000     0.708333         3.15396
  150      0.980000     0.875000         2.12793

edit> clear  deltacp  2y/b  .94
edit> tabulate
```

```
 wart #      2y/b            x/c            deltacp

  133      0.900000      0.416667E-01      6.09007
  134      0.900000      0.208333          3.02826
  135      0.900000      0.375000          2.12340
  136      0.900000      0.541667          1.60278
  137      0.900000      0.708333          1.17190
  138      0.900000      0.875000          0.711813
  139      0.940000      0.416667E-01      0.
  140      0.940000      0.208333          0.
  141      0.940000      0.375000          0.
  142      0.940000      0.541667          0.
  143      0.940000      0.708333          0.
  144      0.940000      0.875000          0.
  145      0.980000      0.416667E-01     10.5822
  146      0.980000      0.208333          5.61437
  147      0.980000      0.375000          4.43053
  148      0.980000      0.541667          3.78733
  149      0.980000      0.708333          3.15396
  150      0.980000      0.875000          2.12793

edit>
```

Direct Replacement

Command ***assign*** allows you to set one or more raw data cells to a desired numeric value. Its form is:

assign *value* [*object list*]

where *value* is the numeric value being assigned and the optional *object list* identifies the affected raw data cells. If the object list is omitted, the current default target list is assumed. As an illustration of how *assign* may be used, consider the following dialog:

```
edit> open  test
edit> scan

      This TOAD file contains 4 variables:

            col1              col2              col3
            col4

       . . . and has a total of 6 data warts.

edit> tabulate

 wart #       col1          col2          col3          col4

   1        101.0000      102.0000      103.0000      104.0000
   2        201.0000      202.0000      203.0000      204.0000
   3        301.0000      302.0000      303.0000      304.0000
   4        401.0000      402.0000      403.0000      404.0000
   5        501.0000      502.0000      503.0000      504.0000
```

6	601.0000	602.0000	603.0000	604.0000

edit> **assign 991 1 col2**
edit> **tabulate**

wart #	col1	col2	col3	col4
1	101.0000	991.0000	103.0000	104.0000
2	201.0000	202.0000	203.0000	204.0000
3	301.0000	302.0000	303.0000	304.0000
4	401.0000	402.0000	403.0000	404.0000
5	501.0000	502.0000	503.0000	504.0000
6	601.0000	602.0000	603.0000	604.0000

edit> **assign 992 3t4 col1 col2 col3**
edit> **tabulate**

wart #	col1	col2	col3	col4
1	101.0000	991.0000	103.0000	104.0000
2	201.0000	202.0000	203.0000	204.0000
3	992.0000	992.0000	992.0000	304.0000
4	992.0000	992.0000	992.0000	404.0000
5	501.0000	502.0000	503.0000	504.0000
6	601.0000	602.0000	603.0000	604.0000

edit> **assign 993 5**
edit> **tabulate**

wart #	col1	col2	col3	col4
1	101.0000	991.0000	103.0000	104.0000
2	201.0000	202.0000	203.0000	204.0000
3	992.0000	992.0000	992.0000	304.0000
4	992.0000	992.0000	992.0000	404.0000
5	993.0000	993.0000	993.0000	993.0000
6	601.0000	602.0000	603.0000	604.0000

edit> **assign 994 col4**
edit> **tabulate**

wart #	col1	col2	col3	col4
1	101.0000	991.0000	103.0000	994.0000
2	201.0000	202.0000	203.0000	994.0000
3	992.0000	992.0000	992.0000	994.0000
4	992.0000	992.0000	992.0000	994.0000
5	993.0000	993.0000	993.0000	994.0000
6	601.0000	602.0000	603.0000	994.0000

edit>

Warning

It is important to realize that the Editor alters data as instructed and that you are responsible for judging the validity of any assignment. The trivial operations performed in this example dialog are intended only to demonstrate how *clear* and *assign* are used, not when such assignments are warranted.

Mathematical Operations

Basic Arithmetic

The four basic arithmetic functions are provided via the ***add***, ***subtract***, ***multiply***, and ***divide*** commands. Each has the form

command item1 item2 [*item3*] [*object list*]

where *command* is the command name, *item1* and *item2* are the two operands used, *item3* receives the result of the operation, and the *object list* selects the desired data subset. Using command *add* as an example, the directive

add x y z

reads "x plus y yields z" and can be expressed as

x + y = z

Item1 and *item2* are variable names, symbols, or numbers. *Item3* is either a variable name or a symbol. If *item3* is omitted, the result of the operation is placed back into *item1*. For the *add* and *multiply* commands, if *item3* is omitted and *item1* is a number or numeric expression, the result is placed back into *item2*. If *item3* is omitted and both *item1* and *item2* are numbers or numeric expressions, the result is displayed on your screen. If the *object list* is omitted, the default target list is assumed.

These commands can be used in a variety of ways. For a few examples, consider the following dialog

```
edit> open  test
edit> scan

          This TOAD file contains 3 variables:

               x                    y
               z

          . . . and has a total of 4 data warts.

edit> tabulate

            wart #        x              y              z

              1        101.000        201.000        301.000
              2        102.000        202.000        302.000
              3        103.000        203.000        303.000
```

```
        4          104.000          204.000          304.000

    4 wart subsets listed.

edit> add x y z

    4 data warts were changed.

edit> tabulate

      wart #         x                y                z

        1          101.000          201.000          302.000
        2          102.000          202.000          304.000
        3          103.000          203.000          306.000
        4          104.000          204.000          308.000

    4 wart subsets listed.

edit> define ten 10
edit> mult x ten

    4 data warts were changed.

edit> tabulate

      wart #         x                y                z

        1          1010.00          201.000          302.000
        2          1020.00          202.000          304.000
        3          1030.00          203.000          306.000
        4          1040.00          204.000          308.000

    4 wart subsets listed.

edit> subtract 1000 z z 2t3

    2 data warts were changed.

edit> tabulate

      wart #         x                y                z

        1          1010.00          201.000          302.000
        2          1020.00          202.000          696.000
        3          1030.00          203.000          694.000
        4          1040.00          204.000          308.000

    4 wart subsets listed.

edit> define temp 0
edit> subtract 1000 z temp 3
```

```
edit> show  symbol  temp

            306

edit>
```

Notice in the last *subtract* example that the object list was used to select which value of *z* was used to calculate *temp*. Without this object list, four results from four subtraction operations would have been generated, one per data wart, which would have overwhelmed a single symbol and triggered an error message.

Utilities

Four utility functions are provided: ***abs***, ***invert***, ***sqrt***, ***factorial***, and ***sign***. The first four commands have the form

command item1 [*item2*] [*object list*]

where *command* is the command name, *item1* is the operand used, *item2* receives the the result of the operation, and the *object list* selects the desired data subset. Using command *abs* as an example, the directive

abs x y

reads "the absolute value of x yields y" and can be expressed as

$|x| = y$

Similarly, the *inverse*, *sqrt*, and *factorial* commands perform the functions

$1/x = y$
$(x)^{1/2} = y$
$x! = y$

respectively.

Item1 is a variable name, symbol, or number. *Item2* is either a variable name or a symbol. If *item2* is omitted, the result of the operation is placed back into *item1*. If *item2* is omitted and *item1* is a number or numeric expression, the result is displayed on your screen. If the *object list* is omitted, the default target list is assumed. In addition, the functional domains are:

abs	unlimited
invert	any nonzero value
sqrt	any positive value
factorial	positive integer less than 70

To illustrate how these commands may be used, consider the following dialog:

```
edit> open  test
edit> scan
```

```
        This TOAD file contains 3 variables:

             x                    y
             z

        . . . and has a total of 5 data warts.

edit> tabulate

          wart #        x              y              z

            1        11.0000        21.0000        31.0000
            2        12.0000        22.0000        32.0000
            3        13.0000        23.0000        33.0000
            4        14.0000        24.0000        34.0000
            5        15.0000        25.0000        35.0000

        5 wart subsets listed.

edit> mult  x  -1  y  2t4

        3 data warts were changed.

edit> tabulate

          wart #        x              y              z

            1        11.0000        21.0000        31.0000
            2        12.0000       -12.0000        32.0000
            3        13.0000       -13.0000        33.0000
            4        14.0000       -14.0000        34.0000
            5        15.0000        25.0000        35.0000

        5 wart subsets listed.

edit> abs  y

        3 data warts were changed.

edit> tabulate

          wart #        x              y              z

            1        11.0000        21.0000        31.0000
            2        12.0000        12.0000        32.0000
            3        13.0000        13.0000        33.0000
            4        14.0000        14.0000        34.0000
            5        15.0000        25.0000        35.0000

        5 wart subsets listed.

edit> invert  z
```

```
5 data warts were changed.

edit> tabulate

wart #        x              y              z

  1        11.0000        21.0000        0.322581E-01
  2        12.0000        12.0000        0.312500E-01
  3        13.0000        13.0000        0.303030E-01
  4        14.0000        14.0000        0.294118E-01
  5        15.0000        25.0000        0.285714E-01

5 wart subsets listed.

edit> factorial x y 1t3

3 data warts were changed.

edit> tabulate

wart #        x              y              z

  1        11.0000      0.399168E+08     0.322581E-01
  2        12.0000      0.479002E+09     0.312500E-01
  3        13.0000      0.622702E+10     0.303030E-01
  4        14.0000       14.0000         0.294118E-01
  5        15.0000       25.0000         0.285714E-01

5 wart subsets listed.

edit>
```

The *sign* command has the form

sign *item1* *item2* [*item3*] [*object list*]

where *item1* is the magnitude used, *item2* supplies the sign information, *item3* receives the the result of the operation, and the *object list* selects the desired data subset.

Because the sign operation may be new to many readers, a brief description is in order. The operation

sign(a1,a2)

is interpreted as "combine the magnitude of a1 and the sign of a2." Similarly, the directive

sign x y z

reads "combine the magnitude of x with the sign of y and store the result in z."

Item1 and *item2* are variable names, symbols, or numbers. *Item3* is either a variable name or a symbol. If *item3* is omitted, the result of the operation is placed back into *item1*. If *item3* is omitted and *item1* is a number or numeric expression, the result is displayed on your screen. If the *object list* is omitted, the default target list is assumed. In practice *item1* is usually the numeric value 1.

The *sign* command is somewhat specialized and at times may be indispensable. For example, suppose we are given an aircraft performance data file which contains the square of the angle of attack, both positive and negative values, which we must convert to just plain angle of attack. How can we adjust the angle of attack while preserving its sign? Consider the following dialog:

```
edit> open  test1
edit> tabulate  press  portid  1  2  alpha

 wart #        press        portid        alpha

  101         6.09007         1          -25.0000
  102         7.38284         2          -25.0000
  201         3.02826         1          -4.00000
  202         3.76906         2          -4.00000
  301         2.12340         1           4.00000
  302         2.71414         2           4.00000
  401         1.60278         1           25.0000
  402         2.03536         2           25.0000
  501         1.17190         1           100.000
  502         1.50435         2           100.000
  601         0.711813        1           225.000
  602         0.937932        2           225.000

edit> sign  portid  alpha
edit> tabulate  press  portid  -2  2  alpha

 wart #        press        portid        alpha

  101         6.09007        -1          -25.0000
  102         7.38284        -2          -25.0000
  201         3.02826        -1          -4.00000
  202         3.76906        -2          -4.00000
  301         2.12340         1           4.00000
  302         2.71414         2           4.00000
  401         1.60278         1           25.0000
  402         2.03536         2           25.0000
  501         1.17190         1           100.000
  502         1.50435         2           100.000
  601         0.711813        1           225.000
  602         0.937932        2           225.000

edit> abs  alpha
edit> sqrt  alpha
edit> tabulate  press  portid  -2  2  alpha

 wart #        press        portid        alpha

  101         6.09007        -1           5.00000
  102         7.38284        -2           5.00000
  201         3.02826        -1           2.00000
  202         3.76906        -2           2.00000
  301         2.12340         1           2.00000
  302         2.71414         2           2.00000
```

401	1.60278	1	5.00000
402	2.03536	2	5.00000
501	1.17190	1	10.0000
502	1.50435	2	10.0000
601	0.711813	1	15.0000
602	0.937932	2	15.0000

```
edit> sign alpha portid
edit> abs portid
edit> tabulate press portid 1 2 alpha
```

wart #	press	portid	alpha
101	6.09007	1	-5.00000
102	7.38284	2	-5.00000
201	3.02826	1	-2.00000
202	3.76906	2	-2.00000
301	2.12340	1	2.00000
302	2.71414	2	2.00000
401	1.60278	1	5.00000
402	2.03536	2	5.00000
501	1.17190	1	10.0000
502	1.50435	2	10.0000
601	0.711813	1	15.0000
602	0.937932	2	15.0000

Powers and Roots

Two functions are available: ***power*** and ***root***. Each has the form

command item1 item2 [*item3*] [*object list*]

where *command* is the command name, *item1* and *item2* are the operands used, *item3* receives the the result of the operation, and the *object list* selects the desired data subset. Using command *power* as an example, the directive

power x y z

reads "x raised to the y^{th} power yields z" and can be expressed as

$$(x)^y = z$$

Similarly, the *root* command performs the function

$$(x)^{1/y} = z$$

Item1 and *item2* are variable names, symbols, or numbers. *Item3* is either a variable name or a symbol. If *item3* is omitted, the result of the operation is placed back into *item1*. If *item3* is omitted and *item1* is a number or numeric expression, the result is displayed on your screen. If the *object list* is omitted, the default target list is assumed. In addition, the general functional domains are:

power	x: any nonzero value y: unlimited
root	x: any nonzero value y: any nonzero value

There are further limitations. Roots of negative values are allowed only when the order (y) is an odd integer. Fractional powers of negative values are allowed only when the order (y) is the inverse of an odd integer. All other instances involving negative values are illegal.

Logarithms and Exponents

Four functions are provided: ***log***, ***log10***, ***exp***, and ***exp10***. Each has the form

command item1 [*item2*] [*object list*]

where *command* is the command name, *item1* is the operand used, *item2* receives the the result of the operation, and the *object list* selects the desired data subset. Using command *log* as an example, the directive

log x y

reads "the natural log of x yields y" and can be expressed as

$\ln(x) = y$

Similarly, the *log10*, *exp*, and *exp10* commands perform the functions

$\log_{10}(x) = y$

$e^x = y$

$10^x = y$

respectively.

Item1 is a variable name, symbol, or number. *Item2* is either a variable name or a symbol. If *item2* is omitted, the result of the operation is placed back into *item1*. If *item2* is omitted and *item1* is a number or numeric expression, the result is displayed on your screen. If the *object list* is omitted, the default target list is assumed. In addition, the functional domains are:

log	any positive value
log10	any positive value
exp	unlimited
exp10	unlimited

Trigonometry

A full set of trigonometric functions is provided: ***sin***, ***cos***, ***tan***, and their inverses. Each has the form

command item1 [*item2*] [*object list*]

where *command* is the command name, *item1* is the operand used, *item2* receives the the result of the operation, and the *object list* selects the desired data subset. Using command *sin* as an example, the directive

sin x y

reads "the sine of x yields y" and can be expressed as

sin(x) = y

Similarly, the other commands perform the functions

cos(x) = y
tan(x) = y

respectively.

Item1 is a variable name, symbol, or number. *Item2* is either a variable name or a symbol. If *item2* is omitted, the result of the operation is placed back into *item1*. If *item2* is omitted and *item1* is a number or numeric expression, the result is displayed on your screen. If the *object list* is omitted, the default target list is assumed. In addition, the functional domains are:

sin	unlimited
cos	unlimited
tan	unlimited

All three of these commands assume the incoming angle (x) to be in radians. The mirror commands for angles in degrees are: *sind*, *cosd*, and *tand*.

The inverse functions are called *arcsin*, *arcsind*, *arccos*, *arccosd*, *arctan*, and *arctand*, corresponding to the functions

$\sin^{-1}(x) = y$
$\cos^{-1}(x) = y$
$\tan^{-1}(x) = y$

respectively. Their functional domains are:

arcsin	[-1,1]
arcsind	[-1,1]
arccos	[-1,1]
arccosd	[-1,1]
arctan	unlimited
arctand	unlimited

Hyperbolic Trigonometry

A full set of hyperbolic trigonometric functions is also provided: ***sinh***, ***cosh***, ***tanh***, and their inverses. Each has the form

command item1 [*item2*] [*object list*] where *command* is the command name, *item1* is the operand used, *item2* receives the the result of the operation, and the *object list* selects the desired data subset. Using command *sinh* as an example, the directive

sinh x y

reads "the hyperbolic sine of x yields y" and can be expressed as

$$\sinh(x) = y$$

Similarly, the other commands perform the functions

$$\cosh(x) = y$$
$$\tanh(x) = y$$

respectively.

Item1 is a variable name, symbol, or number. *Item2* is either a variable name or a symbol. If *item2* is omitted, the result of the operation is placed back into *item1*. If *item2* is omitted and *item1* is a number or numeric expression, the result is displayed on your screen. If the *object list* is omitted, the default target list is assumed. In addition, the functional domains are:

sinh	unlimited
cosh	unlimited
tanh	unlimited

The inverse functions are called *arcsinh*, *arccosh*, and *arctanh*, corresponding to the functions

$$\sinh^{-1}(x) = y$$
$$\cosh^{-1}(x) = y$$
$$\tanh^{-1}(x) = y$$

respectively. Their functional domains are:

arcsinh	unlimited
arccosh	greater than or equal to 1
arctanh	(-1,1)

Statistics

The following descriptive statistics can be generated from the raw data:

Command	Description
frequency	# of raw data cells qualified by the target object list.
sum	summation of the targeted values.
minimum	minimum value contained in the targeted set.
maximum	maximum value contained in the targeted set.
range	difference between the minimum and maximum values.
mean	average value -- an unbiased estimate for the mean (μ).
variance	an unbiased (n) estimate for the variance (σ^2).

stdeviation a biased (n-1) estimate for the standard deviation (σ).
sterror a biased (n-1) estimate for the standard error.

Most have the form

command variable [*symbol*] [*object list*]

where *command* is the command name, *variable* identifies the column supplying the raw data for the operation, *symbol* is the symbol to receive the statistic, and the *object list* selects the desired data subset. If the symbol is omitted, the result will appear on your screen. If the *object list* is omitted, the default target list is assumed. Using command *max* as an example, the directive

max temp hottest

presumably searches all temperature data (*temp*) and puts the highest value in symbol *hottest*. Similarly, the directives

variance temp m6 mach 6
variance temp m9 mach 9

presumably processes temperatures (*temp*) at Mach 6 and Mach 9 and puts the variance in symbols *m6* and *m9*, respectively.

The *freq* command has a different form:

freq [*symbol*] [*object list*]

where *symbol* is the symbol to receive the frequency count (always in integer) and the *object list* selects the desired data subset. If the symbol is omitted, the result will appear on your screen. If the *object list* is omitted, the default target list is assumed. For example, the directive

freq nwarts all

counts the number of data warts available from the active file. A more realistic example directive is

freq nhot temp [1000,*]

which presumably counts the number of temperature readings over 1000 degrees and puts the result in symbol *nhot*.

The statistical commands are unique because they only place results in symbols -- a variable name cannot receive the statistic. Why? Unlike the other mathematical commands, which generate a result for each data cell or data pair processed, the statistical commands generate an aggregate result for all data cells processed (i.e., what is the variance of a single raw data cell?). As an illustration, the directive

log temp temp2

is interpreted as each temperature generating another, logarithmic temperature. However, a similar case for *variance* might look like

variance temp temp2 ?

How do we interpret this? It looks like the variance of each temperature reading is placed in another data column, but that doesn't make sense. A variance is a single value describing one attribute of a collection of raw data, rather than of a single piece of raw data. Therefore it doesn't make sense for *variance* to pair up an operand data column with a results data column. It does make sense to provide some way of capturing the single quantity generated, and that's why symbols are used.

Another source of confusion is the notion of "biased" and "unbiased" estimators. An *unbiased* estimator is not adjusted to compensate for small sample sets. For example, the sample set average calculated using *mean* is unbiased because it is the sum of the raw data, divided by the number of raw data cells used (*n*). The variance is also an unbiased estimator. In contrast, the standard deviation and standard error are *biased* estimators, meaning that both are adjusted (divided by (*n-1*) rather than by *n*) to compensate for small sample sets. Why compensate? An unbiased standard deviation tends to underestimate the larger population's true standard deviation -- the smaller the sample set, the more pronounced the error. Biasing the standard deviation and standard error eliminates much of this discrepancy and significantly improves their reliability.

Conversions

Command ***convert*** provides an assortment of functions for converting units of measure. Its form is

convert *item1 function item2* [*object list*]

where *item1* is the operand used, *function* is the desired conversion function, *item2* receives the the result of the conversion, and the *object list* selects the desired data subset. Using the conversion function *m2ft* (meters to feet) as an example, the directive

convert x m2ft y

reads "converting x from meters to feet yields y" and can be expressed as

x * 3.280839895 = y

Item1 is a variable name, symbol, or number. *Item2* is either a variable name or a symbol. If *item2* is omitted, the result of the operation is placed back into *item1*. If *item2* is omitted and *item1* is a number or numeric expression, the result is displayed on your screen. If the *object list* is omitted, the default target list is assumed.

Many conversion functions are available, including

degrees	<--->	radians
Celsius	<--->	Fahrenheit
Fahrenheit	<--->	Rankine
Rankine	<--->	Kelvin
Kelvin	<--->	Celsius
kilometers	<--->	miles
meters	<--->	feet
millimeters	<--->	inches
kilograms	<--->	pounds
liters	<--->	gallons
seconds	<--->	minutes
minutes	<--->	hours
mph	<--->	knots
mph	<--->	fps

BTU's	<--->	joules
Pascals	<--->	psi
Pascals	<--->	atmospheres
Pascals	<--->	mmHg

A current list of the conversion functions is displayed when you enter the directive

help convert

Adding and Deleting Data

Creating a New Column

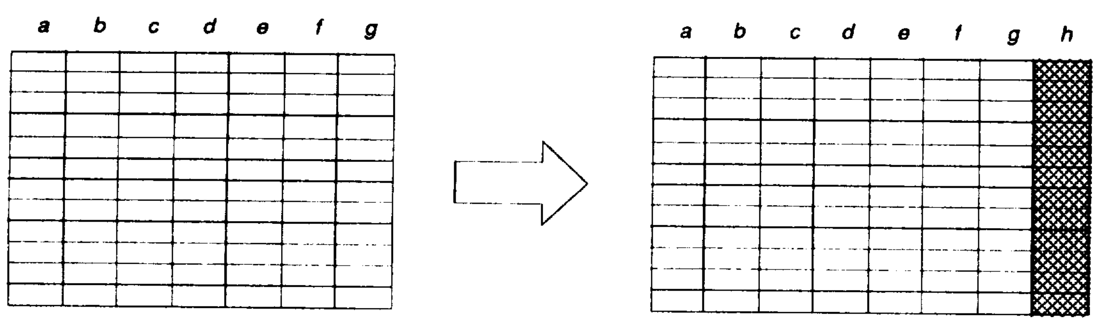
Command ***create*** establishes a new data column and a new variable within the active TOAD file. Its form is

create *variable* [*preset value*]

where *variable* is the name of the variable being created and *preset value* is the numeric value to which all new data cells are initialized. The new variable name must be unique among existing variable and symbol names. If the optional preset value is omitted, all data cells are initialized to zero. For example, the directive

create cl_ratio 1

creates a new variable called *cl_ratio* and presets all values to 1. Using a spreadsheet analogy, a *create* operation can be portrayed as



where *h* is the new column created.

Because creating a new variable increases the overall size of the active TOAD file, it's possible to exceed either the variable capacity (number of columns) or the raw data capacity (number of data cells) of the Editor. If the request would cause the capacity to be exceeded, the Editor writes the message

