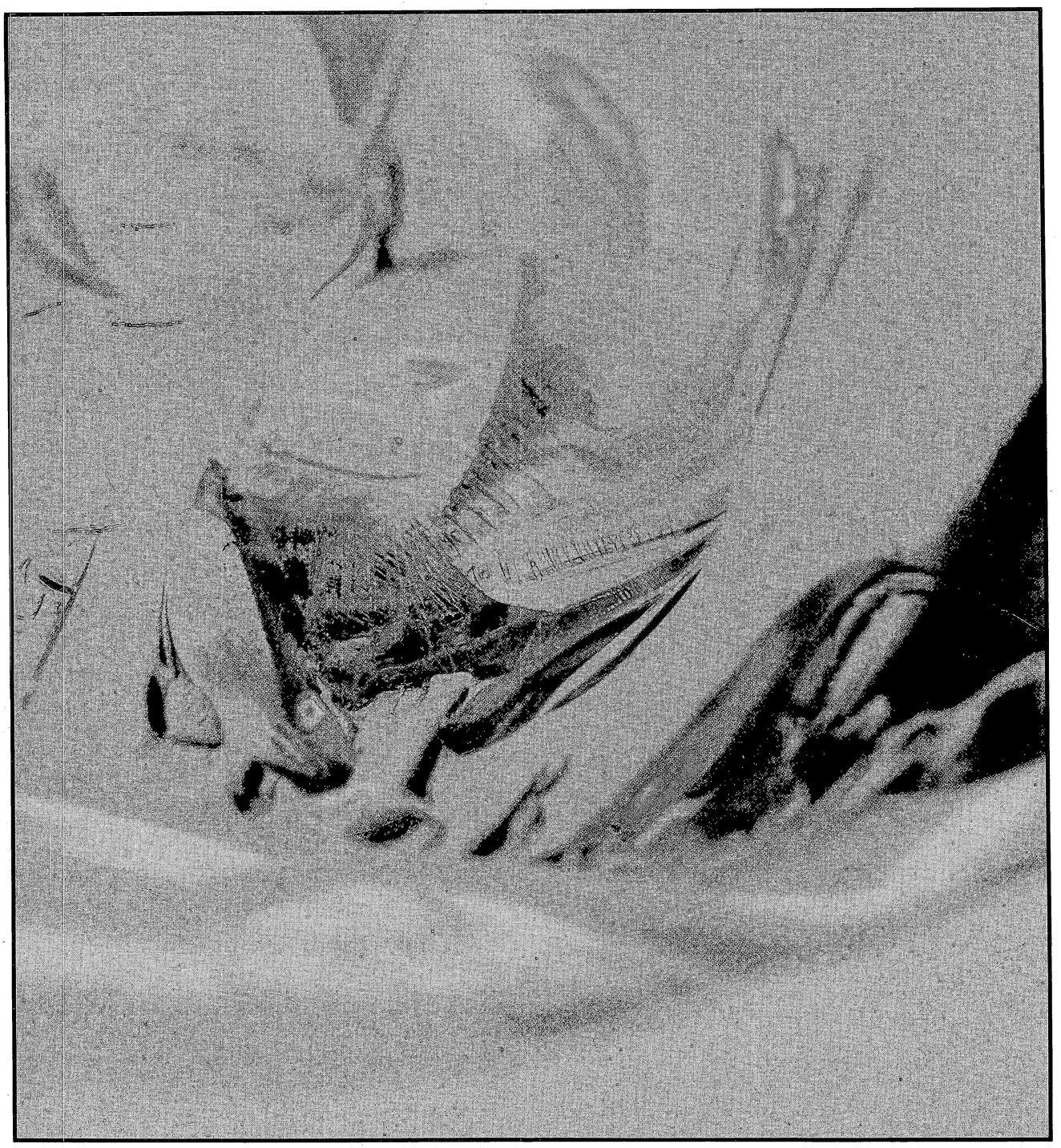
MP/AOS

Advanced Program Development Utilities



DataGeneral

MP/AOS
Advanced Program Development Utilities

Data General

Notice

Data General Corporation (DGC) has prepared this document for use by DGC personnel, customers, and prospective customers. The information contained herein shall not be reproduced in whole or in part without DGC's prior written approval.

DGC reserves the right to make changes in specifications and other information contained in this document without prior notice, and the reader should in all cases consult DGC to determine whether any such changes have been made.