```
Unable to create this variable - Insufficient capacity.

Only n additional raw data can be accommodated
or all n data columns are already full.
```

and does not perform the *create* request. If the current capacity prevents you from effectively using the Editor, we suggest you follow the instructions presented in Section 6, "In Case of Problems."

The *create* command is particularly useful when performing operations on variables whose original values are to be retained. For example, suppose you had a column of temperature data, measured in degrees Kelvin, which you want to retain and use to make another column of temperature data, measured in degrees Celsius:

```
edit> open  test
edit> scan

        This TOAD file contains 2 variables

             eta                    temp

        . . . and has a total of 9 data warts

edit> tabulate

  wart #        eta           temp

    1        0.100000        1303.72
    2        0.200000        1285.43
    3        0.300000        1231.13
    4        0.400000        1142.45
    5        0.500000        1022.10
    6        0.600000        873.736
    7        0.700000        701.860
    8        0.800000        511.696
    9        0.900000        309.024

edit> rename  temp  kelvin
edit> create  celsius
edit> scan

        This TOAD file contains 3 variables

             eta                    kelvin
             celsius

        . . . and has a total of 9 data warts

edit> tabulate

  wart #        eta           kelvin          celsius

    1        0.100000        1303.72           0.
    2        0.200000        1285.43           0.
    3        0.300000        1231.13           0.
    4        0.400000        1142.45           0.
    5        0.500000        1022.10           0.
    6        0.600000        873.736           0.
    7        0.700000        701.860           0.
```

8	0.800000	511.696	0.
9	0.900000	309.024	0.

```
edit> convert  kelvin  kelvin2celsius  celsius
edit> tabulate
```

wart #	eta	kelvin	celsius
1	0.100000	1303.72	1030.57
2	0.200000	1285.43	1012.28
3	0.300000	1231.13	957.977
4	0.400000	1142.45	869.302
5	0.500000	1022.10	748.953
6	0.600000	873.736	600.586
7	0.700000	701.860	428.710
8	0.800000	511.696	238.546
9	0.900000	309.024	35.8738

After changing the variable name from *temp* to *kelvin*, a new variable, *celsius*, is created. The temperature data is then converted from the Kelvin scale to the Celsius scale.

There is an additional item worth mentioning. When variable *celsius* was created it automatically qualified in the default target list *all*. Had the default target list been anything other than *all*, the new variable would not have been added, and the directive

tabulate eta kelvin celsius

would have been necessary.

Deleting an Existing Column

Command ***delete*** removes an existing variable within the active TOAD file. Its form is

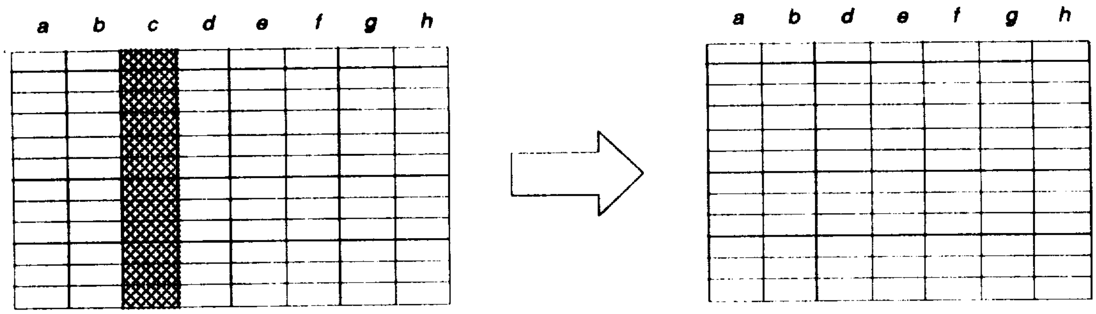
delete *variable*

where *variable* is the name of the existing variable being deleted.

The *delete* command removes the entire column of data associated with the named variable. For example, the directive

delete alpha

removes the variable *alpha*, it's associated column of data, and any related entries in the default target list. Using a spreadsheet analogy, a *delete* operation can be portrayed as



where *c* is the column deleted.

Warnings

Deleting a variable used in the default target list removes that variable from the default target list. This makes the default target list less selective, qualifying more warts than intended. In general, a *delete* never constricts the target selection criteria.

Like all of the other commands which alter the active file, *delete* can be revoked via the *undo* command. However, while *undo* restores the active file back to its previous state it does not restore the default target list. Thus deleting a variable used in the default target list followed immediately by an *undo* still changes the default target list.

Removing Existing Warts

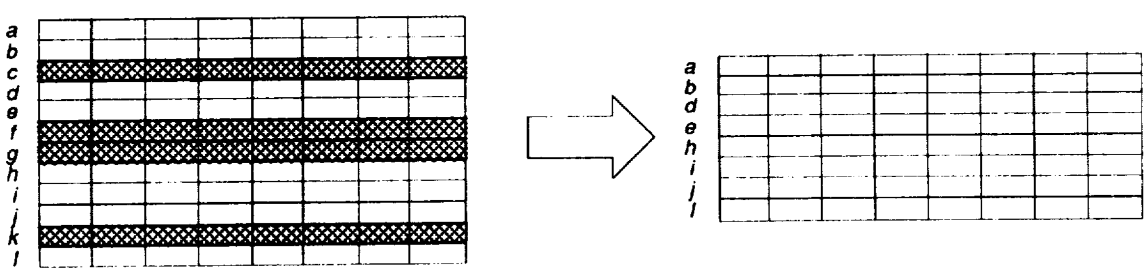
Command ***knockout*** removes selected data warts from the active TOAD file. Its form is

knockout [*object list*]

where the optional *object list* selects those data warts to be eliminated. If the object list is omitted, the current default target list is assumed. For example, the directive

knockout 2y/b .9

removes all data warts associated with a 90% (.9) span location (*2y/b*). Using a spreadsheet analogy, a *knockout* operation can be portrayed as



where the shaded rows represent data warts to be eliminated from the active file. Remember that *delete* removes columns of data -- *knockout* eliminates rows of data.

To further illustrate the *knockout* command, consider the following dialog:

```
edit> open  test
edit> scan

            This TOAD file contains 2 variables

                 eta                      temp

            . . . and has a total of 9 data warts

edit> tabulate

  wart #        eta              temp

    1        0.100000          1303.72
    2        0.200000          1285.43
    3        0.300000          1231.13
    4        0.400000          1142.45
    5        0.500000          1022.10
    6        0.600000          873.736
    7        0.700000          701.860
    8        0.800000          511.696
    9        0.900000          309.024

edit> knockout  3t6
edit> scan

            This TOAD file contains 2 variables

                 eta                      temp

            . . . and has a total of 5 data warts

edit> tabulate

  wart #        eta              temp

    1        0.100000          1303.72
    2        0.200000          1285.43
    3        0.700000          701.860
    4        0.800000          511.696
    5        0.900000          309.024

edit>
```

If the default target list is used in a *knockout* directive, subsequent directives which also use the default target list may trigger the message

```
No qualifying data
```

For example,

```
edit> open  test
edit> scan

        This TOAD file contains 2 variables

              eta                    temp

        . . . and has a total of 9 data warts

edit> tabulate

  wart #        eta            temp

    1         0.100000        1303.72
    2         0.200000        1285.43
    3         0.300000        1231.13
    4         0.400000        1142.45
    5         0.500000        1022.10
    6         0.600000        873.736
    7         0.700000        701.860
    8         0.800000        511.696
    9         0.900000        309.024

edit> target  eta  .3  .6
edit> knockout
edit> scan

        This TOAD file contains 2 variables

              eta                    temp

        . . . and has a total of 5 data warts

edit> tabulate

        No qualifying data

edit> tabulate  all

  wart #        eta            temp

    1         0.100000        1303.72
    2         0.200000        1285.43
    3         0.700000        701.860
    4         0.800000        511.696
    5         0.900000        309.024

edit>
```

Why? By definition, all data meeting this criteria were eliminated by the *knockout* directive, usually leaving none for subsequent operations. The solution is to either provide a direct object list or redefine the default target list.

The *knockout* command removes those rows of data identified in the object list. This can sometimes lead to the deletion of more data than intended. For example, again using our example file *test*, consider the following dialog:

```
edit> open  test
edit> scan

        This TOAD file contains 2 variables

             eta                    temp

        . . . and has a total of 9 data warts

edit> tabulate

  wart #        eta             temp

    1        0.100000         1303.72
    2        0.200000         1285.43
    3        0.300000         1231.13
    4        0.400000         1142.45
    5        0.500000         1022.10
    6        0.600000         873.736
    7        0.700000         701.860
    8        0.800000         511.696
    9        0.900000         309.024

edit> knockout  temp

        The entire file cannot be KO'd.

edit> knockout  temp  *  1000
edit> tabulate

  wart #        eta             temp

    1        0.100000         1303.72
    2        0.200000         1285.43
    3        0.300000         1231.13
    4        0.400000         1142.45
    5        0.500000         1022.10
```

The first *knockout* directive

knockout temp

qualifies the entire active file, since <u>all</u> values of variable *temp* are selected. This request is essentially the same as the directive

knockout all

which would normally erase the entire contents of the active file. To avoid potentially catastrophic results, this request is denied. We then enter

knockout temp * 1000

eliminating all data warts containing temperatures less than or equal to 1000, our original intention.

Wart Editing

Sometimes the easiest way to edit a file is to work directly with the data warts. For example, "squaring out" data often requires that data warts be inserted at specific locations. Likewise, it may be convenient to use an aircraft's starboard wing's pressure data to create the port wing's pressure distribution. All commands which allow you to directly manipulate entire data warts are called "wart editing" commands. Five such commands are currently available: ***addwart***, ***dupwart***, ***copywart***, ***cutwart***, and ***pastewart*** (Macintosh and Sun workstation users may already be familiar with the concept of copy, cut, and paste operations). Each is presented individually.

Adding Zero-Filled Warts

Command ***addwart*** expands the active file by creating empty (zero-filled) data warts one or more times. Its form is

addwart *wart_id* [*n*]

where *wart_id* identifies a location within the active file after which the new wart is inserted and *n* is an integer counter. The *wart_id* must be a valid wart id (a positive integer less than or equal to the number of current warts) or one of the keywords *top, first, bottom*, or *last*. Normally, the new warts are inserted immediately after the specified wart. However, when the keyword *top* or *first* is used the new warts are inserted before the first wart. The counter, *n*, must be a positive integer. If omitted, one new wart is assumed. For example, the directive

addwart 5 2

creates and inserts two new zero-filled warts following wart 5. Graphically portrayed, the operation looks like

1	*a1*	*b1*	*c1*	*d1*
2	*a2*	*b2*	*c2*	*d2*
3	*a3*	*b3*	*c3*	*d3*
4	*a4*	*b4*	*c4*	*d4*
5	*a5*	*b5*	*c5*	*d5*
6	*a6*	*b6*	*c6*	*d6*
7	*a7*	*b7*	*c7*	*d7*
8	*a8*	*b8*	*c8*	*d8*

addwart 5 2

1	*a1*	*b1*	*c1*	*d1*
2	*a2*	*b2*	*c2*	*d2*
3	*a3*	*b3*	*c3*	*d3*
4	*a4*	*b4*	*c4*	*d4*
5	*a5*	*b5*	*c5*	*d5*
6	*0*	*0*	*0*	*0*
7	*0*	*0*	*0*	*0*
8	*a6*	*b6*	*c6*	*d6*
9	*a7*	*b7*	*c7*	*d7*
10	*a8*	*b8*	*c8*	*d8*

where the vertical bar marks the new zero-filled warts added to the active file.

Because *addwart* increases the overall size of the active TOAD file, it's possible to exceed the raw data capacity of the Editor. If the request would cause the capacity to be exceeded, the Editor writes the message

```
Unable to add n new warts - Insufficient capacity.

Only n additional warts can be accommodated.
```

and does not perform the *addwart* operation. If the current capacity prevents you from effectively using the Editor, we suggest you follow the instructions presented in Section 6, "In Case of Problems."

Duplicating Existing Warts

Similar to *addwart*, command ***dupwart*** expands the active file by duplicating existing warts one or more times. Its form is

dupwart *wart_id* [*n*]

where *wart_id* identifies which wart is duplicated and *n* is an integer counter. All duplicate warts are inserted immediately after the original. The *wart_id* must be a valid wart id (a positive integer less than or equal to the number of current warts) or one of the keywords *top*, *first*, *bottom*, or *last*. Normally, the new warts are inserted immediately after the specified wart. However, when the keyword *top* or *first* is used the new warts are inserted before the first wart. The counter, *n*, must be a positive integer. If omitted, one duplicate wart is assumed. For example, the directive

dupwart 4

duplicates wart 4 and inserts it immediately after wart 4. A graphic portrayal of this operation is

1	*a1*	*b1*	*c1*	*d1*
2	*a2*	*b2*	*c2*	*d2*
3	*a3*	*b3*	*c3*	*d3*
4	*a4*	*b4*	*c4*	*d4*
5	*a5*	*b5*	*c5*	*d5*
6	*a6*	*b6*	*c6*	*d6*
7	*a7*	*b7*	*c7*	*d7*
8	*a8*	*b8*	*c8*	*d8*

dupwart 4

1	*a1*	*b1*	*c1*	*d1*	
2	*a2*	*b2*	*c2*	*d2*	
3	*a3*	*b3*	*c3*	*d3*	
4	*a4*	*b4*	*c4*	*d4*	
5	*a4*	*b4*	*c4*	*d4*	|
6	*a5*	*b5*	*c5*	*d5*	
7	*a6*	*b6*	*c6*	*d6*	
8	*a7*	*b7*	*c7*	*d7*	
9	*a8*	*b8*	*c8*	*d8*	

where the marked wart represents a duplicate of wart 4.

Because *dupwart* increases the overall size of the active TOAD file, it's possible to exceed the raw data capacity of the Editor. If the request would cause the capacity to be exceeded, the Editor writes the message

```
Unable to add n new warts - Insufficient capacity.