THE TERMS AND CONDITIONS GOVERNING THE SALE OF DGC HARDWARE PRODUCTS AND THE LICENSING OF DGC SOFTWARE CONSIST SOLELY OF THOSE SET FORTH IN THE WRITTEN CONTRACTS BETWEEN DGC AND ITS CUSTOMERS. NO REPRESENTATION OR OTHER AFFIRMATION OF FACT CONTAINED IN THIS DOCUMENT INCLUDING BUT NOT LIMITED TO STATEMENTS REGARDING CAPACITY, RESPONSE-TIME PERFORMANCE, SUITABILITY FOR USE OR PERFORMANCE OF PRODUCTS DESCRIBED HEREIN SHALL BE DEEMED TO BE A WARRANTY BY DGC FOR ANY PURPOSE, OR GIVE RISE TO ANY LIABILITY OF DGC WHATSOEVER.

IN NO EVENT SHALL DGC BE LIABLE FOR ANY INCIDENTAL, INDIRECT, SPECIAL, OR CONSEQUENTIAL DAMAGES WHATSOEVER (INCLUDING BUT NOT LIMITED TO LOST PROFITS) ARISING OUT OF OR RELATED TO THIS DOCUMENT OR THE INFORMATION CONTAINED IN IT, EVEN IF DGC HAS BEEN ADVISED, KNEW OR SHOULD HAVE KNOWN OF THE POSSIBILITY OF SUCH DAMAGES.

Ordering No. 069-400208

Rev. 00, July 1982

Preface

This manual describes two utilities: Text Control System (TCS) and BUILD both of which are useful for maintaining order among related files. Although their usefulness is not restricted to program development environments, these utilities have been specifically designed to address the needs of program development teams who design and maintain multi-version, multi-module products. The third utilitity described in this manual, FIND, allows you to conveniently scan text files for the occurrence of a pattern.

TCS maintains several versions of a single file; BUILD maintains *products.* A product may be a program, a collection of programs, a library, a document, or any file or collection of files. TCS and BUILD may be used separately, but they have been designed to work together and each utility brings out the full power of the other. TCS does, however, logically precede BUILD, and you should become familiar with TCS before using Build.

TCS allows you to keep order among files that are versions of a single file. TCS records the date and nature of all modifications to a file, and in so doing reduces the amount of space required to store multiple versions. Familiarity with the CLI and a text editor are the pre-requisites for TCS use.

The BUILD utility speeds and simplifies the construction of products through the transformation of files. For example, BUILD can speed the creation of a program file through the compilation, assembly, and binding of source files. BUILD maintains a record of the dependencies among files that go into a product. Then, when updating a product, BUILD inspects all the input files and determines what actions to perform to make the product up to date without doing unnecessary work.

The FIND program locates and calls out character patterns in text files. FIND uses a range of pattern-matching templates for ease and versatility. These templates are also employed by the SLATE text editor *(MP/AOS Slate Text Editor*, DGC No. 069-400209).

Related Manuals

The following manuals also belong to the series of books published on the MP/AOS operating system.

MP/AOS Concepts and Facilities (DGC No. 069-400200) provides a concise but thorough introduction to the MP/AOS operating system for users who want to assess the system's advantages.

MP/AOS System Programmer's Reference (DGC No. 093-400051) documents MP/AOS system structure and provides a complete dictionary of system calls and library routines.

MP/AOS Command Line Interpreter (CLI) (DGC No.069-400201) describes the interactive CLI program, the user's primary interface to the MP/AOS system. A command dictionary provides command descriptions, formats, and examples.

Loading MP/AOS (DGC No. 069-400207) describes how to install MP/AOS software on ECLIPSE-line computers and how to load tailored systems.

MP/AOS System Generation and Related Utilities (DGC No. 069-400206) describes the generation of an MP/AOS system tailored to specific applications. It also describes the following utilities, including sample dialogues as appropriate:

- SYSGEN, the interactive system generation utility;
- DINIT, the disk initializer;
- FIXUP, the disk repair utility;
- SPOOLER, which controls line printer operations;
- ELOG (error logger), the utility for interpreting the system log file.

MP/AOS Debugger and Performance Monitoring Utilities (DGC No. 069-400205) describes the following utilities, providing a dictionary of debugger commands and sample dialogues as appropriate:

- FLIT, the process debugger;
- PROFILE, which measures execution-time performance;
- OPM, the process monitor that displays current system resource allocation and status.

MP/AOS Macroassembler, Binder, and Library Utilities (DGC 069-400210) documents the MP/AOS macroassembler and binder as well as the library file editor (LED) and system cross-reference analyzer (SCAN). It includes programming examples and a dictionary of assembler pseudo-ops.

MP/AOS SPEED Text Editor (DGC No. 069-400202) documents the features of SPEED, the MP/AOS character-oriented text editor.

MP/AOS SLATE Text Editor (DGC 069-400209) documents the features of SLATE, a screen- and line-oriented text editor.

MP/AOS File Utilities (DGC No. 069-400204) describes the following utility programs, providing sample dialogues for each:

- FEDIT, a file editor that permits modification of system files, including program and data files;
- FDISP, which can display the address and data contents of a file or compare two files, displaying the parts that differ;
- SCMP, which can compare two source programs line by line;
- MOVE, which allows the transfer of files among directories;
- AOSMIC, which allows manipulation of MP/AOS and MP/OS disks and files on an AOS system;
- FOXFIRE, which permits the transfer of files among MP/OS, MP/AOS, and AOS systems over asynchronous communication lines.

SP/Pascal Programmer's Reference (DGC No. 069-400203) documents an extended Pascal for system programmers. SP/Pascal has all of the features of MP/Pascal as well as special features targeted for the MP/AOS and AOS operating systems.

Books on three additional programming languages supported by MP/AOS have previously been published as part of the bookset for the MP/OS operating system:

MP/Pascal Programmer's Reference (DGC No. 069-400031) documents for system programmers a Pascal-based language targeted for the MP/OS operating system.

MP/FORTRAN IV Programmer's Reference (DGC No. 069-400033) documents for system programmers a language based on ANSI 1966 standard FORTRAN with extensions.

MP/Basic Programmer's Reference (DGC No. 069-400032) documents for new users a programming language based on ANSI standard Basic with extensions.

MP/OS

For information on Microproducts and a bibliography of documentation on the Microproducts line, see *Introduction to Microproducts,* DGC No. 014-000685.

For information on cross development between MP/OS and MP/AOS, see *MP/OS System Programmer's Reference,* DGC No. 093-400001.

Contents

Figures

Tables

Part 1
Text Control System

Introduction to TCS

1

Context of the Text Control System

Imagine you have a standard letter or memorandum that you regularly use in slightly different versions. You may from time to time wish to look up which version of the letter you used on a given occasion. You may also wish to know when that version was written and by whom. The Text Control System (TCS) provides a mechanism for efficiently keeping track of this information.

Similarly, when a program which had been working suddenly develops a bug, the first question a programmer asks is, "What changed?". The next questions usually are, "When did it change?", "Who changed it?" and "Why was it changed?". TCS provides a mechanism for answering all of these questions.

In addition, TCS enables you to mark each version of the file with one or more revision numbers, which you can use to co-ordinate versions of different files that are used in a single product.

You may use TCS to manage any file that exists in multiple versions. The only requirements are that the file be text (ASCII), and that the file end with a data-sensitive delimiter. Data-sensitive delimiters are New-line (), Carriage Return (CR), Formfeed (CTRL L), and null. To ensure that your text file ends with one of these, end editing by positioning the cursor at the end of the file and pressing New-line.

Using one file, called the *master file*, TCS records the history of a text file through successive edits. The master file contains the encoded information necessary to recreate any version of the file; for your reference it also contains a log of changes made from one version of a file to the next. It records who made the changes, when the changes were made, and comments entered at the time.

Thus TCS serves three functions:

- It provides an unambiguous history of all versions of the file.
- It conserves disk storage space.
- It provides a mechanism for co-ordinating versions of different files.

Overview of Functions

Use TCS by typing the TCS commands in response to the CLI prompt. Each command causes execution of the TCS program, which may, in turn, prompt you for a response. When the action specified by the command is completed, TCS terminates and the CLI resumes. The TCS commands are briefly described below and fully explained in Chapter 2. Table 1.1 lists the TCS commands and their functions.

Command	Function
ACCESS	Creates a copy of a version in the working directory
CHECKIN	Adds new version to master file
CHECKOUT	Obtains a copy of latest version for modification
NEWBASIS	Creates new branch on master file
NEWTCS	Creates new master file
REVMARK	Sets revision number on a version.
TCSPRINT	Obtains information about the creation of any version(s).

Table 1.1 TCS commands

Each of the TCS commands is a CLI macro. In order for the commands to work, the files TCS.PR, TCS.OL and SCMP.PR must be on your searchlist. If these files are not on your searchlist, modify each of the TCS macros by inserting the appropriate SEARCHLIST command. Refer to the release notice supplied with your software for further information about installing TCS on your system. The TCS on-line HELP facility is also described in the release notice.