Only n additional warts can be accommodated.
```

and does not perform the *dupwart* operation. If the current capacity prevents you from effectively using the Editor, we suggest you follow the instructions presented in Section 6, "In Case of Problems."

Using the Wart Paste Buffer

More advanced wart editing commands are also available: ***copywart***, ***cutwart***, and ***pastewart***. The *copywart* and *cutwart* commands move one or more warts to the paste buffer for later use by the *pastewart* command. Command *copywart* moves the warts to the paste buffer but leaves the original active file undisturbed -- *cutwart* moves the warts to the paste buffer and removes them from the active file. Both have the form

command [*object list*]

where *command* is the command and the optional *object list* selects the warts to be copied or cut. If the object list is omitted, the default target list is assumed.

Once one or more warts are moved to the paste buffer they may be inserted back into the active file via the *pastewart* command. It has the form

pastewart *wart_id*

where *wart_id* identifies the insertion point within the active file. The *wart_id* must be a valid wart id (a positive integer less than or equal to the number of current warts) or one of the keywords *top*, *first*, *bottom*, or *last*. Normally, the warts contained on the paste buffer are inserted immediately after the specified wart. However, when the keyword *top* or *first* is used the buffered warts are inserted before the first wart.

To illustrate how *copywart* and *pastewart* are used together, consider the following directive sequence:

copywart 3t5

1	a1	b1	c1	d1
2	a2	b2	c2	d2
3	a3	b3	c3	d3
4	a4	b4	c4	d4
5	a5	b5	c5	d5
6	a6	b6	c6	d6
7	a7	b7	c7	d7
8	a8	b8	c8	d8

a3	b3	c3	d3
a4	b4	c4	d4
a5	b5	c5	d5

1	a1	b1	c1	d1
2	a2	b2	c2	d2
3	a3	b3	c3	d3
4	a4	b4	c4	d4
5	a5	b5	c5	d5
6	a6	b6	c6	d6
7	a7	b7	c7	d7
8	a8	b8	c8	d8

Note that the targeted warts are retained in the active file.

pastewart 7

1	a1	b1	c1	d1
2	a2	b2	c2	d2
3	a3	b3	c3	d3
4	a4	b4	c4	d4
5	a5	b5	c5	d5
6	a6	b6	c6	d6
7	a7	b7	c7	d7
8	a3	b3	c3	d3
9	a4	b4	c4	d4
10	a5	b5	c5	d5
11	a8	b8	c8	d8

a3	b3	c3	d3
a4	b4	c4	d4
a5	b5	c5	d5

Warts 3-5 are moved from the active file to the paste buffer, then pasted back following wart 7. Vertical bars mark the affected portions of the file during each operation. Notice how *copywart* left the active file undisturbed.

The *cutwart* and *pastewart* commands are used in a similar manner. As an illustration, consider the following directive sequence:

cutwart 4t7

1	a1	b1	c1	d1
2	a2	b2	c2	d2
3	a3	b3	c3	d3
4	a4	b4	c4	d4
5	a5	b5	c5	d5
6	a6	b6	c6	d6
7	a7	b7	c7	d7
8	a8	b8	c8	d8

a4	b4	c4	d4
a5	b5	c5	d5
a6	b6	c6	d6
a7	b7	c7	d7

1	a1	b1	c1	d1
2	a2	b2	c2	d2
3	a3	b3	c3	d3
4	a8	b8	c8	d8

Note that the targeted warts are removed from the active file.

pastewart top

1	a4	b4	c4	d4
2	a5	b5	c5	d5
3	a6	b6	c6	d6
4	a7	b7	c7	d7
5	a1	b1	c1	d1
6	a2	b2	c2	d2
7	a3	b3	c3	d3
8	a8	b8	c8	d8

a4	b4	c4	d4
a5	b5	c5	d5
a6	b6	c6	d6
a7	b7	c7	d7

Warts 4-7 are moved from the active file to the paste buffer, then pasted back at the top of the file. Vertical bars mark the affected portions of the file during each operation. Notice that *cutwart* removed the targeted warts from the active file.

Because *pastewart* increases the overall size of the active TOAD file, it's possible to exceed the raw data capacity of the Editor. If the request would cause the capacity to be exceeded, the Editor writes the message

```
Unable to add n new warts - Insufficient capacity.

Only n additional warts can be accommodated.
```

and does not perform the *pastewart* operation. If the current capacity prevents you from effectively using the Editor, we suggest you follow the instructions presented in Section 6, "In Case of Problems."

The paste buffer is maintained by the Editor. Although you can't directly edit its contents you can enter

show buffer

or

show paste

which tells you how many warts and how many columns of data the paste buffer contains.

Finally, the contents of the paste buffer are retained after a *pastewart* operation. Thus, you can move some warts into the paste buffer and then repeatedly insert them using multiple *pastewart* operations. In fact, the paste buffer is retained until you use another *copywart* or *cutwart*, replacing its contents. The paste buffer is not, however, retained between editing sessions. If you attempt to use an empty paste buffer (e.g., using a *pastewart* before a *copywart* or *cutwart*), the Editor writes the message

```
The paste buffer is empty.
```

Note

The wart editing commands move entire warts of data, not wart subsets. If you wish to move only a few columns of data, consider using *export* and *import*, discussed in the next subsection.

Warning

It is important to realize that the Editor moves warts as instructed, and that you are responsible for judging the validity of any wart editing operation. Careless use of the wart editing commands may create a meaningless or misleading TOAD file.

Using External Files

It is often necessary to have access to external files from within an ongoing editing session. For example, merging two or more TOAD files together requires that the Editor be able to read at least one external file. Selectively extracting data requires an ability to rewrite existing or create new external files.

The Editor offers six commands for exchanging data between the active TOAD file and external files. Commands ***write*** and ***read*** are designed to access and create, respectively, single-column TOAD files. Commands ***export*** and ***import*** perform similar functions for multi-column TOAD files. Commands ***before*** and ***after*** insert blocks of data warts at selected locations. All are individually presented.

Single Column

Command ***write*** extracts data from a selected column and places it into an external, single-column TOAD file. Its form is

write *variable* [*file*] [*object list*]

where *variable* identifies the column from which to extract the data, *file* is the optional name of the external file to be written, and the optional *object list* selects which data warts are used. The variable provided must already exist. If the external file name is omitted, file *tadpole* is assumed. If the external file does not exist, it is created. If the file does exist, the message

```
This request will overwrite the original contents of
an existing file.  Do you really want it performed ?
```

may appear, depending upon the state of the OverWrite protection toggle. Answering "yes" instructs the Editor to overwrite the file. Entering "no" instructs it to ignore the previous *write* directive. If the object list is omitted, the current default target list is assumed. For example, the directive

write temp nose eta .1 .5

extracts all temperature data (*temp*) for *eta* .1 through .5 and writes them to external file *nose*. The directive

write deltacp

extracts all of the available pressure data (assuming the default target list is set to *all*) and writes it into file *tadpole*.

The *write* command is normally used to create temporary files which are later accessed with the *read* command. A full discussion of how the two work together is presented within the description for *read*, presented next.

Command ***read*** places data from an external, single-column TOAD file in a selected column. Its form is

read *variable* [*file*] [*object list*]

where *variable* identifies the column to receive the data, *file* is the optional name of the external file to be read, and the optional *object list* selects those data warts to receive the data. The variable name provided must already exist -- the Editor does not create it. If the external file name is omitted, file *tadpole* is assumed. If the object list is omitted, the current default target list is assumed. For example, the directive

read temp hotcase eta .2 .7

reads TOAD file *hotcase* and places its contents in variable *temp* as the values of *eta* fall within the interval [.2,.7]. The directive

read press nose

reads the TOAD file *nose* and places its contents into variable *press*, as controlled by the default target list.

Note

> The *read* command replaces values within existing data cells -- it does not increase the size of the active TOAD file. If you want to increase the file's size, use *create*, *before*, *after*, *addwart*, *dupwart*, or *pastewart*.

Only single-column TOAD files are accepted. If the file provided contains more than one variable, the file is rejected and an error message is written. Further, the number of values available from the external file and the number of data cells to be filled must match exactly. If the external file contains either too little or too much data for the number of targeted data cells, an error message is written and no data is transferred.

In practice, few single-column TOAD files exist outside of those created using the *write* command. The *read* and *write* commands are often used together to transfer blocks of data between two different TOAD files. As an illustration, suppose we monitored an experiment in which reentry vehicle skin temperature (*temp*) and pressure (*press*) data were collected as a function of nondimensional body station (*eta*). Unfortunately, the temperature data and the pressure data, although measured at the same locations, were stored in two different TOAD files. The following dialog demonstrates how the data can be merged into one file:

```
edit> open  hot_press
edit> tabulate

 wart #       eta            press

   1       0.100000       3.080525
   2       0.200000       3.033725
   3       0.300000       2.894747
   4       0.400000       2.667813
   5       0.500000       2.359819
   6       0.600000       1.980124
   7       0.700000       1.540263
   8       0.800000       1.053602
   9       0.900000      0.5349276

edit> write  press  hold1
edit> open  hot_temp
edit> tabulate

 wart #       eta            temp

   1       0.100000       1303.72
   2       0.200000       1285.43
   3       0.300000       1231.13
   4       0.400000       1142.45
```

```
    5        0.500000        1022.10
    6        0.600000        873.736
    7        0.700000        701.860
    8        0.800000        511.696
    9        0.900000        309.024

edit> save  hot_both
edit> open  hot_both
edit> scan

        This TOAD file contains 2 variables

             eta                     temp

        . . . and has a total of 9 data warts

edit> create  press
edit> menu

        This TOAD file contains 3 variables

             eta                     temp
             press

edit> tabulate

   wart #       eta           temp           press

     1        0.100000       1303.72          0.
     2        0.200000       1285.43          0.
     3        0.300000       1231.13          0.
     4        0.400000       1142.45          0.
     5        0.500000       1022.10          0.
     6        0.600000       873.736          0.
     7        0.700000       701.860          0.
     8        0.800000       511.696          0.
     9        0.900000       309.024          0.

edit> read  press  hold1
edit> tabulate

   wart #       eta           temp           press

     1        0.100000       1303.72        3.080525
     2        0.200000       1285.43        3.033725
     3        0.300000       1231.13        2.894747
     4        0.400000       1142.45        2.667813
     5        0.500000       1022.10        2.359819
     6        0.600000       873.736        1.980124
     7        0.700000       701.860        1.540263
     8        0.800000       511.696        1.053602
     9        0.900000       309.024        0.5349276
```

```
edit>
```

Warning

It is important to realize that the Editor merges data as instructed, and that you are responsible for judging the validity of any merge operation. In this example both temperature and pressure are measured at the same values for *eta*, and in the same order. This ensures that the resulting file correctly matches *eta*, *temp*, and *press*. Had the separate temperature and pressure files not been compatible, the same merge operation would create a meaningless or misleading file.

Multiple Columns

Command ***export*** extracts a selected data subset and places it into an external, single- or multi-column TOAD file. Its form is

export [*file*] [*object list*]

where *file* is the optional name of the external file to be written and the optional *object list* selects the desired data subset. If the external file name is omitted, file *tadpole* is assumed. If the external file does not exist, it is created. If the file does exist, the message

```
This request will overwrite the original contents of
an existing file.  Do you really want it performed ?
```

may appear, depending upon the state of the OverWrite protection toggle. Answering "yes" instructs the Editor to overwrite the file. Entering "no" instructs it to ignore the previous *export* directive. If the object list is omitted, the current default target list is assumed. For example, the directive

export nose eta .1 .5 temp press

writes a TOAD file called *nose* containing three variables (*eta*, *temp*, and *press*) using data from those warts in which the value of *eta* falls within the interval [.1,.5].

To illustrate how *export* may be used to extract data, consider the following dialog:

```
edit> open  toad1
edit> tabulate  2y/b  .9  x/c  deltacp

  wart #        x/c           deltacp

   133      0.416667E-01     6.09007
   134      0.208333         3.02826
   135      0.375000         2.12340
   136      0.541667         1.60278
   137      0.708333         1.17190
   138      0.875000        0.711813

edit> export  tip_cp  2y/b  .9  x/c  deltacp
edit> open  tip_cp
edit> tabulate
```

wart #	2y/b	x/c	deltacp
1	0.900000	0.416667E-01	6.09007
2	0.900000	0.208333	3.02826
3	0.900000	0.375000	2.12340
4	0.900000	0.541667	1.60278
5	0.900000	0.708333	1.17190
6	0.900000	0.875000	0.711813

```
edit> open toad1
edit> tabulate x/c .041666 2y/b deltacp
```

wart #	2y/b	deltacp
1	0.200000E-01	1.05367
7	0.600000E-01	1.16457
13	0.100000	1.29959
19	0.140000	1.43850
25	0.180000	1.57196
31	0.220000	1.70156
37	0.260000	1.82998
43	0.300000	1.95902
49	0.340000	2.09000
55	0.380000	2.22420
61	0.420000	2.36313
67	0.460000	2.50851
73	0.500000	2.66236
79	0.540000	2.82717
85	0.580000	3.00611
91	0.620000	3.20335
97	0.660000	3.42449
103	0.700000	3.67719
109	0.740000	3.97186
115	0.780000	4.32299
121	0.820000	4.75334
127	0.860000	5.30779
133	0.900000	6.09007
139	0.940000	7.38284
145	0.980000	10.5822

```
edit> export le_cp x/c .041666 2y/b deltacp
edit> open le_cp
edit> tabulate
```

wart #	x/c	2y/b	deltacp
1	0.416667E-01	0.200000E-01	1.05367
2	0.416667E-01	0.600000E-01	1.16457
3	0.416667E-01	0.100000	1.29959
4	0.416667E-01	0.140000	1.43850
5	0.416667E-01	0.180000	1.57196
6	0.416667E-01	0.220000	1.70156
7	0.416667E-01	0.260000	1.82998

8	0.416667E-01	0.300000	1.95902
9	0.416667E-01	0.340000	2.09000
10	0.416667E-01	0.380000	2.22420
11	0.416667E-01	0.420000	2.36313
12	0.416667E-01	0.460000	2.50851
13	0.416667E-01	0.500000	2.66236
14	0.416667E-01	0.540000	2.82717
15	0.416667E-01	0.580000	3.00611
16	0.416667E-01	0.620000	3.20335
17	0.416667E-01	0.660000	3.42449
18	0.416667E-01	0.700000	3.67719
19	0.416667E-01	0.740000	3.97186
20	0.416667E-01	0.780000	4.32299
21	0.416667E-01	0.820000	4.75334
22	0.416667E-01	0.860000	5.30779
23	0.416667E-01	0.900000	6.09007
24	0.416667E-01	0.940000	7.38284
25	0.416667E-01	0.980000	10.5822

```
edit>
```

The first *export* directive creates the file *tip_cp* containing pressure (*deltacp*) as a function of chord location (*x/c*) at a 90% wing semispan (*2y/b* =.9), which is near the wing's outboard tip. The second *export* directive creates another file, *le_cp*, containing pressure (*deltacp*) as a function of wing semispan (*2y/b*) at a 1/24th chord location (*x/c* =.0416667), which is along the wing's leading edge.

These examples are simplified to illustrate how *export* is used. In practice, actual TOAD files and target object lists tend to have more controlling independent variables and are much more complex. By no means should you feel constrained in the use of *export*. If you can identify the data subset using an object list, the Editor can extract the data and write the corresponding TOAD file.

Command ***Import*** replaces a selected data subset with data contained on an external, single- or multi-column TOAD file. Its form is

Import [*file*] [*object list*]

where *file* is the optional name of the external file to be read and the optional *object list* targets the receiving data cells. If the external file name is omitted, file *tadpole* is assumed. If the object list is omitted, the current default target list is assumed. For example, the directive

Import tip_cp 2y/b .98 x/c deltacp

reads a TOAD file called *tip_cp* and places the incoming values into data cells belonging to variables *2y/b*, *x/c*, and *deltacp* only when the original value for *2y/b* is 98% (.98).

Note

The *import* command replaces values within existing data cells -- it does not increase the size of the active TOAD file. If you want to increase the file's size, use *create*, *before*, *after*, *addwart*, *dupwart*, or *pastewart*.

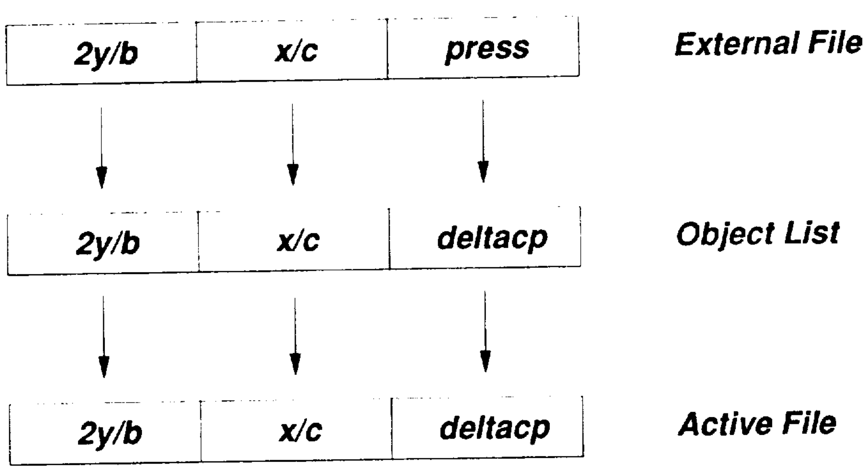
Single- or multi-column TOAD files are accepted. The number of variables contained on the external file must match the number of variables in the target list. If the file contains either too few or too many variables, the file is rejected and an error message is written. Further, the number of values available from the external file and the number of data cells to be filled must match exactly. If the external file contains either too little or too much data for the number of targeted data cells, the file is again rejected and an error message is written.

To illustrate a few of the *import* command's many variations, consider the following dialog:

```
edit> op  tip_cp
edit> scan

        This TOAD file contains 3 variables:

             2y/b                    x/c
             press

        . . . and has a total of 6 data warts

edit> tabulate

  wart #      2y/b          x/c              press

     1      0.980000      0.416667E-01     10.1589
     2      0.980000      0.208333          5.38980
     3      0.980000      0.375000          4.25331
     4      0.980000      0.541667          3.63584
     5      0.980000      0.708333          3.02780
     6      0.980000      0.875000          2.04281

edit> op  toad1
edit> scan

        This TOAD file contains 6 variables:

             mach                    cldes                   planform
             x/c                     2y/b                    deltacp

        . . . and has a total of 150 data warts

edit> target 2y/b  .94  *  x/c  deltacp
edit> tabulate

  wart #      2y/b          x/c              deltacp

   139      0.940000      0.416667E-01      7.38284
   140      0.940000      0.208333          3.76906
   141      0.940000      0.375000          2.71414
   142      0.940000      0.541667          2.03536
   143      0.940000      0.708333          1.50435
   144      0.940000      0.875000          0.937932
   145      0.980000      0.416667E-01     10.5822
   146      0.980000      0.208333          5.61437
```