NOTE: *Under Data General's AOS and AOS/VS operating systems screenedit is supported on all interactive TCS commands.*

Figure 1.1 illustrates the master file as a sequence of versions of a file, together with information about each version. This illustration does not represent the true internal structure of the master file. It only shows the types of information contained in the file and the sequential relationship of different versions. Other illustrations in this chapter and the next show the relationship of TCS commands to the master file.

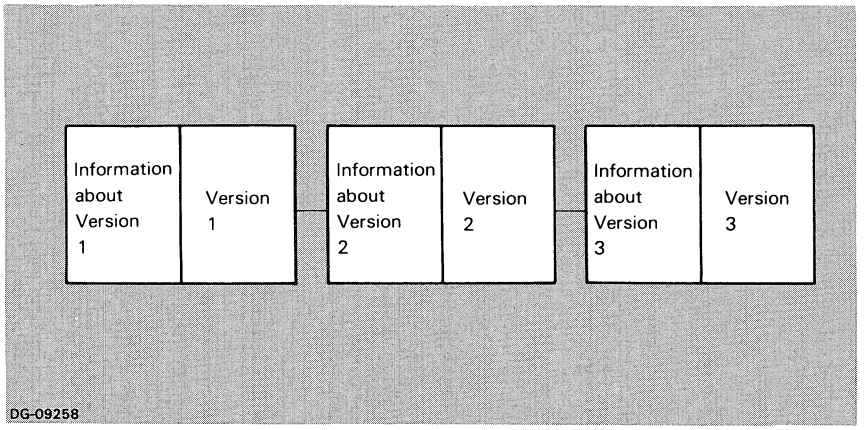


Figure 1.1 TCS master file

After you have created a text file using a text editor of your choice, you place the file under TCS control using the NEWTCS procedure. TCS uses the file to create the master file, which has the filename extension .TCS. For example, if you place a file named SOURCE under TCS, TCS creates a master file called SOURCE.TCS.

NOTE: *The TCS master file is printable, and can be examined. You do not need to understand or even look at this file in order to use TCS. Never modify the file. TCS takes care of that.*

With the CHECKOUT command you can obtain a copy of the last version of the file in order to edit that version. After you have edited the file, you use the CHECKIN command to add the edited version to the master file. This version is now the latest one in the master file, and will be the next version checked out.

When you check a version out, TCS marks the master file (and not the file checked out). The file remains checked out until you check it back in, whether you log off or the system shuts down in the meantime. While the file is checked out, TCS ensures that no one else can check out the same version. The checked-out version must be checked in by the same person who checked it out. In this way TCS prevents two people from making independent, simultaneous changes to the same file.

With the ACCESS command you may create a copy of any version. Like the CHECKOUT command, the ACCESS command creates a copy of a version in the current working directory, with the original file name. Unlike the CHECKOUT command, the ACCESS command does not mark the master file, and you may not use ACCESSed files to create new versions for incorporation into the master file.

NOTE: *You cannot check a version in unless a version has been checked out. If you check out a file named JOE and then accidentally delete it, you may check in any file named JOE, regardless of how the new JOE was created. You may obtain a copy of the deleted version using the ACCESS command, and check that version in. In general, however, ACCESS is not used for files to be checked back in.*

Table 1.2 summarizes some of the attributes of the ACCESS and CHECKOUT commands.

Command	Mark master file?	Require /USER?	Last cycle in basis only?	Create copy in working directory?
ACCESS	No	No	No	Yes
CHECKOUT	Yes	Yes	Yes	Yes

Table 1.2 Comparison of ACCESS and CHECKOUT

With the TCSPRINT command you may print any or all of the information about successive versions.

Figure 1.2 shows the uses of the NEWTCS, CHECKOUT, CHECKIN, ACCESS, and TCSPRINT commands. For many applications, these are the only commands you will need.

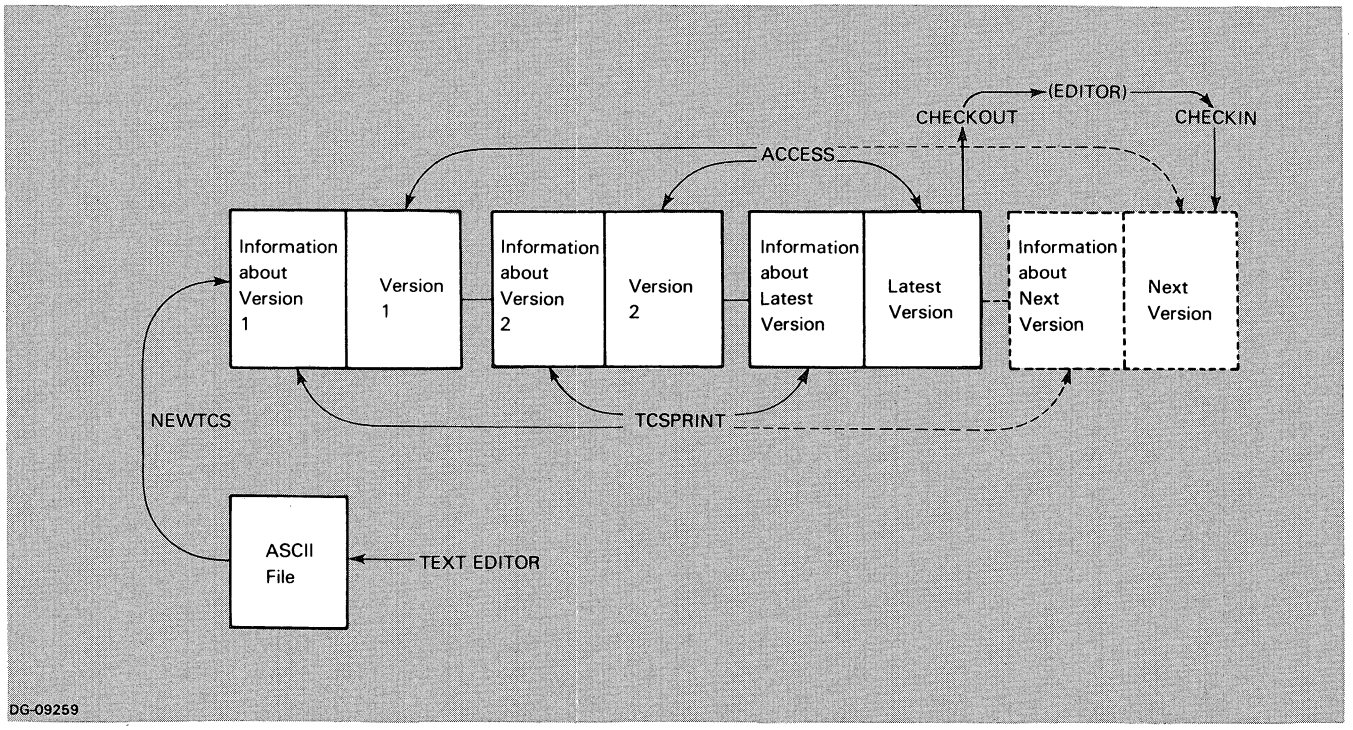


Figure 1.2 TCS commands

With the NEWBASIS command you may select a version other than the last one as starting point, or *basis*, for the CHECKOUT -(edit)- CHECKIN cycle.

Figure 1.3 illustrates a master file with two bases.