```
 147      0.980000        0.375000        4.43053
 148      0.980000        0.541667        3.78733
 149      0.980000        0.708333        3.15396
 150      0.980000        0.875000        2.12793

edit> import tip_cp 2y/b .98 x/c deltacp
edit> tabulate

 wart #      2y/b            x/c             deltacp

 139      0.940000        0.416667E-01    7.38284
 140      0.940000        0.208333        3.76906
 141      0.940000        0.375000        2.71414
 142      0.940000        0.541667        2.03536
 143      0.940000        0.708333        1.50435
 144      0.940000        0.875000        0.937932
 145      0.980000        0.416667E-01    10.1589
 146      0.980000        0.208333        5.38980
 147      0.980000        0.375000        4.25331
 148      0.980000        0.541667        3.63584
 149      0.980000        0.708333        3.02780
 150      0.980000        0.875000        2.04281

edit>
```

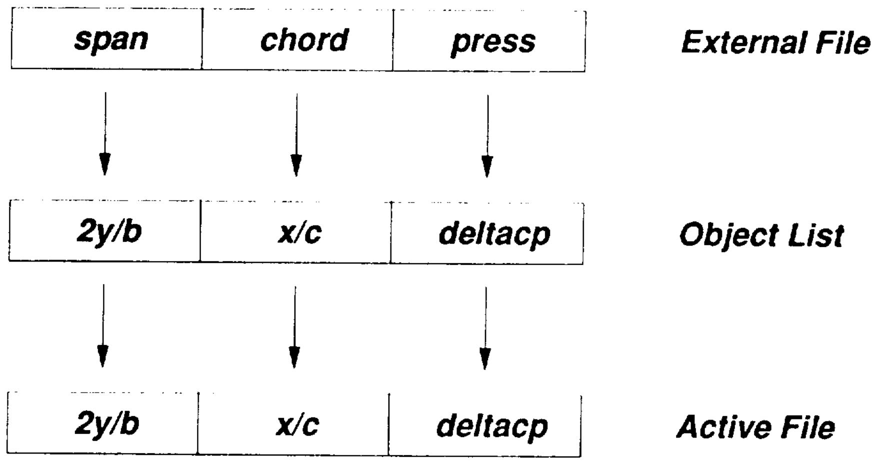
Notice that external file *tip_cp* contains three variables and six data warts, exactly the size of the *import* directive's object list. Also notice the values for *2y/b* and *x/c* within the external file *tip_cp* exactly match those within the active file *toad1*. Finally, notice the values associated with variable *press* from the external file are used to replace values for *deltacp*. There is nothing special about the variable names *press* or *deltacp*. Rather, the Editor merely substituted values from the external file's third variable for those values associated with the active file's third variable in the object list. This can be portrayed graphically as



It doesn't really matter what the external file's variable names are, as long as the number of variables it contains matches the number of variables specified in the object list. For example, the external file

wart #	span	chord	press
1	0.980000	0.416667E-01	10.1589
2	0.980000	0.208333	5.38980
3	0.980000	0.375000	4.25331
4	0.980000	0.541667	3.63584
5	0.980000	0.708333	3.02780
6	0.980000	0.875000	2.04281

would still be imported as



Suppose another external file contains the same data, only arranged differently:

wart #	press	chord	span
1	10.1589	(.416667E-01	0.980000
2	5.38980	0.208333	0.980000
3	4.25331	0.375000	0.980000
4	3.63584	0.541667	0.980000
5	3.02780	0.708333	0.980000
6	2.04281	0.875000	0.980000

Can we still import its data?

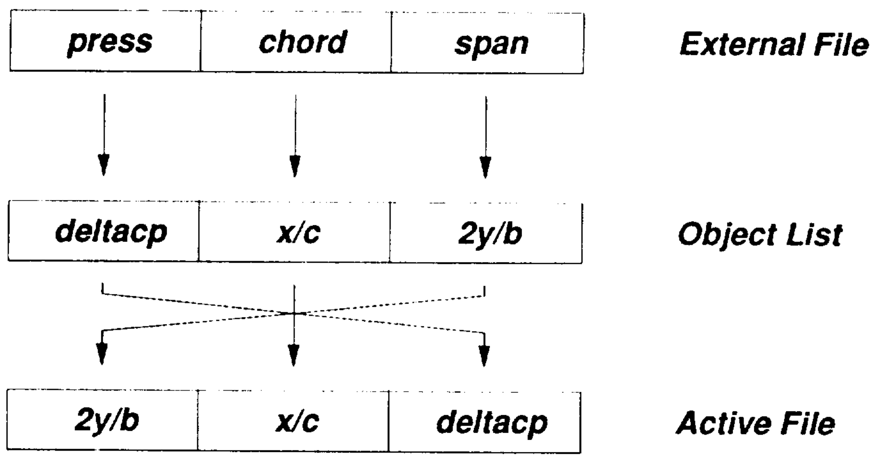
The answer lies within the concept of targeting and object lists. We can't change how the external file is structured, or how it is read, but we can control the receiving pattern of the active file's targeted data cells. For example, consider the following dialog:

```
edit> op  tip_press
edit> scan

          This TOAD file contains 3 variables:

               press               chord
               span
```

```
         . . . and has a total of 6 data warts

edit> tabulate

  wart #      press         chord          span

    1        10.1589      0.416667E-01   0.980000
    2        5.38980      0.208333       0.980000
    3        4.25331      0.375000       0.980000
    4        3.63584      0.541667       0.980000
    5        3.02780      0.708333       0.980000
    6        2.04281      0.875000       0.980000

edit> op  toad1
edit> scan

         This TOAD file contains 6 variables:

              mach               cldes              planform
              x/c                2y/b               deltacp

         . . . and has a total of 150 data warts

edit> targ  2y/b  .94  *  x/c  deltacp
edit> tabulate

  wart #      2y/b           x/c           deltacp

   139      0.940000      0.416667E-01    7.38284
   140      0.940000      0.208333        3.76906
   141      0.940000      0.375000        2.71414
   142      0.940000      0.541667        2.03536
   143      0.940000      0.708333        1.50435
   144      0.940000      0.875000       0.937932
   145      0.980000      0.416667E-01    10.5822
   146      0.980000      0.208333        5.61437
   147      0.980000      0.375000        4.43053
   148      0.980000      0.541667        3.78733
   149      0.980000      0.708333        3.15396
   150      0.980000      0.875000        2.12793

edit> import  tip_press  deltacp  x/c  2y/b  .98
edit> tabulate

  wart #      2y/b           x/c           deltacp

   139      0.940000      0.416667E-01    7.38284
   140      0.940000      0.208333        3.76906
   141      0.940000      0.375000        2.71414
   142      0.940000      0.541667        2.03536
   143      0.940000      0.708333        1.50435
   144      0.940000      0.875000       0.937932
   145      0.980000      0.416667E-01    10.1589
```

```
     146        0.980000        0.208333        5.38980
     147        0.980000        0.375000        4.25331
     148        0.980000        0.541667        3.63584
     149        0.980000        0.708333        3.02780
     150        0.980000        0.875000        2.04281

edit>
```

Notice that only the *import* directive's object list was altered to accommodate what initially appeared to be an incompatible external file structure. This last *import* operation can be graphically portrayed as



Therefore, the key to using the *import* directive's object list is to match variable positions between the external and active file. The variable names within the *import* directive's object list serve only to identify which columns receive the incoming data.

This flexibility can easily be misused. For example, consider the following dialog:

```
edit> op  tip_press
edit> scan

        This TOAD file contains 3 variables:

             press                    chord
             span

        . . . and has a total of 6 data warts

edit> tabulate

  wart #       press           chord              span

     1         10.1589         0.416667E-01       0.980000
     2         5.38980         0.208333           0.980000
     3         4.25331         0.375000           0.980000
     4         3.63584         0.541667           0.980000
     5         3.02780         0.708333           0.980000
```

```
    6         2.04281         0.875000         0.980000

edit> op  toad1
edit> scan

        This TOAD file contains 6 variables:

            mach                    cldes                  planform
            x/c                     2y/b                   deltacp

        . . . and has a total of 150 data warts

edit> targ 2y/b  .94  *  x/c  deltacp
edit> tabulate

  wart #       2y/b            x/c             deltacp

   139       0.940000       0.416667E-01      7.38284
   140       0.940000       0.208333          3.76906
   141       0.940000       0.375000          2.71414
   142       0.940000       0.541667          2.03536
   143       0.940000       0.708333          1.50435
   144       0.940000       0.875000         0.937932
   145       0.980000       0.416667E-01      10.5822
   146       0.980000       0.208333          5.61437
   147       0.980000       0.375000          4.43053
   148       0.980000       0.541667          3.78733
   149       0.980000       0.708333          3.15396
   150       0.980000       0.875000          2.12793

edit> import  tip_press  2y/b  .98  x/c  deltacp
edit> tabulate

  wart #       2y/b            x/c             deltacp

   139       0.940000       0.416667E-01      7.38284
   140       0.940000       0.208333          3.76906
   141       0.940000       0.375000          2.71414
   142       0.940000       0.541667          2.03536
   143       0.940000       0.708333          1.50435
   144       0.940000       0.875000         0.937932
   145        10.1589       0.416667E-01     0.980000
   146        5.38980       0.208333         0.980000
   147        4.25331       0.375000         0.980000
   148        3.63584       0.541667         0.980000
   149        3.02780       0.708333         0.980000
   150        2.04281       0.875000         0.980000

edit>
```

Because the *import* directive's object list doesn't properly align the external file's data with the active file's columns, the external file's span data (*span*) and pressure data (*press*) end up in the active file's columns for coefficient of pressure (*deltacp*) and span location (*2y/b*). Such an operation is a clear misuse of *import*.

Note

The *export* and *import* commands are designed for manipulating scattered warts or wart subsets. If you are manipulating contiguous blocks of entire warts, *copywart*, *cutwart*, and *pastewart* may be easier to use.

Warning

It is your responsibility to ensure that the incoming data from the external file is appropriate for the targeted cells within the internal active file. Improper use of *import* can create worthless or misleading TOAD files.

General File Insertion

Commands ***before*** and ***after*** insert the contents of a single- or multi-column TOAD before or after, respectively, the specified data wart. Their forms are

before *wart_id* [*file*]

and

after *wart_id* [*file*]

where *wart_id* identifies where the new data is inserted and *file* is the optional name of the external file to be read. If the external file name is omitted, file *tadpole* is assumed. For example, the directive

before 32 extra

inserts the contents of file *extra* between existing data warts 31 and 32. Similarly, the directive

after 32 extra

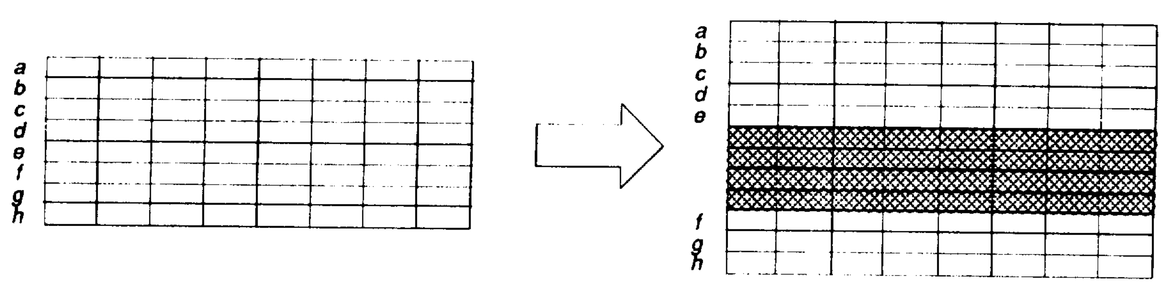
inserts the contents of file *extra* between existing data warts 32 and 33. In addition to a numeric wart id, the keywords **top**, **first**, **bottom**, and **last** may also be used, as in the directives

before top extra or **before first extra**

after bottom extra or **after last extra**

which insert the contents of file *extra* before the first data wart or after the last, respectively.

Using a spreadsheet analogy, a *before* or *after* operation can be portrayed as



where the shaded rows represent those data warts added to the active file.

Single- or multi-column TOAD files are accepted. The number of variables contained on the external file must match the number of variables in the active file. If the file contains either too few or too many variables, the file is rejected and an error message is written.

Because adding more data warts increases the overall size of the active TOAD file, it's possible to exceed the raw data capacity (number of data cells) of the Editor. If the request would cause the capacity to be exceeded, the Editor writes the message

```
Unable to use this file - Insufficient capacity.

Only n additional raw data can be accommodated.
```

and does not perform the *before* or *after* request. If the current capacity prevents you from effectively using the Editor, we suggest you follow the instructions presented in Section 6, "In Case of Problems."

Warning

It is your responsibility to ensure that the incoming data from the external file is appropriate for the targeted cells within the internal active file. For example, pressure data should not be brought in and subsequently treated as temperature data. Improper use of *before* or *after* can create worthless or misleading TOAD files.

Directive Files and Macros

Directive Files

You may prefer to have the Editor read long or repetitive directive sequences from an external file, rather than entering them interactively. Such a file is called a directive file and is executed via the ***Include*** command:

Include *file*

where *file* is the name of the directive file to be executed. For example, suppose we have the disc file *group1* which contains the following directives:

```
open toad1
target deltacp x/c 2y/b [.94,.98] alpha [0,30]
tabulate
```

It is invoked by entering

```
include group1
```

whereupon the directives

```
open toad1
target deltacp x/c 2y/b [.94,.98] alpha [0,30]
tabulate
```

are read and executed.

Under normal circumstances the editing session is controlled by your keyboard entries. However, when a directive file is invoked it assumes control and returns it only after all of the directives within the file have been processed. Very long or very complex directive files take a commensurate amount of time to process, which may create a noticeable delay.

Helpful Hint

The EntryEcho toggle is particularly useful when using directive files. Under normal circumstances, the Editor displays little if any progress information after you've started executing the contents of a directive file. If, at the beginning of the directive file, you enable the EntryEcho toggle, each directive is displayed as it is executed, providing a live report of the Editor's progress. We highly recommend this practice and offer the following as a pattern for all of your directive files:

Enable entryecho
directive
directive
.
.

```
directive
Disable entryecho
```

More than one directive file may be executed during a single editing session. For example, imagine we have two directive files: *findq* and *findcp*. File *findq* contains the directives

```
create dynamicp
create spare
power freev 2 spare
mult spare rho
divide spare 2 dynamicp
delete spare
```

and file *findcp* contains the directives

```
create cp
create spare
subtract press staticp spare
divide spare dynamicp cp
delete spare
```

Entering the directives

```
include findq
include findcp
```

executes the directive sequence

```
create dynamicp
create spare
power freev 2 spare
mult spare rho
divide spare 2 dynamicp
delete spare
create cp
create spare
subtract press staticp spare
divide spare dynamicp cp
delete spare
```

which creates dynamic pressure and pressure coefficient tables.

A directive file itself may call another directive file. For example, consider the directive file *set200*:

```
open run203
include findq
include findcp
save
open run204
include findq
include findcp
save
```

```
open run205
include findq
include findcp
save
```

which makes three calls to the directive files *findq* and *findcp*, already defined. Entering

```
include set200
```

creates a directive sequence which opens three files, performing calculations for the dynamic pressure and pressure coefficient tables in each. There is no limit on the number of levels within such a directive file hierarchy, nor is there a limit on the number of directive files which may be called within the same level. Repetitive calls to a single directive file, as illustrated above, are allowed. However, a directive file cannot call itself; that is, directive file recursion is not allowed.

Macros

A macro is a sequence of directives which, taken collectively, is executed by name. For example, suppose you have a series of wind tunnel results files in which the model's angle of attack must be converted from degrees to radians and the temperature readings must be converted from degrees Rankine to degrees Kelvin. You could consolidate the necessary directives as a macro called *fix* and then have them executed by simply entering

```
fix
```

Why use a macro when you could use a directive file? There are two reasons. First, macros are generally more convenient than directive files simply because fewer keystrokes are required. Entering

```
fix
```

is easier and more convenient than entering

include *file*

Second, macros have the ability to pass and use arguments. This allows the macro to customize its directives according to the information you pass it. Unlike a directive file, which always executes the same set of directives, a macro execution may be "adjusted" via passed arguments. A full description of macro arguments will be presented on the next page.

Creating and Executing Macros

Macros are created using the ***macro*** and ***endmacro*** commands, as illustrated below:

```
edit> macro  fix
macro> convert  alpha  degrees2radians
macro> convert  temp  rankine2kelvin
macro> endmacro
edit>
```

This dialog creates the macro *fix*, which converts all angle of attack values (*alpha*) from degrees into radians (*degrees2radians*) and converts all temperatures (*temp*) from the Rankine scale to the Kelvin scale (*rankine2kelvin*). Notice that after the *macro* directive the prompt becomes `macro>` and that after *endmacro* it changes back to the original `edit>` prompt. All directives entered after a *macro* directive and before an *endmacro* directive are considered to be that macro's "script." You must complete the definition of a macro before beginning to define another. That is, you are not permitted to begin another macro definition at the `macro>` prompt.

Once macro *fix* is defined, the two conversion directives

```
convert alpha degrees2radians
convert temp rankine2kelvin
```

can be executed by simply entering

```
fix
```

Warning

Macro names must not match Editor commands, or their aliases, unless you intend to replace that command with one of your own. Also, macro names which match active variable or symbol names have the potential for creating severe problems.

Macros may also be created to accept arguments. For example, the macro definition

```
macro halfsquare $a
divide $a 2
power $a 2
endmacro
```

creates the macro *halfsquare*, which divides a variable in half and then finds its square. This definition also declares one variable, *a*, which appears as *$a*. The dollar sign prefix (called the "macro character") marks all occurrences of the argument *a*. Use of the macro character is optional for declaring a macro variable but is mandatory when marking the variable within the macro script. Thus, the same macro definition could also be written as

```
macro halfsquare a
divide $a 2
power $a 2
endmacro
```

where the macro character is omitted from variable *a* only when it is declared in the *macro* directive.

To repeat, use of the macro character in marking macro variables within the macro script is mandatory. To illustrate the significance of using the macro character, consider the macro definition

```
macro halfsquare a
divide a 2
power a 2
endmacro
```

Variable *a* is correctly declared on the first line. However, because the macro character is not used to mark subsequent appearances, the parameter *a* in the *divide* and *power* directives is assumed to be a variable name within the active TOAD file, not occurrences of the macro variable *a*.

Why use macro arguments? Declaring and using macro arguments provide versatility not found in directive files. For example, again using our example macro definition

```
macro halfsquare a
divide $a 2
power $a 2
endmacro
```

when we enter the directive

```
halfsquare alpha
```

the Editor substitutes and processes the directives

```
divide alpha 2
power alpha 2
```

Similarly, entering the directive

```
halfsquare 'yaw angle'
```

leads to the execution of the directives

```
divide 'yaw angle' 2
power 'yaw angle' 2
```

Thus the argument "mimics" whatever is entered in its position when the macro is invoked.

Multiple arguments are declared and used in a similar manner. For example, consider the macro definition

```
macro findrpm base delta final
create $final
add $base $delta $final
mult $final .918333
endmacro
```

which creates macro *findrpm* with three variables: *base*, *delta*, and *final*. Entering the directive

```
findrpm msid1021 msid1078 rpmloxpump
```

leads to the execution of the directives

```
create rpmloxpump
add msid1021 msid1078 rpmloxpump
mult rpmloxpump .918333
```

Macro arguments may be omitted only if a suitable default value is available. For example, the macro definition

```
macro findrpm base=msid1021 delta=msid1078 final=rpmloxpump
create $final
add $base $delta $final
```

```
mult $final .918333
endmacro
```

declares the same macro arguments as before, with the addition of default values. Now, if variable *base* is omitted, it takes on its default value, *msid1021*. Likewise, if macro variables *delta* or *final* are omitted, the values *msid1078* or *rpmloxpump* are assumed, respectively. Thus the directive

```
findrpm
```

executes the directives

```
create rpmloxpump
add msid1021 msid1078 rpmloxpump
mult rpmloxpump .918333
```

When values are provided they override any defaults. For example, entering

```
findrpm msid1022
```

executes the directives

```
create rpmloxpump
add msid1022 msid1078 rpmloxpump
mult rpmloxpump .918333
```

Entering

```
findrpm ,, msid1079
```

executes the directives

```
create rpmloxpump
add msid1021 msid1079 rpmloxpump
mult rpmloxpump .918333
```

And entering

```
findrpm msid1056 msid1097 rpmh2pump
```

executes the directives

```
create rpmh2pump
add msid1056 msid1097 rpmh2pump
mult rpmh2pump .918333
```

Macro arguments pass any type of information, including commands and keywords. For example, consider the macro definition

```
macro merge file1 column1 file2 column2 command column3 object_list
create $column1
create $column2
create $column3
read $column1 $file1 $object_list
```

```
read $column2 $file2 $object_list
$command $column1 $column2 $column3 $object_list
endmacro
```

Macro *merge* reads the contents of two external files into two new columns, then uses a mathematical function to calculate the contents of a third new column, all subject to an object list. Entering

```
merge loxdata loxmass h2data h2mass add propmass 'time 1500 2500'
```

executes the directives

```
create loxmass
create h2mass
create propmass
read loxmass loxdata 'time 1500 2500'
read h2mass h2data 'time 1500 2500'
add loxmass h2mass propmass 'time 1500 2500'
```

which presumably reads liquid oxygen and hydrogen mass tables from the external files *loxdata* and *h2data*, respectively, then sums the two columns to arrive at total propellant mass (*propmass*), all between the event times 1500 and 2500. Although it's an unusual example, this macro does show how file names, commands, and object lists can be transmitted to the macro via arguments. A more realistic example is the macro definition