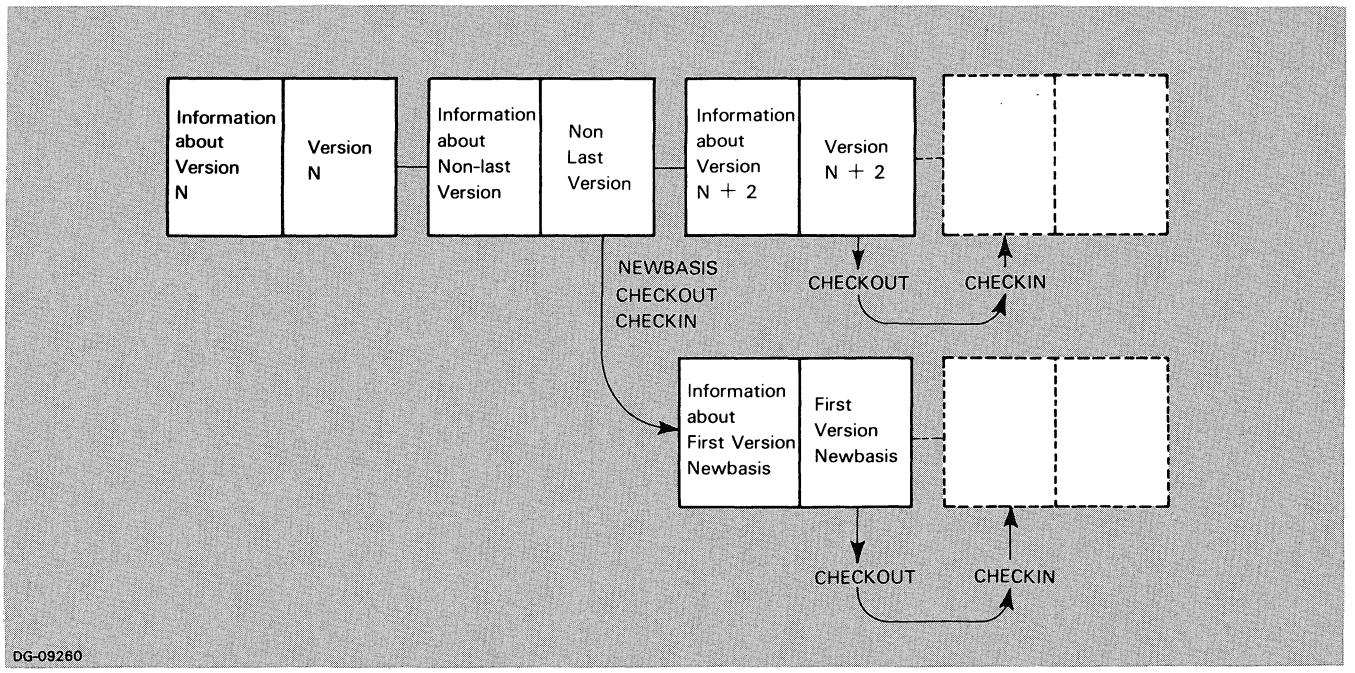


Figure 1.3 Master file with two bases

With the REVMARK command you may assign one or more product revision numbers to each version of the file. The revision number is analogous to, but distinct from, the program revision number that you can set with the CLI REVISION command.

See the discussion of revision marks in the section of this chapter entitled "Identifing Versions" for further explanation and an example.

Keeping Track of Changes

In order to note changes made to the master file, TCS assigns an identifier to each version and associates each version with the name of the person who created it.

Identifying Versions

Each version in the master file is uniquely identified by its basis name and cycle number, which are explained below. In addition, each file may be assigned one or more revision numbers. When using TCS commands, you specify the version to be acted on by appending a *version identifier* (VID) to the filename. The VID contains either the basis name and cycle number or the revision number. Different formats of the VID are described below.

The Basis Name

A basis is a branch in the master file created by using the NEWBASIS command; the basis name is assigned during the NEWBASIS procedure. The default basis is the original branch of the master file; its name is the null string. The basis name may be up to ten characters long, may include letters, digits, underscores and question marks, and may *not* start with a digit.

The Cycle Number

The cycle number represents the number of times the CHECKOUT (edit) CHECKIN cycle has been performed since the basis was created. The original version is cycle 1, the first version checked in is cycle 2, etc.

The Revison Number

TCS can mark each version in the master file with one or more revision numbers, which you can use to co-ordinate versions of different files that are used to make the same product. Since each version can have more than one revision number, you can use the same version of a file to create different versions of a product.

The format of the revision number is *maj.min* where *maj* and *min* are integers in the range 0 to 255, and represent the major and minor revision, respectively.

For example, consider a product called SET, that consists of one version each of file A, file B, and file C. Files A, B, and C are all maintained under TCS. Assume SET exists in three revisions -- 1.00, 2.00, and 3.00.

This relationship is illustrated in Figure 1.4. In the figure, cycles of files A, B, and C are represented as adjacent boxes. Cycle numbers are indicated by decimals -- cycle 1 is .1, etc. Revision numbers are highlighted. For example, in file A, cycle 1 is revision 1.00, cycle 2 is revision 2.00, cycle 4 is revision 3.00, and cycle 3 does not have a revision number. In file C, cycle 1 is revision number 1.00,2.00, and 3.00.

Revision 1.00 of SET contains A cycle 1, B cycle 1, C cycle 1.
Revision 2.00 of SET contains A cycle 2, B cycle 2, C cycle 1.
Revision 3.00 of SET contains A cycle 4, B cycle 3, C cycle 1.