```
macro yanklep file1 file2
open $file1
define leading_edge=0
min x/c leading_edge
export $file2 deltacp x/c leading_edge 2y/b
delsymbol leading_edge
close
endmacro
```

which can be used as

```
yanklep run203 lep203
yanklep run204 lep204
yanklep run205 lep205
yanklep run206 lep206
```

to isolate and extract leading edge pressure tables from a series of raw TOAD files.

Helpful Hints

Our experience with defining and using macros with arguments suggests that, when properly designed, a few macros can go a long way. We recommend that, until you become proficient in their use, you limit their number, size, and complexity.

Our experience also suggests that allowing omitted parameters is justified in only a few situations -- the most common being when working with a series of files which contain the same type of data with the same variable names. Using an improper default value for an omitted parameter may create severe problems which may go unnoticed. If you decide to allow omitted parameters, it is usually best to place them after the required parameters. For example, the macro definition

macro frame scene color=blue

may be invoked by

frame missile

whereas the definition

macro frame color=blue scene

leads to the directive

frame ,, missile

which adds confusion to an already difficult feature.

General Notes

There's no need for you to define all of your macros "live" from the keyboard. We suggest putting all macro definitions in the startup file. In fact, the startup file is intended to be the place to keep your macros and have them defined automatically before every editing session.

How macro definitions are arranged within the startup file is a matter of personal style. However, macro novices should be aware of two extreme schools: the "big bang" approach and the "fragment" approach. The big bang approach puts all of the macro definitions directly into the startup file (embedded comments are always helpful). This method centralizes all macro definitions but may complicate later editing. The fragment approach puts each macro in a separate file, each to be included as a directive file within the startup file. This method makes the macro definitions more modular but often grows into a large file set. Again, it's a matter of personal style, so there's no "right" or "wrong" way to use your startup file.

To display a list of all current macros, enter

show macros

which creates a report in the form

```
The defined macros are:
```

macro #1
macro #2
.
.
macro #n

To display the argument list and directive script associated with a particular macro, enter

show macro *name*

where *name* is the name of the macro to be displayed, which creates a report in the form

```
Macro:      macro name

Parameters:    parameter #1      = default value
               parameter #2      = default value
               .
               .
               parameter #n      = default value

Script:

     directive
     directive
     .
     .
     directive
```

Renaming and Deleting Macros

To rename an existing macro, use the directive

renmacro *old_name* *new_name*

where *old_name* is the name of the macro being renamed and *new_name* is its new name. Both parameters are required -- the Editor cannot make any assumptions if either or both are omitted. In addition, *old_name* must be an existing macro, and *new_name* cannot be an existing macro.

To delete an existing macro, use the directive

delmacro *name*

where *name* is the name of the macro to be deleted.

Undoing Macros

The *undomacro* command allows you to restore the active file back to what it was immediately before the most recent macro execution, whether that macro changed the file or not. As an illustration, consider the following dialog:

```
edit> macro z
macro> add col1 1000
macro> mult col2 -1
macro> endmacro
edit> open test1
edit> tab

     wart #      col1        col2        col3        col4

       1       101.000     102.000     103.000     104.000
```

```
   2        201.000      202.000      203.000      204.000
   3        301.000      302.000      303.000      304.000
   4        401.000      402.000      403.000      404.000
   5        501.000      502.000      503.000      504.000
   6        601.000      602.000      603.000      604.000
   7        701.000      702.000      703.000      704.000
   8        801.000      802.000      803.000      804.000
   9        901.000      902.000      903.000      904.000

 9 wart subsets listed.

edit> z

 [ add col1 1000 ]

 9 data warts changed.

 [ mult col2 -1 ]

 9 data warts changed.

edit> tab

 wart #      col1         col2         col3         col4

   1       1101.00     -102.000      103.000      104.000
   2       1201.00     -202.000      203.000      204.000
   3       1301.00     -302.000      303.000      304.000
   4       1401.00     -402.000      403.000      404.000
   5       1501.00     -502.000      503.000      504.000
   6       1601.00     -602.000      603.000      604.000
   7       1701.00     -702.000      703.000      704.000
   8       1801.00     -802.000      803.000      804.000
   9       1901.00     -902.000      903.000      904.000

 9 wart subsets listed.

edit> undomacro

 The active file has reverted back to how it was
 before the last macro was executed.

edit> tab

 wart #      col1         col2         col3         col4

   1        101.000      102.000      103.000      104.000
   2        201.000      202.000      203.000      204.000
   3        301.000      302.000      303.000      304.000
   4        401.000      402.000      403.000      404.000
   5        501.000      502.000      503.000      504.000
   6        601.000      602.000      603.000      604.000
   7        701.000      702.000      703.000      704.000
```

```
      8        801.000      802.000      803.000      804.000
      9        901.000      902.000      903.000      904.000

  9 wart subsets listed.
```

So far, so good. But suppose we entered *undomacro* by mistake - can we "undo" the *undomacro* command? Yes, since *undomacro* is the most recent directive which changed the active file. This is shown in the continuing dialog:

```
edit> undo

   The active file has reverted back to how it was
   before the last UndoMacro command.

edit> tab

   wart #      col1         col2         col3         col4

      1        1101.00      -102.000     103.000      104.000
      2        1201.00      -202.000     203.000      204.000
      3        1301.00      -302.000     303.000      304.000
      4        1401.00      -402.000     403.000      404.000
      5        1501.00      -502.000     503.000      504.000
      6        1601.00      -602.000     603.000      604.000
      7        1701.00      -702.000     703.000      704.000
      8        1801.00      -802.000     803.000      804.000
      9        1901.00      -902.000     903.000      904.000

  9 wart subsets listed.
```

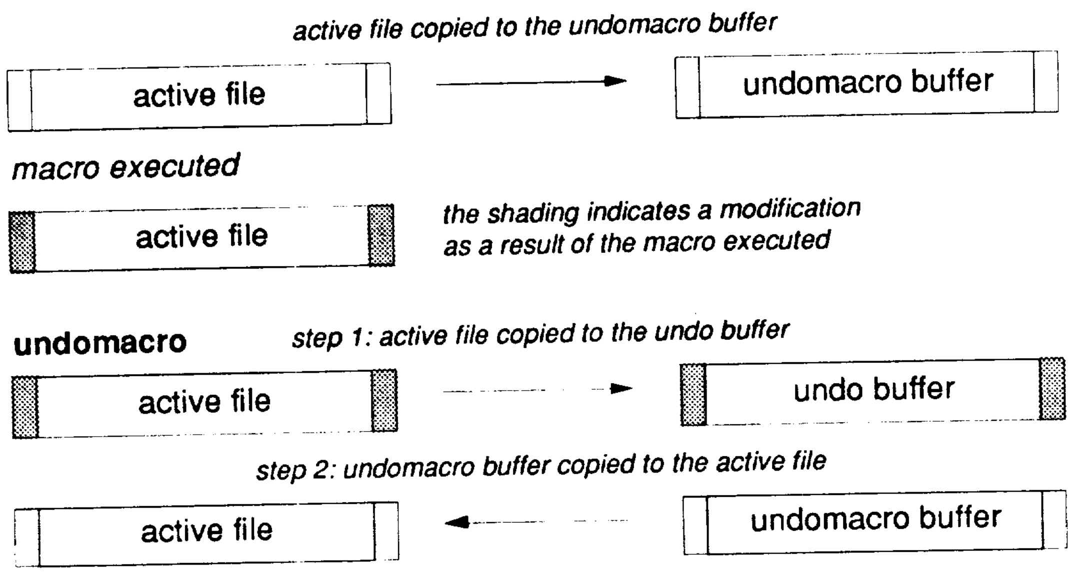
Oops! Maybe we wanted to undo that macro after all - can we recover the file back to where it was just after *undomacro* (or just before the last *undo*)? Sure, as shown below:

```
edit> undo

   The active file has reverted back to how it was
   before the last Undo command.

edit> tab

   wart #      col1         col2         col3         col4

      1        101.000      102.000      103.000      104.000
      2        201.000      202.000      203.000      204.000
      3        301.000      302.000      303.000      304.000
      4        401.000      402.000      403.000      404.000
      5        501.000      502.000      503.000      504.000
      6        601.000      602.000      603.000      604.000
      7        701.000      702.000      703.000      704.000
      8        801.000      802.000      803.000      804.000
      9        901.000      902.000      903.000      904.000

  9 wart subsets listed.
```

```
edit>
```

Notice that *undomacro* is itself undone via *undo*, as opposed to another *undomacro*. Why? Consider what it means to use two consecutive *undomacro* directives. The first *undomacro* revokes all changes made by the previous macro execution, but what should the second *undomacro* do? We can't revoke a macro previous to the one most recently executed, so we've already revoked the only macro we can. Therefore the second *undomacro* again restores the active file to its state immediately before the last macro execution. Because the first *undomacro* already did this, the second *undomacro* merely duplicates the restoration and has no real effect upon the active file.

Commands *undo* and *undomacro* have some interesting and very handy interactions. Graphically portrayed, a macro execution and subsequent *undomacro* operation appear as



In other words, after the active file is copied to the undo buffer, *undomacro* replaces the active file with the undomacro buffer, provided that the undomacro buffer was initially filled via a macro execution.

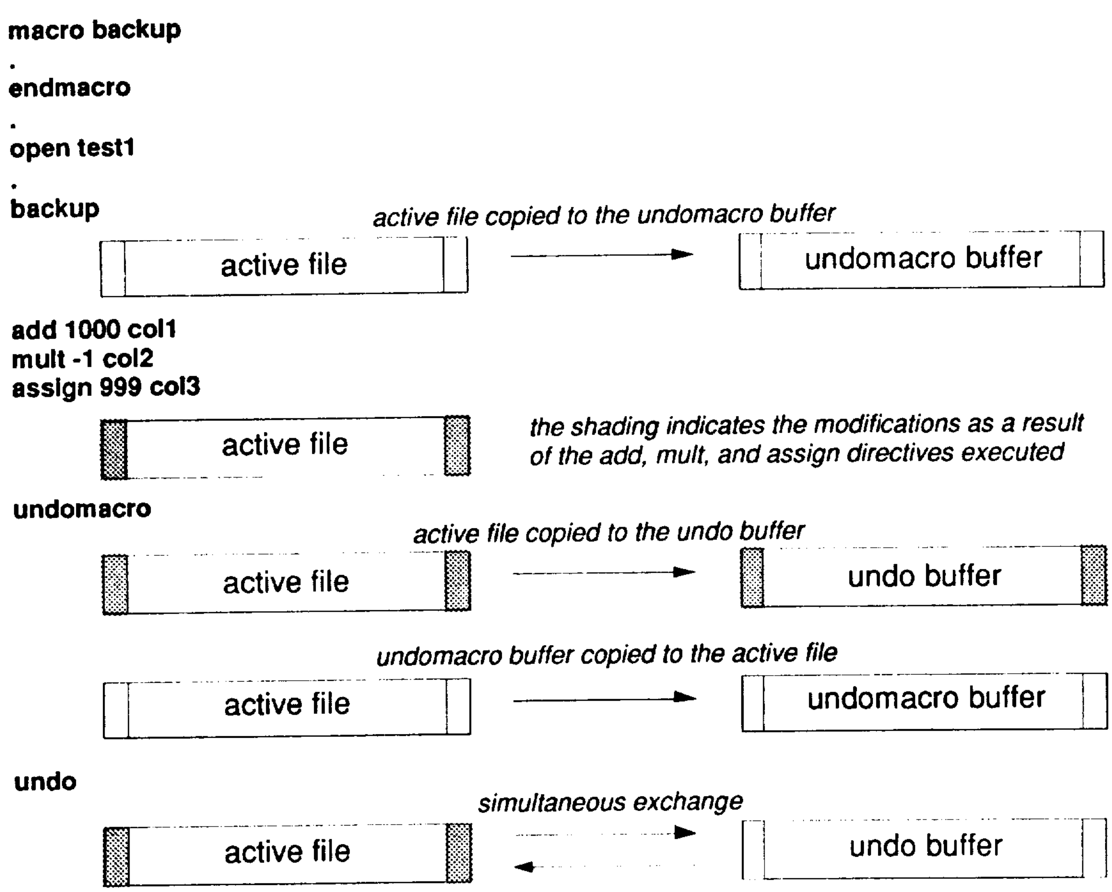
Unlike *undo* and the undo buffer, an *undomacro* does not exchange the active file and the undomacro buffer. Thus it is possible to issue an *undomacro* long after other directives have made substantial changes to the active file. This has the effect of a "superundo" because it can revoke the effects of a series of directives, compared to the *undo* command's ability to revoke only the most recent directive. Some users deliberately create a null macro just for this purpose. As an illustration, consider the following dialog (the *echo* command is discussed later in this section):

```
edit> disable  macroecho
edit> macro  backup
macro> echo
macro> echo  Active  file  written  to  undomacro  buffer.
macro> echo
macro> endmacro
edit> open  test1
edit> backup

          Active file written to undomacro buffer.
```

```
edit> tab

    wart #      col1        col2        col3        col4

      1       101.000     102.000     103.000     104.000
      2       201.000     202.000     203.000     204.000
      3       301.000     302.000     303.000     304.000
      4       401.000     402.000     403.000     404.000
      5       501.000     502.000     503.000     504.000
      6       601.000     602.000     603.000     604.000
      7       701.000     702.000     703.000     704.000
      8       801.000     802.000     803.000     804.000
      9       901.000     902.000     903.000     904.000

  9 wart subsets listed.

edit> add  1000  col1

  9 data warts changed.

edit> mult  -1  col2

  9 data warts changed.

edit> assign  999  col3
edit> tab

    wart #      col1        col2        col3        col4

      1       1101.00    -102.000     999.000     104.000
      2       1201.00    -202.000     999.000     204.000
      3       1301.00    -302.000     999.000     304.000
      4       1401.00    -402.000     999.000     404.000
      5       1501.00    -502.000     999.000     504.000
      6       1601.00    -602.000     999.000     604.000
      7       1701.00    -702.000     999.000     704.000
      8       1801.00    -802.000     999.000     804.000
      9       1901.00    -902.000     999.000     904.000

  9 wart subsets listed.

edit> undomacro

  The active file has reverted back to how it was
  before the last macro was executed.

edit> tab

    wart #      col1        col2        col3        col4

      1       101.000     102.000     103.000     104.000
      2       201.000     202.000     203.000     204.000
      3       301.000     302.000     303.000     304.000
```

```
 4        401.000      402.000      403.000      404.000
 5        501.000      502.000      503.000      504.000
 6        601.000      602.000      603.000      604.000
 7        701.000      702.000      703.000      704.000
 8        801.000      802.000      803.000      804.000
 9        901.000      902.000      903.000      904.000

   9 wart subsets listed.

edit> undo

   The active file has reverted back to how it was
   before the last UndoMacro command.

edit>
```

Notice that the second *undomacro* revokes the effects of the preceding <u>three</u> directives, which is beyond the capabilities of a normal *undo*. The final *undo* revokes the effects of the *undomacro* command. This dialog can be portrayed graphically as:



One last point. Once a past version of the active file is in the undomacro buffer it can be recalled at any time in the future. We could enter another series of directives (excluding macro executions) and again use *undomacro* to restore the version of the active file saved by macro *backup*. Thus a null macro and *undomacro* provide a simple means for intermediate file backups.

Creating a Directive File from a Macro

Macros generally offer more control (via *undomacro*) and more flexibility (via parameters) than a directive file equivalent. However, there may be times when you wish to create a directive file from a macro using specific parameters. As an example, suppose we've been determining turbopump rpm's using the macro *findrpm* :

```
macro findrpm base delta final
create $final
add $base $delta $final
mult $final .918333
endmacro
```

Further, suppose we've noticed that the base rpm and deltarpm are almost always measurement id's *msid1021* and *msid1078*, respectively, and that the final rpm goes into *rpmloxpump*. Instead of continually entering

```
findrpm msid1021 msid1078 rpmloxpump
```

we would prefer to enter

```
include loxrpm
```

In other words, we want to take a specific instance of a macro and turn it into a directive file.

The key is the session file. By default, directives executed within a macro are echoed to the terminal screen but are not echoed to the session file. (Recall that the session file is intended to serve as a step-by-step record of your entries, to the extent that you could use the resulting session file as a directive file to exactly duplicate the editing session. Echoing the call to the macro is proper. Echoing each directive within the macro defeats the original purpose of the session file because, if used as a directive file in a subsequent editing session, it would twice execute each directive within the macro.) This can be changed by entering

```
enable expand
```

which enables the session file expansion toggle. To illustrate, consider the following UNIX dialog:

```
% toaded


                  ----------------------------------

                    T O A D   F i l e   E d i t o r

                      Release 1.0    October 1990

                  ----------------------------------
```

```
[No startup file]

edit> macro findrpm base delta final
edit> create $final
edit> add $base $delta $final
edit> mult $final .918333
edit> endmacro
edit> enable expand
edit> findrpm msid1021 msid1078 rpmloxpump
edit> quit

Normal session.

% cat session
!
!  TOAD Editor session file.
!
macro findrpm base delta final
create $final
add $base $delta $final
mult $final .918333
endmacro
enable expand
findrpm msid1021 msid1078 rpmloxpump
!
!  Expanding macro findrpm.
!
create rpmloxpump
add msid1021 msid1078 rpmloxpump
mult rpmloxpump .918333
!
!  End macro expansion.
!
quit
```

Now all we have to do is edit the session file and copy the highlighted directives to the file *loxrpm*, which would then be available as a directive file.

Embedding Messages within Directive Files and Macros

The ***echo*** command is similar to the UNIX shell command *echo* -- whatever text follows the command is written to your terminal screen. Its form is

echo *text*

where *text* is the text string to be displayed. For example, the directive

echo LOX pump calculations finished . . . Begin H2 pump

displays the message

```
LOX pump calculations finished . . . Begin H2 pump
```

The *echo* command is designed to be used within the startup file and directive files to provide some measure of progress feedback. For example, consider the startup file

```
#
# TOAD Editor startup file
#
echo Begin startup sequence...
.
[environmental settings]
.
echo ...Environmentals set
.
[symbol definitions]
.
echo ...Symbols defined
.
[macro definitions]
.
echo ...Macros defined
echo End startup sequence
```

Such a startup file would display the following messages at the beginning of each editing session:

```
Begin startup sequence...
...Environmentals set
...Symbols defined
...Macros defined
End startup sequence
```

Such messages can be highly customized within macros by virtue of the macros ability to perform parameter substitution. Consider our previous example macro *yanklep*:

```
macro yanklep file1 file2
open $file1
define leading_edge=0
min x/c leading_edge
export $file2 deltacp x/c leading_edge 2y/b
delsymbol leading_edge
close
endmacro
```

This macro could be modified to include a few *echo* directives, such as

```
macro yanklep file1 file2
#
echo
echo Using raw data file $file1 to create summary file $file2
echo
echo Leading edge x/c location:
min x/c
#
open $file1
define leading_edge=0
```

```
min x/c leading_edge
export $file2 deltacp x/c leading_edge 2y/b
delsymbol leading_edge
close
#
endmacro
```

Now macro *yanklep* provides some feedback when it's executed:

```
edit> yanklep  run203  lep203

        Using raw data file run203 to create summary file lep203

        Leading edge x/c location:

            .025

edit>
```

Notice that, in the above macro example, parameter substitution was performed. If you would prefer not to have this substitution performed, either leave off the macro character prefix

```
echo Using raw data file file1 to create summary file file2
```

which creates the message

```
Using raw data file file1 to create summary file file2
```

or surround the parameter with single or double quotation marks:

```
echo Using raw data file '$file1' to create summary file "$file2"
```

which creates the message

```
Using raw data file $file1 to create summary file $file2
```

The *echo* command can also be used as a debugging tool when developing a new macro. For example, again using the macro yanklep:

```
macro yanklep file1 file2
open $file1
define leading_edge=0
min x/c leading_edge
export $file2 deltacp x/c leading_edge 2y/b
delsymbol leading_edge
close
endmacro
```

suppose we aren't sure that the macro parameters *file1* and *file2* are coming in correctly. We could verify their values by adding the directives

```
echo file1: $file1
echo file2: $file2
```

and commenting out the *export* directive, as illustrated below:

```
macro yanklep file1 file2
echo 'file1: ' file1
echo 'file2: ' file2
open $file1
define leading_edge=0
min x/c leading_edge
#export $file2 deltacp x/c leading_edge 2y/b
delsymbol leading_edge
close
endmacro
```

When executed, this version of the macro only displays the values of its two parameters. For example, entering

```
yanklep run21 lep21
```

displays the messages

```
file1: run21
file2: lep21
```

which verifies that the desired file names were indeed brought in correctly. This technique is particularly useful for tracing parameters passed down through many macro layers. As a more realistic example, consider the following macro hierarchy:

```
macro wingstats span chord pressure lefile tefile tipfile
FrontBackCp $chord $pressure $lefile $tefile
OutboardCp $span $pressure $tipfile
endmacro

macro FrontBackCp xloc Cp lefile tefile
le $xloc $Cp $lefile
te $xloc $Cp $tefile
endmacro

macro le x scalar file
define xle = 0
min $x xle
export $file $x xle $scalar
delsymbol xle
endmacro

macro te x scalar file
define xte = 0
max $x xte
export $file $x xte $scalar
delsymbol xte
endmacro

macro OutboardCp y scalar file
define ytip = 0
```

```
max $y ytip
export $file $y ytip $scalar
delsymbol ytip
endmacro
```

By instrumenting the macro scripts (i.e., inserting diagnostic *echo* directives and commenting out the active directives) we can trace all of the parameters used. Instrumented versions of these macros might be

```
  macro wingstats span chord pressure lefile tefile tipfile
* echo wingstats incoming parameters:
* echo span: $span
* echo chord: $chord
* echo pressure: $pressure
* echo lefile: $lefile
* echo tefile: $tefile
* echo tipfile: $tipfile
* echo
* echo calling FrontBackCp
  FrontBackCp $chord $pressure $lefile $tefile
* echo
* echo calling OutboardCp
  OutboardCp $span $pressure $tipfile
* echo
* echo macro wingstats complete
  endmacro

  macro FrontBackCp xloc Cp lefile tefile
* echo entering FrontBackCp with parameters $xloc $Cp $lefile $tefile
* echo
* echo calling le
  le $xloc $Cp $lefile
* echo
* echo calling te
  te $xloc $Cp $tefile
* echo
* echo macro FrontBackCp complete
  endmacro

  macro le x scalar file
* echo entering le with parameters $x $scalar $file
  define xle = 0
  #min $x xle
  #export $file $x xle $scalar
  delsymbol xle
* echo macro le complete
  endmacro

  macro te x scalar file
* echo entering te with parameters $x $scalar $file
  define xte = 0
  #max $x xte
  #export $file $x xte $scalar
```

```
   delsymbol xte
*  echo macro te complete
   endmacro

   macro OutboardCp y scalar file
*  echo entering OutboardCp with parameters $y $scalar $file
   define ytip = 0
   #max $y ytip
   #export $file $y ytip $scalar
   delsymbol ytip
*  echo macro OutboardCp complete
   endmacro
```

where the * marks the *echo* directives added during instrumentation. If we execute macro *wingstats* by entering

```
wingstats 2y/b x/c deltacp run23le run23te run23tip
```

the following messages are displayed:

```
wingstats incoming parameters:
span: 2y/b
chord: x/c
pressure: deltacp
lefile: run23le
tefile: run23te
tipfile: run23tip

calling FrontBackCp
entering FrontBackCp with parameters x/c deltacp run23le run23te

calling le
entering le with parameters x/c deltacp run23le
macro le complete

calling te
entering te with parameters x/c deltacp run23te
macro te complete

macro FrontBackCp complete

calling OutboardCp
entering OutboardCp with parameters 2y/b deltacp run23tip
macro OutboardCp complete

macro wingstats complete
```

which verifies that all macro parameters are passed as expected.

Changing the Macro Character and Continuation Character

Under normal conditions the macro character and continuation character remain as dollar sign ($) and ampersand (&), respectively. However, there may be an occasion when changing either or both may be more convenient. For example, if your TOAD file contains variables beginning with a dollar sign and you plan to use macros, it's probably in your best interest to change the macro character to something other than dollar sign.

The process of changing either control character is usually performed in three steps: save the current setting for later restoration, change the setting, and restore the setting back to its original state. The second step, change the setting, is accomplished via *set* and has already been covered under subsection "Environmentals," beginning on page 15. The first and third steps are accomplished via the ***store*** and ***restore*** commands, respectively. Their forms are

store *environmental*
restore *environmental*

where *environmental* is a keyword identifying the environmental to be stored or restored. Only two environmentals are currently available: the macro character (keywords *macrochar* or *mchar*) and the continuation character (keywords *contchar* or *cchar*). As an illustration of how these commands are used, consider the following dialog:

```
edit> show  macrochar

        The macro character is '$'.

edit> store  macrochar
edit> set  macrochar  @
edit> show  macrochar

        The macro character is '@'.

        One previous macro character is available:

             -1   '$'

edit> restore  macrochar

        Macro character restored to '$'.

edit> show  macrochar

        The macro character is '$'.

edit>
```

Conceptually, *store* writes the environmental's setting to a "stack" or LIFO (Last In, First Out) list, and *restore* reads an environmental's setting from the stack. Additional *store* and *restore* directives write and read additional entries in the stack, as shown in the following dialog:

```
edit> show  macrochar

        The macro character is '$'.

edit> store  macrochar
edit> set  macrochar  @
```

```
edit> store macrochar
edit> set macrochar %
edit> store macrochar
edit> set macrochar ~
edit> show macrochar

        The macro character is '~'.

        3 previous macro characters are available:

            -1   '%'
            -2   '@'
            -3   '$'

edit> restore macrochar

        Macro character restored to '%'.

edit> show macrochar

        The macro character is '%'.

        2 previous macro characters are available:

            -1   '@'
            -2   '$'

edit> -2
        [ restore macrochar ]

        Macro character restored to '@'.

edit> -1
        [ restore macrochar ]

        Macro character restored to '$'.

edit>
```

One particularly useful application of *store* and *restore* involves changing the macro character within a single macro. For example, suppose we have a TOAD file which we know contains variables beginning with a dollar sign. We want to write a macro which will be effective for this file yet we want the macro to be useful for other TOAD files as well. How can this be accomplished?

The answer is to store, change, and restore the macro character within the macro itself. For example, if the macro is supposed to add two columns together into a third, then multiply the result by 90%, we'd normally write the macro as:

```
macro fix p1 p2 p3
add $p1 $p2 $p3
multiply $p3 .9
endmacro
```

However, because the TOAD file we're using contains variables beginning with dollar signs, this macro may work but it would be very confusing to debug or read in the session file. The solution is to use a different macro character only within this macro. An alternate macro definition is

```
macro fix p1 p2 p3
store macrochar
set macrochar @
add @p1 @p2 @p3
multiply @p3 .9
restore macrochar
endmacro
```

which changes the macro character to "@" only for the duration of the macro. This revised macro satisfies both of our requirements: it accommodates variable names beginning with a dollar sign yet is useful for general TOAD files.

Helpful Hint

The *store* and *restore* commands are intended for the advanced user who prefers a highly customized editing environment. Because of the complexities involved, we do not recommend changing either the macro character or the continuation character. In general, variable names which begin with a dollar sign (the default macro character) or end with an ampersand (the default continuation character) are best renamed.

Section 6
In Case of Problems ...

General

No software is above design and development errors. If you uncover an error, or notice some strange behavior, please follow the steps described below. One minute of your time may save others hours or even days of effort.

Langley Users - All Systems

If possible, assemble the following information:

1. Your host computer's manufacturer, model, operating system, and location.
2. The name of the active TOAD file.
3. A directive sequence which reproduces the error, or a description of the operations performed immediately before the error occurred.

Then call Bradford Bingel at Computer Sciences Corporation, (804) 865-1725. Every attempt will be made to correct the problem, when possible, within a few minutes.

Non-Langley Users - All Systems

Computer Sciences Corporation does not support the TOAD Editor outside of NASA Langley. All questions and problems concerning this software should be directed to Dr. John E. Lamar, mail stop 361, (804) 864-2851.

All comments are appreciated and welcomed !!!

Appendix A
Sample Sessions

Sample Session #1

*The file **toad1** contains general pressure data over the surface of an aircraft wing. We want to familiarize ourselves with the file, then extract five subsets of data: pressure as a function of chord location at three spanwise locations, and pressure as a function of spanwise location along the wing's leading and trailing edges.*

```
% toaded


                ----------------------------------

                  T O A D   F i l e   E d i t o r

                    Release 1.0    October 1990

                ----------------------------------



         [No startup file]

edit> open  toad1
edit> menu

         This TOAD file contains 6 variables:

              mach                cldes               planform
              x/c                 2y/b                deltacp
```

Because we may have to modify this file, we'll make a copy, and work from the copy.

```
edit> save  toad1m

         This request will overwrite the original contents of
         an existing file.  Do you really want it performed ?
> y
```

*Note: This question appears only when **toadm1** already exists.*

```
edit> op  toad1m
edit> show  targ

         The entire TOAD file.
```

```
edit> menu

        This TOAD file contains 6 variables:

             mach                  cldes                 planform
             x/c                   2y/b                  deltacp
```

We suspect that some of the variables in this TOAD file are not of interest in this session, so we'll use the ***stats*** *command to help identify them. Because variables* ***cldes*** *and* ***planform*** *are constant and fulfill no useful purpose in this analysis, they are removed. We also use* ***stats*** *to determine that variable* ***mach*** *is constant at .6.*

```
edit> stat  cldes

        Frequency Count:    150
        Sum:                150
        Range:              0
        Minimum:            1
        Maximum:            1
        Mean:               1
        Variance:           0                         (unbiased)
        Standard Dev:       0                         (biased)
        Standard Error:     0                         (biased)

edit> del  cldes
edit> stat  planform

        Frequency Count:    150
        Sum:                150
        Range:              0
        Minimum:            1
        Maximum:            1
        Mean:               1
        Variance:           0                         (unbiased)
        Standard Dev:       0                         (biased)
        Standard Error:     0                         (biased)

edit> del  planform
edit> stat  mach

        Frequency Count:    150
        Sum:                90
        Range:              0
        Minimum:            0.6
        Maximum:            0.6
        Mean:               0.6
        Variance:           0                         (unbiased)
        Standard Dev:       0                         (biased)
        Standard Error:     0                         (biased)
```

```
edit> scan

        This TOAD file contains 4 variables:

            mach                x/c                 2y/b
            deltacp

        . . . and has a total of 150 data warts.
```

*Looking at the first few data warts, we notice two things: the wing semispan location (**2y/b**) should come before the airfoil chord location (**x/c**), and that the wing semispan location is ordered numerically, from wing's inboard root to its outboard tip. We want the data arranged differently, so we exchange **2y/b** and **x/c** and perform a descending sort on **2y/b**.*

```
edit> tab  1t10

          wart #      mach          x/c           2y/b          deltacp

            1       0.600000    0.416667E-01   0.200000E-01    1.05376
            2       0.600000    0.208333       0.200000E-01   0.843582
            3       0.600000    0.375000       0.200000E-01   0.737244
            4       0.600000    0.541667       0.200000E-01   0.622884
            5       0.600000    0.708333       0.200000E-01   0.479829
            6       0.600000    0.875000       0.200000E-01   0.292267
            7       0.600000    0.416667E-01   0.600000E-01    1.16457
            8       0.600000    0.208333       0.600000E-01   0.864949
            9       0.600000    0.375000       0.600000E-01   0.748665
            10      0.600000    0.541667       0.600000E-01   0.631585

        10 wart subsets listed.

edit> x  x/c  2y/b
edit> -2
        [ tab 1t10 ]

          wart #      mach          2y/b          x/c           deltacp

            1       0.600000    0.200000E-01   0.416667E-01    1.05376
            2       0.600000    0.200000E-01   0.208333       0.843582
            3       0.600000    0.200000E-01   0.375000       0.737244
            4       0.600000    0.200000E-01   0.541667       0.622884
            5       0.600000    0.200000E-01   0.708333       0.479829
            6       0.600000    0.200000E-01   0.875000       0.292267
            7       0.600000    0.600000E-01   0.416667E-01    1.16457
            8       0.600000    0.600000E-01   0.208333       0.864949
            9       0.600000    0.600000E-01   0.375000       0.748665
            10      0.600000    0.600000E-01   0.541667       0.631585

        10 wart subsets listed.

edit> sort  2y/b  d
```

```
edit> -2
        [ tab 1t10 ]
```

wart #	mach	2y/b	x/c	deltacp
1	0.600000	0.980000	0.416667E-01	10.5822
2	0.600000	0.980000	0.208333	5.61437
3	0.600000	0.980000	0.375000	4.43053
4	0.600000	0.980000	0.541667	3.78733
5	0.600000	0.980000	0.708333	3.15396
6	0.600000	0.980000	0.875000	2.12793
7	0.600000	0.940000	0.416667E-01	7.38284
8	0.600000	0.940000	0.208333	3.76906
9	0.600000	0.940000	0.375000	2.71414
10	0.600000	0.940000	0.541667	2.03536

```
10 wart subsets listed.
```

The first few data warts also tell us that there are 6 chord locations at each wing semispan location. To check the file's integrity, let's display the first data wart within each block of 6 warts associated with each semispan location.

edit> **tab 1t145b6**

wart #	mach	2y/b	x/c	deltacp
1	0.600000	0.980000	0.416667E-01	10.5822
7	0.600000	0.940000	0.416667E-01	7.38284
13	0.600000	0.900000	0.416667E-01	6.09007
19	0.600000	0.860000	0.416667E-01	5.30779
25	0.600000	0.820000	0.416667E-01	4.75334
31	0.600000	0.780000	0.416667E-01	4.32299
37	0.600000	0.740000	0.416667E-01	3.97186
43	0.600000	0.700000	0.416667E-01	3.67719
49	0.600000	0.660000	0.416667E-01	3.42449
55	0.600000	0.620000	0.416667E-01	3.20335
61	0.600000	0.580000	0.416667E-01	3.00611
67	0.600000	0.540000	0.416667E-01	2.82717
73	0.600000	0.500000	0.416667E-01	2.66236
79	0.600000	0.460000	0.416667E-01	2.50851
85	0.600000	0.420000	0.416667E-01	2.36313
91	0.600000	0.380000	0.416667E-01	2.22420
97	0.600000	0.340000	0.416667E-01	2.09000
103	0.600000	0.300000	0.416667E-01	1.95902
109	0.600000	0.260000	0.416667E-01	1.82997
115	0.600000	0.220000	0.416667E-01	1.70156

[Return]

wart #	mach	2y/b	x/c	deltacp
121	0.600000	0.180000	0.416667E-01	1.57196
127	0.600000	0.140000	0.416667E-01	1.43850
133	0.600000	0.100000	0.416667E-01	1.29960
139	0.600000	0.600000E-01	0.416667E-01	1.16457
145	0.600000	0.200000E-01	0.416667E-01	1.05376

```
25 wart subsets listed.
```

Good. Since there are 150 data warts in all, and 6 warts per semispan location, we should see 25 semispan locations' worth of data. Let's try it again, only this time we'll isolate the third wart within each block.

```
edit> "  3t147b6

        wart #      mach        2y/b          x/c          deltacp

          3      0.600000    0.980000      0.375000       4.43053
          9      0.600000    0.940000      0.375000       2.71414
         15      0.600000    0.900000      0.375000       2.12340
         21      0.600000    0.860000      0.375000       1.82227
         27      0.600000    0.820000      0.375000       1.62033
         33      0.600000    0.780000      0.375000       1.47393
         39      0.600000    0.740000      0.375000       1.36207
         45      0.600000    0.700000      0.375000       1.27218
         51      0.600000    0.660000      0.375000       1.19711
         57      0.600000    0.620000      0.375000       1.13281
         63      0.600000    0.580000      0.375000       1.07686
         69      0.600000    0.540000      0.375000       1.02764
         75      0.600000    0.500000      0.375000       0.984011
         81      0.600000    0.460000      0.375000       0.945151
         87      0.600000    0.420000      0.375000       0.910150
         93      0.600000    0.380000      0.375000       0.879452
         99      0.600000    0.340000      0.375000       0.851804
        105      0.600000    0.300000      0.375000       0.827234
        111      0.600000    0.260000      0.375000       0.805593
        117      0.600000    0.220000      0.375000       0.786984
[Return]
        123      0.600000    0.180000      0.375000       0.771897
        129      0.600000    0.140000      0.375000       0.761138
        135      0.600000    0.100000      0.375000       0.754821
        141      0.600000    0.600000E-01  0.375000       0.748665
        147      0.600000    0.200000E-01  0.375000       0.737244

    25 wart subsets listed.
```

Good. Now let's set up a default target list and see if the same data is tabulated.

```
edit> show tol

     The default tolerance is 1% (relative).

edit> targ mach 2y/b x/c .375 deltacp
edit> tab

        wart #      mach        2y/b        deltacp

          3      0.600000    0.980000      4.43053
          9      0.600000    0.940000      2.71414
         15      0.600000    0.900000      2.12340
```

```
  21      0.600000     0.860000       1.82227
  27      0.600000     0.820000       1.62033
  33      0.600000     0.780000       1.47393
  39      0.600000     0.740000       1.36207
  45      0.600000     0.700000       1.27218
  51      0.600000     0.660000       1.19711
  57      0.600000     0.620000       1.13281
  63      0.600000     0.580000       1.07686
  69      0.600000     0.540000       1.02764
  75      0.600000     0.500000       0.984011
  81      0.600000     0.460000       0.945151
  87      0.600000     0.420000       0.910450
  93      0.600000     0.380000       0.879452
  99      0.600000     0.340000       0.851804
 105      0.600000     0.300000       0.827234
 111      0.600000     0.260000       0.805593
 117      0.600000     0.220000       0.786984
```

[Return]

```
 123      0.600000     0.180000       0.771897
 129      0.600000     0.140000       0.761138
 135      0.600000     0.100000       0.754821
 141      0.600000     0.600000E-01   0.748665
 147      0.600000     0.200000E-01   0.737244

25 wart subsets listed.
```

*Great! Recall our objective in this first sample session is to create a series of data files: pressure (**deltacp**) as a function of airfoil chord location at three semispan locations; and pressure as a function of semispan location along the wing's leading and trailing edges. As a reminder, let's look at the first few data warts again. We know we want to use the export command, but can't remember its syntax, so we'll also use the help facility.*

edit> **targ all**
edit> **tab 1t15**

wart #	mach	2y/b	x/c	deltacp
1	0.600000	0.980000	0.416667E-01	10.5822
2	0.600000	0.980000	0.208333	5.61437
3	0.600000	0.980000	0.375000	4.43053
4	0.600000	0.980000	0.541667	3.78733
5	0.600000	0.980000	0.708333	3.15396
6	0.600000	0.980000	0.875000	2.12793
7	0.600000	0.940000	0.416667E-01	7.38284
8	0.600000	0.940000	0.208333	3.76906
9	0.600000	0.940000	0.375000	2.71414
10	0.600000	0.940000	0.541667	2.03536
11	0.600000	0.940000	0.708333	1.50435
12	0.600000	0.940000	0.875000	0.937932
13	0.600000	0.900000	0.416667E-01	6.09007
14	0.600000	0.900000	0.208333	3.02826
15	0.600000	0.900000	0.375000	2.12340

```
        15 wart subsets listed.

edit> h  export

        EXPORT     writes a multi-column data fragment.

        syntax:    Export  [file]  [object list]

                   file    the name of the file to be written.  If
                           omitted, "tadpole" is assumed.

                   object list   see the help text for command Target
                                 If omitted, the default target list
                                 is assumed.

        info:      Command Write is simpler for single-column data.
                   If you're moving entire warts, commands CopyWart
                   and CutWart may be simpler.

        aliases:   extract
```

Now that we have the export command's syntax, we'll create the first three files.