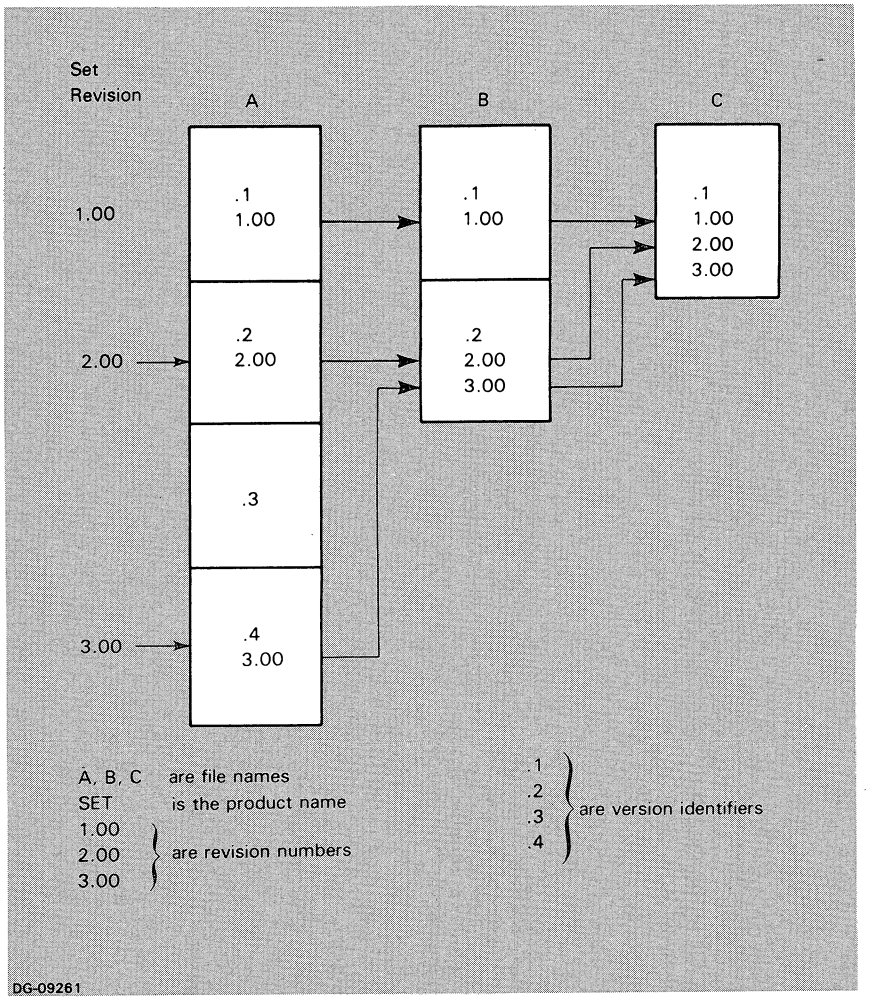


Figure 1.4 Revision number

Format of the Version Identifier

The version identifier can take any of several forms:

If no VID is specified, the default version is the highest-numbered cycle in the default basis.

The format

/<basis>

specifies the highest cycle in <basis>.

The format

/.<cycle>

specifies version number <cycle> of the default basis.

The format

/<basis>.<cycle>

specifies version number <cycle> in the basis <basis>.

When the global /REV switch is used on the TCS command, the format

/<maj>.<min>

specifies the version with revision number <maj>.<min>.

The VID is an identifier used internally by TCS and is not a filename extension. No matter which version you specify, the version created by ACCESS or CHECKOUT has the original filename.

Identifying the User

In order to associate changes made to files with the people who made them, TCS requires that you supply a *username* whenever you modify a file. To do so, you use the USER=<*username*> switch on the CHECKOUT, CHECKIN, NEWTCS, and NEWBASIS commands. The username consists of from 1 to 15 alphanumeric characters. This switch is not optional.

NOTE: *When used under Data General's AOS or AOS/VS operating system, TCS associates file modifications with the username of the person making the changes. Under these systems, the /USER switch is not used.*

TCS Commands

2

This Chapter describes the seven TCS commands and contains short examples of their use.

The first three commands described are NEWTCS, CHECKOUT, and CHECKIN. You need these three to create and modify the master file.

ACCESS and TCSPRINT, two of the most commonly used commands, are described next. The chapter concludes with a discussion of the NEWBASIS and REVMARK commands.

Each of these commands can take multiple filename arguments, although for the sake of clarity the examples show the use of single file arguments. Longer examples are contained Chapter 3.

Initializing the Master File

To create a master file from a text file, use the NEWTCS command.

NEWTCS

The format of this command is

) NEWTCS/user=*username*[/ND]*filename*. . .

where

Username is your name or the name you wish to be recorded as the creator of the file.