```
edit> export  toad98  mach  2y/b  .98  x/c  deltacp

        This request will overwrite the original contents of
        an existing file.  Do you really want it performed ?
> y
```

*Note: This question appears only when **toad98** already exists.*

```
        4 variables, 6 warts written.

edit> "  toad94  mach  2y/b  .94  x/c  deltacp

        This request will overwrite the original contents of
        an existing file.  Do you really want it performed ?
> y
```

*Note: This question appears only when **toad94** already exists.*

```
        4 variables, 6 warts written.

edit> "  toad90  mach  2y/b  .9  x/c  deltacp

        This request will overwrite the original contents of
        an existing file.  Do you really want it performed ?
> y
```

*Note: This question appears only when **toad90** already exists.*

```
        4 variables, 6 warts written.
```

No problems there. We expected four variables and six data warts for each file, and that's exactly what happened. Now for the last two files. How did we do that hopscotch tabulation? Let's check the directive history and try to recreate it.

```
edit> hist

        12  del planform
        13  stat mach
        14  scan
        15  tab 1t10
        16  x x/c 2y/b
        17  tab 1t10
        18  sort 2y/b d
        19  tab 1t10
        20  tab 1t145b6
        21  " 3t147b6
        22  show tol
        23  targ mach 2y/b x/c .375 deltacp
        24  tab
        25  targ all
        26  tab 1t15
        27  h export
        28  export toad98 mach 2y/b .98 x/c deltacp
        29  " toad94 mach 2y/b .94 x/c deltacp
        30  " toad90 mach 2y/b .9 x/c deltacp
        31  hist

edit> 20
    [ tab 1t145b6 ]
```

wart #	mach	2y/b	x/c	deltacp
1	0.600000	0.980000	0.416667E-01	10.5822
7	0.600000	0.940000	0.416667E-01	7.38284
13	0.600000	0.900000	0.416667E-01	6.09007
19	0.600000	0.860000	0.416667E-01	5.30779
25	0.600000	0.820000	0.416667E-01	4.75334
31	0.600000	0.780000	0.416667E-01	4.32299
37	0.600000	0.740000	0.416667E-01	3.97186
43	0.600000	0.700000	0.416667E-01	3.67719
49	0.600000	0.660000	0.416667E-01	3.42449
55	0.600000	0.620000	0.416667E-01	3.20335
61	0.600000	0.580000	0.416667E-01	3.00611
67	0.600000	0.540000	0.416667E-01	2.82717
73	0.600000	0.500000	0.416667E-01	2.66236
79	0.600000	0.460000	0.416667E-01	2.50851
85	0.600000	0.420000	0.416667E-01	2.36313
91	0.600000	0.380000	0.416667E-01	2.22420
97	0.600000	0.340000	0.416667E-01	2.09000
103	0.600000	0.300000	0.416667E-01	1.95902
109	0.600000	0.260000	0.416667E-01	1.82997
115	0.600000	0.220000	0.416667E-01	1.70156

[Return]

121	0.600000	0.180000	0.416667E-01	1.57196
127	0.600000	0.140000	0.416667E-01	1.43850
133	0.600000	0.100000	0.416667E-01	1.29960
139	0.600000	0.600000E-01	0.416667E-01	1.16457
145	0.600000	0.200000E-01	0.416667E-01	1.05376

```
25 wart subsets listed.
```

*Notice that the chord location (**x/c**) is constant at .041667, or 1/24th. These are quarter-chord locations of panel control points, so we expected them to be 1/6th apart, beginning at 1/24th. Sure enough, 1/24th plus 1/6 is 5/24ths, or .208333, and 1/24th plus 5/6ths is 21/24ths, or .875. We could specify the leading edge and trailing edge as the numeric values 1/24 and 21/24, respectively, but it is easier to create and use two symbols (**le** and **t e**) for this purpose.*

```
edit> define le 0
edit> " te 0
edit> min x/c le
edit> max x/c te
edit> sho sym

     The defined symbols are:

          le          = 0.04166667
          te          = 0.875

edit> export toadle mach 2y/b x/c le deltacp

     This request will overwrite the original contents of
     an existing file.  Do you really want it performed ?
> y
```

*Note: This question appears only when **toadle** already exists.*

```
     4 variables, 25 warts written.

edit> export toadte mach 2y/b x/c te deltacp

     This request will overwrite the original contents of
     an existing file.  Do you really want it performed ?
> y
```

*Note: This question appears only when **toadte** already exists.*

```
     4 variables, 25 warts written.
```

Again, the number of variables and data warts written matches what we expect. After making sure we're editing the right file, let's save it, then open and tabulate the new files we've just created.

```
edit> show file

     Active file: toadlm
```

```
edit> save

        This request will overwrite the original contents of
        an existing file.  Do you really want it performed ?
> y

edit> open  toad98
edit> tab
```

wart #	mach	2y/b	x/c	deltacp
1	0.600000	0.980000	0.416667E-01	10.5822
2	0.600000	0.980000	0.208333	5.61437
3	0.600000	0.980000	0.375000	4.43053
4	0.600000	0.980000	0.541667	3.78733
5	0.600000	0.980000	0.708333	3.15396
6	0.600000	0.980000	0.875000	2.12793

6 wart subsets listed.

```
edit> op  toad94
edit> tab
```

wart #	mach	2y/b	x/c	deltacp
1	0.600000	0.940000	0.416667E-01	7.38284
2	0.600000	0.940000	0.208333	3.76906
3	0.600000	0.940000	0.375000	2.71414
4	0.600000	0.940000	0.541667	2.03536
5	0.600000	0.940000	0.708333	1.50435
6	0.600000	0.940000	0.875000	0.937932

6 wart subsets listed.

```
edit> open  toad90
edit> tab
```

wart #	mach	2y/b	x/c	deltacp
1	0.600000	0.900000	0.416667E-01	6.09007
2	0.600000	0.900000	0.208333	3.02826
3	0.600000	0.900000	0.375000	2.12340
4	0.600000	0.900000	0.541667	1.60278
5	0.600000	0.900000	0.708333	1.17190
6	0.600000	0.900000	0.875000	0.711813

6 wart subsets listed.

```
edit> open  toadle
edit> tab
```

wart #	mach	2y/b	x/c	deltacp
1	0.600000	0.980000	0.416667E-01	10.5822
2	0.600000	0.940000	0.416667E-01	7.38284
3	0.600000	0.900000	0.416667E-01	6.09007
4	0.600000	0.860000	0.416667E-01	5.30779
5	0.600000	0.820000	0.416667E-01	4.75334
6	0.600000	0.780000	0.416667E-01	4.32299
7	0.600000	0.740000	0.416667E-01	3.97186
8	0.600000	0.700000	0.416667E-01	3.67719
9	0.600000	0.660000	0.416667E-01	3.42449
10	0.600000	0.620000	0.416667E-01	3.20335
11	0.600000	0.580000	0.416667E-01	3.00611
12	0.600000	0.540000	0.416667E-01	2.82717
13	0.600000	0.500000	0.416667E-01	2.66236
14	0.600000	0.460000	0.416667E-01	2.50851
15	0.600000	0.420000	0.416667E-01	2.36313
16	0.600000	0.380000	0.416667E-01	2.22420
17	0.600000	0.340000	0.416667E-01	2.09000
18	0.600000	0.300000	0.416667E-01	1.95902
19	0.600000	0.260000	0.416667E-01	1.82997
20	0.600000	0.220000	0.416667E-01	1.70156

[Return]

wart #	mach	2y/b	x/c	deltacp
21	0.600000	0.180000	0.416667E-01	1.57196
22	0.600000	0.140000	0.416667E-01	1.43850
23	0.600000	0.100000	0.416667E-01	1.29960
24	0.600000	0.600000E-01	0.416667E-01	1.16457
25	0.600000	0.200000E-01	0.416667E-01	1.05376

25 wart subsets listed.

```
edit> open  toadte
edit> tab
```

wart #	mach	2y/b	x/c	deltacp
1	0.600000	0.980000	0.875000	2.12793
2	0.600000	0.940000	0.875000	0.937932
3	0.600000	0.900000	0.875000	0.711813
4	0.600000	0.860000	0.875000	0.604424
5	0.600000	0.820000	0.875000	0.537458
6	0.600000	0.780000	0.875000	0.490354
7	0.600000	0.740000	0.875000	0.454841
8	0.600000	0.700000	0.875000	0.426885
9	0.600000	0.660000	0.875000	0.404250
10	0.600000	0.620000	0.875000	0.385577
11	0.600000	0.580000	0.875000	0.369983
12	0.600000	0.540000	0.875000	0.356865
13	0.600000	0.500000	0.875000	0.345797
14	0.600000	0.460000	0.875000	0.336463

```
        15      0.600000     0.420000        0.875000     0.328624
        16      0.600000     0.380000        0.875000     0.322091
        17      0.600000     0.340000        0.875000     0.316705
        18      0.600000     0.300000        0.875000     0.312327
        19      0.600000     0.260000        0.875000     0.308820
        20      0.600000     0.220000        0.875000     0.306035
```

[Return]

```
        21      0.600000     0.180000        0.875000     0.303791
        22      0.600000     0.140000        0.875000     0.301837
        23      0.600000     0.100000        0.875000     0.299793
        24      0.600000     0.600000E-01    0.875000     0.297028
        25      0.600000     0.200000E-01    0.875000     0.292267

    25 wart subsets listed.
```

Everything looks fine, so let's end this session.

```
edit> q

    Normal session.

%
```

For the reader's benefit, all of these TOAD files, including those created during this session, are available from the Langley Mustang directory

```
~ntflib/toad_examples
```

Sample Session #2

*We have four TOAD files, **toadrk1**, **toadrk2**, **toadrk3**, and **toadrk4**, that need to be merged into two files for use by the Program to Optimize Simulated Trajectories (POST). Once this example is complete the resultant files can be converted to POST table files by the TOAD Gateway.*

The four files contain both actual and coefficient rocket thrust values. The first three contain data for Mach less than one, and the fourth contains data for Mach greater than or equal to one.

```
% toaded

                    ----------------------------------

                    T O A D   F i l e   E d i t o r

                      Release 1.0    October 1990

                    ----------------------------------

      [No startup file]
```

*Let's work with **toadrk1** first.*

```
edit> open  toadrkl
edit> tab
```

wart #	rocket1	m	aoa
1	1.26000	0.100000	-5.00000
2	1.77000	0.900000	-5.00000
3	1.33000	0.100000	-3.00000
4	1.80000	0.900000	-3.00000
5	2.00000	0.100000	0.
6	2.50000	0.900000	0.
7	2.25000	0.100000	2.00000
8	2.75000	0.900000	2.00000
9	1.92000	0.100000	4.00000
10	1.60000	0.900000	4.00000

```
      10 wart subsets listed.
```

The variables and data values in this file are ordered in the fashion necessary for the TOAD Gateway to convert it into a POST table. The variable names, however, must be changed to the corresponding POST variable names.

```
edit> rename  m  mach
edit> "  aoa  alpha
```

```
edit> tab
```

wart #	rocket1	mach	alpha
1	1.26000	0.100000	-5.00000
2	1.77000	0.900000	-5.00000
3	1.33000	0.100000	-3.00000
4	1.80000	0.900000	-3.00000
5	2.00000	0.100000	0.
6	2.50000	0.900000	0.
7	2.25000	0.100000	2.00000
8	2.75000	0.900000	2.00000
9	1.92000	0.100000	4.00000
10	1.60000	0.900000	4.00000

```
10 wart subsets listed.
```

*According to the notes from the researcher, the thrust coefficients in **rocket1** need to be scaled by .963 due to the conditions of the test site as compared to the actual site.*

```
edit> mult  rocket1  .963

    10 data warts changed.

edit> tab
```

wart #	rocket1	mach	alpha
1	1.21338	0.100000	-5.00000
2	1.70451	0.900000	-5.00000
3	1.28079	0.100000	-3.00000
4	1.73340	0.900000	-3.00000
5	1.92600	0.100000	0.
6	2.40750	0.900000	0.
7	2.16675	0.100000	2.00000
8	2.64825	0.900000	2.00000
9	1.84896	0.100000	4.00000
10	1.54080	0.900000	4.00000

```
10 wart subsets listed.
```

This file is ready to be saved. We will use it later as our final table foundation for the subsonic data.

```
edit> save  toadnr1

    This request will overwrite the original contents of
    an existing file.  Do you really want it performed ?
> y
```

*Note: This question appears only when **toadnr1** already exists.*

*Now, let's look at **toadrk2** and see what needs to be done to it.*

```
edit> open  toadrk2
edit> tab
```

wart #	rocket2	aoasq	m
1	0.500000	-25.0000	0.100000
2	0.620000	-9.00000	0.100000
3	0.930000	0.	0.100000
4	0.860000	4.00000	0.100000
5	0.710000	16.0000	0.100000
6	1.00000	-25.0000	0.900000
7	1.26000	-9.00000	0.900000
8	1.38000	0.	0.900000
9	1.29000	4.00000	0.900000
10	1.16000	16.0000	0.900000

10 wart subsets listed.

*These thrust coeficients are listed as rocket2=f(aoasq,m). **aoasq**, angle of attack squared, is not acceptable in POST, so we have to convert it to alpha. To do so, we must first save off the sign of **aoasq**, as follows:*

```
edit> create  asign
edit> tab
```

wart #	rocket2	aoasq	m	asign
1	0.500000	-25.0000	0.100000	0.
2	0.620000	-9.00000	0.100000	0.
3	0.930000	0.	0.100000	0.
4	0.860000	4.00000	0.100000	0.
5	0.710000	16.0000	0.100000	0.
6	1.00000	-25.0000	0.900000	0.
7	1.26000	-9.00000	0.900000	0.
8	1.38000	0.	0.900000	0.
9	1.29000	4.00000	0.900000	0.
10	1.16000	16.0000	0.900000	0.

10 wart subsets listed.

```
edit> sign  1  aoasq  asign
```

10 data warts changed.

```
edit> tab
```

wart #	rocket2	aoasq	m	asign
1	0.500000	-25.0000	0.100000	-1.00000
2	0.620000	-9.00000	0.100000	-1.00000
3	0.930000	0.	0.100000	1.00000

4	0.860000	4.00000	0.100000	1.00000
5	0.710000	16.0000	0.100000	1.00000
6	1.00000	-25.0000	0.900000	-1.00000
7	1.26000	-9.00000	0.900000	-1.00000
8	1.38000	0.	0.900000	1.00000
9	1.29000	4.00000	0.900000	1.00000
10	1.16000	16.0000	0.900000	1.00000

10 wart subsets listed.

*Now, we can change **aoasq** into angle of attack values.*

edit> **abs aoasq**

4 data warts changed.

edit> **sqrt aoasq**

10 data warts changed.

edit> **tab**

wart #	rocket2	aoasq	m	asign
1	0.500000	5.00000	0.100000	-1.00000
2	0.620000	3.00000	0.100000	-1.00000
3	0.930000	0.	0.100000	1.00000
4	0.860000	2.00000	0.100000	1.00000
5	0.710000	4.00000	0.100000	1.00000
6	1.00000	5.00000	0.900000	-1.00000
7	1.26000	3.00000	0.900000	-1.00000
8	1.38000	0.	0.900000	1.00000
9	1.29000	2.00000	0.900000	1.00000
10	1.16000	4.00000	0.900000	1.00000

10 wart subsets listed.

*Now, put the sign back on **aoasq** which is really now the absolute value of the angle of attack.*

edit> **sign aoasq asign aoasq**

10 data warts changed.

edit> **tab**

wart #	rocket2	aoasq	m	asign
1	0.500000	-5.00000	0.100000	-1.00000
2	0.620000	-3.00000	0.100000	-1.00000
3	0.930000	0.	0.100000	1.00000
4	0.860000	2.00000	0.100000	1.00000
5	0.710000	4.00000	0.100000	1.00000

6	1.00000	-5.00000	0.900000	-1.00000
7	1.26000	-3.00000	0.900000	-1.00000
8	1.38000	0.	0.900000	1.00000
9	1.29000	2.00000	0.900000	1.00000
10	1.16000	4.00000	0.900000	1.00000

10 wart subsets listed.

edit> **del asign**

Let's put the right names on these variables:

edit> **rename aoasq alpha**
edit> **" m mach**
edit> **tab**

wart #	rocket2	alpha	mach
1	0.500000	-5.00000	0.100000
2	0.620000	-3.00000	0.100000
3	0.930000	0.	0.100000
4	0.860000	2.00000	0.100000
5	0.710000	4.00000	0.100000
6	1.00000	-5.00000	0.900000
7	1.26000	-3.00000	0.900000
8	1.38000	0.	0.900000
9	1.29000	2.00000	0.900000
10	1.16000	4.00000	0.900000

10 wart subsets listed.

*Our foundation table file, **toadnr1**, has the data listed as rocket1=f(mach,alpha), not rocket1=f(alpha,mach), so we need to fix this file to match **toadnr1**'s structure. This might prove to be a common problem, so let's define a macro, fix_mach_alpha, to fix it.*

edit> **macro fix_mach_alpha**
macro> **exch alpha mach**
macro> **tab**
macro> **sort alpha**
macro> **tab**
macro> **endmacro**
edit> **fix_mach_alpha**

[exch alpha mach]
[tab]

wart #	rocket2	mach	alpha
1	0.500000	0.100000	-5.00000
2	0.620000	0.100000	-3.00000
3	0.930000	0.100000	0.
4	0.860000	0.100000	2.00000

5	0.710000	0.100000	4.00000
6	1.00000	0.900000	-5.00000
7	1.26000	0.900000	-3.00000
8	1.38000	0.900000	0.
9	1.29000	0.900000	2.00000
10	1.16000	0.900000	4.00000

10 wart subsets listed.

[sort alpha]
[tab]

wart #	rocket2	mach	alpha
1	0.500000	0.100000	-5.00000
2	1.00000	0.900000	-5.00000
3	0.620000	0.100000	-3.00000
4	1.26000	0.900000	-3.00000
5	0.930000	0.100000	0.
6	1.38000	0.900000	0.
7	0.860000	0.100000	2.00000
8	1.29000	0.900000	2.00000
9	0.710000	0.100000	4.00000
10	1.16000	0.900000	4.00000

10 wart subsets listed.

*This file is now acceptable, so let's save the changes and write **rocket2** to a tadpole for later use.*

edit> **save toadnr2**

This request will overwrite the original contents of
an existing file. Do you really want it performed ?

> **y**

edit> **write rocket2 tad_rk2**

This request will overwrite the original contents of
an existing file. Do you really want it performed ?

> **y**

10 data warts written.

*Note: These questions appear only when **toadnr2** and **tad_rk2** already exist.*

The next file on the list is ***toadrk3***.

```
edit> open  toadrk3
edit> tab

          wart #      rocket3          m              q

            1         1800.00       0.100000       500.000
            2         2100.00       0.900000       500.000

        2 wart subsets listed.
```

File ***toadrk3*** *contains actual thrust values instead of thrust coefficients as* ***toadrk1*** *and* ***toadrk2*** *did.* ***rocket3*** *needs to be normalized by* ***q*** *in order to convert from actual thrust values to thrust coefficients. This too might prove to be a common problem, so let's define another macro, normalize, to fix it.*

```
edit> macro  normalize  $actual  $normalizer
macro> div  $actual  $normalizer
macro> tab
macro> del  $normalizer
macro> tab
macro> endmacro
edit> normalize  rocket3  q

        [ div rocket3 q ]

        2 data warts changed.

        [ tab ]

          wart #      rocket3          m              q

            1         3.60000       0.100000       500.000
            2         4.20000       0.900000       500.000

        2 wart subsets listed.

        [ del q ]
        [ tab ]

          wart #      rocket3          m

            1         3.60000       0.100000
            2         4.20000       0.900000

        2 wart subsets listed.
```

Let's fix the variable name:

```
edit> rename  m  mach
```

```
edit> tab

          wart #       rocket3        mach

            1          3.60000        0.100000
            2          4.20000        0.900000

        2 wart subsets listed.
```

*Since this data is not a function of alpha at all, it will be applied to all the coefficient values regardless of alpha. It needs to be duplicated four times to match the length of **toadnr1** and **tad_rk2**. Notice the use of **-1** to repeat the **after** command*

```
edit> save  toadnr3

        This request will overwrite the original contents of
        an existing file.  Do you really want it performed ?
> y
```

*Note: This question appears only when **toadnr3** already exists.*

```
edit> after  last  toadnr3
edit> -1
        [ after last toadnr3 ]
edit> -1
        [ after last toadnr3 ]
edit> -1
        [ after last toadnr3 ]
edit> tab

          wart #       rocket3        mach

            1          3.60000        0.100000
            2          4.20000        0.900000
            3          3.60000        0.100000
            4          4.20000        0.900000
            5          3.60000        0.100000
            6          4.20000        0.900000
            7          3.60000        0.100000
            8          4.20000        0.900000
            9          3.60000        0.100000
            10         4.20000        0.900000

        10 wart subsets listed.
```

This data is cycled just liked the two previous files, so we are finished with this one. Note we are using the same file name as earlier.

```
edit> save  toadnr3

        This request will overwrite the original contents of
        an existing file.  Do you really want it performed ?
> y
```

ORIGINAL PAGE IS OF POOR QUALITY

```
edit> write  rocket3  tad_rk3

        This request will overwrite the original contents of
        an existing file.  Do you really want it performed ?
> y

        10 data warts written.
```

*Note: These questions appear only when **toadnr3** and **tad_rk3** already exist.*

*We have completed all of the changes for the Mach < 1.0 data, so let's move on to the Mach => 1.0 data which is in **toadrk4**.*

```
edit> open  toadrk4
edit> tab
```

wart #	rocket4	aoa	q	m
1	375.000	-8.00000	500.000	1.00000
2	1000.00	-6.00000	500.000	1.00000
3	1500.00	0.	500.000	1.00000
4	750.000	5.00000	500.000	1.00000
5	125.000	10.0000	500.000	1.00000
6	165.000	-8.00000	500.000	2.50000
7	905.000	-6.00000	500.000	2.50000
8	1885.00	0.	500.000	2.50000
9	1760.00	5.00000	500.000	2.50000
10	965.000	10.0000	500.000	2.50000

```
        10 wart subsets listed.
```

*These actual thrust values need to be normalized as the **toadrk3** data was.*

```
edit> normalize  rocket4  q

        [ div rocket4 q ]

        10 data warts changed.

        [ tab ]
```

wart #	rocket4	aoa	q	m
1	0.750000	-8.00000	500.000	1.00000
2	2.00000	-6.00000	500.000	1.00000
3	3.00000	0.	500.000	1.00000
4	1.50000	5.00000	500.000	1.00000
5	0.250000	10.0000	500.000	1.00000
6	0.330000	-8.00000	500.000	2.50000
7	1.81000	-6.00000	500.000	2.50000
8	3.77000	0.	500.000	2.50000
9	3.52000	5.00000	500.000	2.50000

```
  10        1.93000     10.0000     500.000     2.50000

10 wart subsets listed.

[ del q ]
[ tab ]

 wart #     rocket4       aoa          m

   1       0.750000    -8.00000     1.00000
   2        2.00000    -6.00000     1.00000
   3        3.00000          0.     1.00000
   4        1.50000     5.00000     1.00000
   5       0.250000     10.0000     1.00000
   6       0.330000    -8.00000     2.50000
   7        1.81000    -6.00000     2.50000
   8        3.77000          0.     2.50000
   9        3.52000     5.00000     2.50000
  10        1.93000     10.0000     2.50000

10 wart subsets listed.
```

Let's fix some variable names.

```
edit> rename aoa alpha
edit> rename m mach
edit> tab

 wart #     rocket4      alpha        mach

   1       0.750000    -8.00000     1.00000
   2        2.00000    -6.00000     1.00000
   3        3.00000          0.     1.00000
   4        1.50000     5.00000     1.00000
   5       0.250000     10.0000     1.00000
   6       0.330000    -8.00000     2.50000
   7        1.81000    -6.00000     2.50000
   8        3.77000          0.     2.50000
   9        3.52000     5.00000     2.50000
  10        1.93000     10.0000     2.50000

10 wart subsets listed.
```

This data is a function of alpha and mach, not mach and alpha ...

```
edit> fix_mach_alpha

    [ exch alpha mach ]
    [ tab ]

 wart #     rocket4       mach        alpha

   1       0.750000     1.00000    -8.00000
```

2	2.00000	1.00000	-6.00000
3	3.00000	1.00000	0.
4	1.50000	1.00000	5.00000
5	0.250000	1.00000	10.0000
6	0.330000	2.50000	-8.00000
7	1.81000	2.50000	-6.00000
8	3.77000	2.50000	0.
9	3.52000	2.50000	5.00000
10	1.93000	2.50000	10.0000

10 wart subsets listed.

[sort alpha]
[tab]

wart #	rocket4	mach	alpha
1	0.750000	1.00000	-8.00000
2	0.330000	2.50000	-8.00000
3	2.00000	1.00000	-6.00000
4	1.81000	2.50000	-6.00000
5	3.00000	1.00000	0.
6	3.77000	2.50000	0.
7	1.50000	1.00000	5.00000
8	3.52000	2.50000	5.00000
9	0.250000	1.00000	10.0000
10	1.93000	2.50000	10.0000

10 wart subsets listed.

edit> **rename rocket4 tvc2t**
edit> **tab**

wart #	tvc2t	mach	alpha
1	0.750000	1.00000	-8.00000
2	0.330000	2.50000	-8.00000
3	2.00000	1.00000	-6.00000
4	1.81000	2.50000	-6.00000
5	3.00000	1.00000	0.
6	3.77000	2.50000	0.
7	1.50000	1.00000	5.00000
8	3.52000	2.50000	5.00000
9	0.250000	1.00000	10.0000
10	1.93000	2.50000	10.0000

10 wart subsets listed.

We are finished with this data now.

edit> **save toadtvc2**

This request will overwrite the original contents of

```
        an existing file.  Do you really want it performed ?
> y
```

*Note: This question appears only when **toadtv2** already exists.*

*Remember we are using **toadnr1** as our foundation file for our final file for the subsonic data. Now, let's build our final file from the ones we fixed earlier.*

```
edit> open  toadnrl
edit> tab

          wart #      rocket1        mach          alpha

            1         1.21338      0.100000      -5.00000
            2         1.70451      0.900000      -5.00000
            3         1.28079      0.100000      -3.00000
            4         1.73340      0.900000      -3.00000
            5         1.92600      0.100000            0.
            6         2.40750      0.900000            0.
            7         2.16675      0.100000       2.00000
            8         2.64825      0.900000       2.00000
            9         1.84896      0.100000       4.00000
           10         1.54080      0.900000       4.00000

        10 wart subsets listed.
```

*According to the researcher, the data from **toadnr2** and **toadnr3** are to be directly added to **rocket1**, so let's create one more macro, sum_data, to do it for us.*

```
edit> macro  sum_data  $rocket  $rocket_file
macro> create  $rocket
macro> tab
macro> read  $rocket  $rocket_file
macro> tab
macro> add  rocketl  $rocket
macro> tab
macro> delete  $rocket
macro> endmacro
edit> sum_data  rocket2  tad_rk2

        [ create rocket2 ]
        [ tab ]

          wart #      rocket1        mach          alpha          rocket2

            1         1.21338      0.100000      -5.00000             0.
            2         1.70451      0.900000      -5.00000             0.
            3         1.28079      0.100000      -3.00000             0.
            4         1.73340      0.900000      -3.00000             0.
            5         1.92600      0.100000            0.             0.
            6         2.40750      0.900000            0.             0.
            7         2.16675      0.100000       2.00000             0.
            8         2.64825      0.900000       2.00000             0.
```

9	1.84896	0.100000	4.00000	0.
10	1.54080	0.900000	4.00000	0.

10 wart subsets listed.

[read rocket2 tad_rk2]

10 data cells read.

[tab]

wart #	rocket1	mach	alpha	rocket2
1	1.21338	0.100000	-5.00000	0.500000
2	1.70451	0.900000	-5.00000	1.00000
3	1.28079	0.100000	-3.00000	0.620000
4	1.73340	0.900000	-3.00000	1.26000
5	1.92600	0.100000	0.	0.930000
6	2.40750	0.900000	0.	1.38000
7	2.16675	0.100000	2.00000	0.860000
8	2.64825	0.900000	2.00000	1.29000
9	1.84896	0.100000	4.00000	0.710000
10	1.54080	0.900000	4.00000	1.16000

10 wart subsets listed.

[add rocket1 rocket2]

10 data warts changed.

[tab]

wart #	rocket1	mach	alpha	rocket2
1	1.71338	0.100000	-5.00000	0.500000
2	2.70451	0.900000	-5.00000	1.00000
3	1.90079	0.100000	-3.00000	0.620000
4	2.99340	0.900000	-3.00000	1.26000
5	2.85600	0.100000	0.	0.930000
6	3.78750	0.900000	0.	1.38000
7	3.02675	0.100000	2.00000	0.860000
8	3.93825	0.900000	2.00000	1.29000
9	2.55896	0.100000	4.00000	0.710000
10	2.70080	0.900000	4.00000	1.16000

10 wart subsets listed.

[delete rocket2]

edit> **sum_data rocket3 tad_rk3**

[create rocket3]

[tab]

wart #	rocket1	mach	alpha	rocket3
1	1.71338	0.100000	-5.00000	0.
2	2.70451	0.900000	-5.00000	0.
3	1.90079	0.100000	-3.00000	0.
4	2.99340	0.900000	-3.00000	0.
5	2.85600	0.100000	0.	0.
6	3.78750	0.900000	0.	0.
7	3.02675	0.100000	2.00000	0.
8	3.93825	0.900000	2.00000	0.
9	2.55896	0.100000	4.00000	0.
10	2.70080	0.900000	4.00000	0.

10 wart subsets listed.

[read rocket3 tad_rk3]

10 data cells read.

[tab]

wart #	rocket1	mach	alpha	rocket3
1	1.71338	0.100000	-5.00000	3.60000
2	2.70451	0.900000	-5.00000	4.20000
3	1.90079	0.100000	-3.00000	3.60000
4	2.99340	0.900000	-3.00000	4.20000
5	2.85600	0.100000	0.	3.60000
6	3.78750	0.900000	0.	4.20000
7	3.02675	0.100000	2.00000	3.60000
8	3.93825	0.900000	2.00000	4.20000
9	2.55896	0.100000	4.00000	3.60000
10	2.70080	0.900000	4.00000	4.20000

10 wart subsets listed.

[add rocket1 rocket3]

10 data warts changed.

[tab]

wart #	rocket1	mach	alpha	rocket3
1	5.31338	0.100000	-5.00000	3.60000
2	6.90451	0.900000	-5.00000	4.20000
3	5.50079	0.100000	-3.00000	3.60000
4	7.19340	0.900000	-3.00000	4.20000
5	6.45600	0.100000	0.	3.60000
6	7.98750	0.900000	0.	4.20000
7	6.62675	0.100000	2.00000	3.60000

8	8.13825	0.900000	2.00000	4.20000
9	6.15896	0.100000	4.00000	3.60000
10	6.90080	0.900000	4.00000	4.20000

```
10 wart subsets listed.

[ delete rocket3 ]
```

We have combined all of the subsonic data together, except for a scale factor that the researcher provided. First, let's rename ***rocket1****.*

```
edit> rename  rocket1  tvclt
edit> tab
```

wart #	tvclt	mach	alpha
1	5.31338	0.100000	-5.00000
2	6.90451	0.900000	-5.00000
3	5.50079	0.100000	-3.00000
4	7.19340	0.900000	-3.00000
5	6.45600	0.100000	0.
6	7.98750	0.900000	0.
7	6.62675	0.100000	2.00000
8	8.13825	0.900000	2.00000
9	6.15896	0.100000	4.00000
10	6.90080	0.900000	4.00000

```
10 wart subsets listed.
```

Now, let's apply the scale factor, do one last tabulation, and save the file.

```
edit> mult  tvclt  .264

     10 data warts changed.

edit> tab
```

wart #	tvclt	mach	alpha
1	1.40273	0.100000	-5.00000
2	1.82279	0.900000	-5.00000
3	1.45221	0.100000	-3.00000
4	1.89906	0.900000	-3.00000
5	1.70438	0.100000	0.
6	2.10870	0.900000	0.
7	1.74946	0.100000	2.00000
8	2.14850	0.900000	2.00000
9	1.62597	0.100000	4.00000
10	1.82181	0.900000	4.00000

```
10 wart subsets listed.
```

```
edit> save  toadtvc1

         This request will overwrite the original contents of
         an existing file.  Do you really want it performed ?
> y
```

*Note: This question appears only when **toadtvc1** already exists.*

```
edit> q

         Normal session.

%
```

For the reader's benefit, all of these TOAD files, including those created during this session, are available from the Langley Mustang directory

```
~ntflib/toad_examples
```

Appendix B
The TOAD Format (summarized)

The Transferable Output ASCII Data (TOAD) format was developed by Computer Sciences Corporation for NASA Langley Research Center as a uniform way to store and retrieve tabulated data. A full discussion of the TOAD format is presented in NASA Contractor Report 178361. However, most readers will find the following abbreviated description adequate for their purposes.

TOAD files are sequential-access, formatted, and use fixed-length records of 80 characters. This file type makes them simple to edit, write to or read from magnetic media, or send across communications networks. Unfortunately, these same characteristics make them large compared to their unformatted, variable record-length counterparts. Therefore, we recommend that TOAD files be used only when relatively small amounts of data are to be retained (less than 5000 pieces of data), or when any amount of data must be transferred from one computer to another (usually different) computer via magnetic media or a communications network.

Blocks of information within a TOAD file are called "warts." Each wart has its own purpose, and may use one or more records. For example, consider the abbreviated TOAD file below:

```
BEGIN
SKIP Predicted aerodynamic properties of a modified F-4D fighter
COUNT              9
LABELMACH          ALPHA          2Y/B           CL-V           CD-V
     CM-V          CL-Z           CD-Z           CM-Z
DATA    .85000000E+00  .10000000E+01  .70800000E+00  .97261000E+00  .15166000E+00
      -.24139000E+00  .88951000E+00  .11640000E+00 -.24754000E+00
DATA    .85000000E+00  .10000000E+01  .79200000E+00  .89415000E+00  .11423000E+00
      -.27911000E+00  .78920000E+00  .69700000E-01 -.27105000E+00
DATA    .85000000E+00  .10000000E+01  .87500000E+00  .78330000E+00  .72870000E-01
      -.29796000E+00  .65651000E+00  .19080000E-01 -.26920000E+00
END
```

Notice that the file begins with a BEGIN wart and ends with an END wart. The SKIP wart is used to insert comments inside the file. The COUNT wart indicates that there are 9 variables in this TOAD file. The LABEL wart assigns a 15-character name with each of these variables. Each DATA wart contains information gathered at some common event. For example, the second DATA wart indicates that at Mach .85, 10 degrees angle of attack, and at 79.2% semispan the full vortex flow coefficients of lift, drag and moment (C_l, C_d and C_m) are .89415, .11423 and -.27911, respectively, while the zero leading-edge suction coefficients of lift, drag and moment are .7892, .0697 and -.27105, respectively.

The FORTRAN 77 edit descriptors for each type of wart are:

Wart Type	Write Format	Read Format
SKIP	'SKIP ',A75	T6,A75
COUNT	'COUNT',I15	T6,I15
LABEL	'LABEL',(5A15)	(T6,5A15)
DATA	'DATA ',(5E15.8)	(T6,5E15.8)

The following rules must always be observed when creating and using TOAD files:

1. Exactly one BEGIN wart must appear in the TOAD file, and it must be the very first record.
2. Exactly one END wart must appear in the TOAD file, and it must be the very last record.
3. A COUNT wart must appear before any LABEL or DATA warts.
4. No wart may come between two records within another multi-record wart.
5. SKIP warts may appear anywhere in the TOAD file, subject to condition 4.
6. Multiple DATA warts are expected. All DATA warts must contain the same amount of data and use the same number of records.
7. There is no limit on the number of warts or records in a TOAD file.

NASA
National Aeronautics and Space Administration

Report Documentation Page

1. Report No.	2. Government Accession No.	3. Recipient's Catalog No.
NASA CR-187507		

4. Title and Subtitle	5. Report Date
Transferable Output ASCII Data (TOAD) Editor Version 1.0 User's Guide	February 1991
	6. Performing Organization Code
7. Author(s)	8. Performing Organization Report No.
Bradford D. Bingel Anne L. Shea Alicia S. Hofler	
	10. Work Unit No.
	505-59-10-03
9. Performing Organization Name and Address	11. Contract or Grant No.
Computer Sciences Corporation Applied Technology Division Hampton, VA 23666-1379	NAS1-19038
	13. Type of Report and Period Covered
12. Sponsoring Agency Name and Address	Contractor Report
National Aeronautics and Space Administration Langley Research Center Hampton, VA 23665-5225	14. Sponsoring Agency Code

15. Supplementary Notes

Langley Technical Monitor: Dr. John E. Lamar

16. Abstract

The Transferable Output ASCII Data (TOAD) Editor is an interactive software tool for manipulating the contents of TOAD files. The TOAD Editor is specifically designed to work with tabular data. Selected subsets of data may be displayed to the user's screen, sorted, exchanged, duplicated, removed, replaced, inserted, or tranferred to and from external files. It also offers a number of useful features including on-line help, macros, a command history, an "undo" option, variables, and a full compliment of mathematical functions and conversion factors. Written in ANSI FORTRAN 77 and completely self-contained, the TOAD Editor is very portable and has already been installed on SUN, SGI/IRIS, and CONVEX hosts.

17. Key Words (Suggested by Author(s))	18. Distribution Statement
Computer Programs Software Tools Data Management Data Manipulation Data Storage	Unclassified - Unlimited Subject Category 61

19. Security Classif. (of this report)	20. Security Classif. (of this page)	21. No. of pages	22. Price
Unclassified	Unclassified	147	A07