ND is the non-deletion switch described below.

filename is the name of the file to be placed under TCS. The file may be any file that contains only ASCII data and ends with a data-sensitive delimiter.

When you enter this command, TCS displays the message

```
Describe <filename>
))
```

The two right parentheses are the prompt.

Enter a description of the file. To end the description and return to the CLI, type a third right parenthesis followed by a new line in response to the)) prompt.

The description that you enter is incorporated in the TCS file and is retrievable by using the TCSPRINT command.

NEWTCS causes a new file named <*filename*>.TCS to be created in your working diretory. Unless you use the /ND switch, <*filename*> is deleted.

When you use the /ND switch, your original file remains unchanged in the working directory.

Creating New Versions

Once you have created the master file using the NEWTCS command, you create new versions of the file using the CHECKOUT/CHECKIN cycle. (If you wish to create a copy of the file for use without modification -- for example, for assembling or executing -- use the ACCESS command).

CHECKOUT

The CHECKOUT command creates a copy of the latest version in one of the bases of the file that you wish to modify. The master file is marked to indicate that this file is checked out; no other user may check out the same version.

The version checked out is the latest version of the default basis unless you specify another basis with the /basis switch.

The format of the command is

)CHECKOUT/USER=<*username*>[/*sw*] *filename*[/*basis*] <. . .>

where

/USER is the user identification switch described in Chapter 1.

sw is one of the function switches listed in Table 2.1

filename is the version you wish to check out (may only be most recent version of any basis).

basis is the basis name. If you are checking out a version of the default basis, you do not need to use this switch. (The default basis is the original stem of the master file).

Switch	Action
/O	Overwrites an existing copy, if any, of *filename* in the working directory. If you do not use this switch and there is a copy of the file in the working directory, the file will not be checked out.
/A	Aborts on errors instead of issuing a warning and continuing.
/S	Suppresses standard output that informs user of actions performed by TCS.
/REV	Causes VID to be interpreted as a revision number.
/V=<int>	If int is greater than 1, TCS informs you if changes have been made to another basis while the file was checked out. See section entitled "Changes to Other Bases."

Table 2.1 CHECKOUT function switches

Once you have checked the file out, make whatever changes are necessary. The checking-out marks the master file, not the file you have checked out. The file remains checked out until you check it in, regardless of how many times you edit the file or log off the system. When you have made a version that you wish to record in the master file, use the CHECKIN command described below.

NOTE: *If you accidentally delete a checked out version, use the ACCESS command to obtain a copy to check in.*

CHECKIN

The CHECKIN command incorporates into the master file the revised version of a checked-out file, and prompts you for comments about changes made to the file.

The format of the command is

) CHECKIN/USER=<*username*>[/*sw*] *filename*[/*basis*] <. . .>

where

username is the name used when the file was checked out. See discussion in Chapter 1.

sw is one of the function switches listed in Table 2.2

filename is the name of a checked-out version of the file

/basis is the basis identification switch, which you must use if modifying a file from a basis other than the default.

If nothing has been changed since the file was checked out, TCS does not create a new version.

Switch	Action
/ND	Does not delete the copy of the checked out file from the user's working directory. If this switch is not used, the file is automatically deleted after a successful checkin.
/A	Aborts on errors instead of issuing a warning and continuing.
/S	Suppresses standard output that informs user of actions performed by TCS.
/REV	Causes VID to be interpreted as a revision number.

Table 2.2 CHECKIN function switches

Changes to Other Basis

When you use the /V switch, TCS informs you if changes were made to any basis other than the one you are checking out since this basis was last checked out. In other words, if you are now checking out the last version in basis X, TCS informs you if changes have been made to any basis other than X since the last time that basis X was checked out.

This switch can help you maintain "fixed" and "developmental" versions of a file. For example, you might use the default basis to record the fixed or "official" version of a file. Other bases would be used for experimentation. Thus, if you were checking out the latest version of an experimental basis, TCS would inform you if the official version had been changed.

The ACCESS Command

Use the ACCESS command whenever you wish to use a version of a file without creating a new version to be checked in. This command creates a copy of the version you specify in the working directory.

The format of this command is

) ACCESS[/*switches*] *filename*[/VID] <. . .>

where

/switches is one of the switches listed in Table 2.3

filename is the (original) name of the file

/VID is the version identifier

Switch	Action
/O	Overwrites an existing copy, if any of filename in the working directory. If you do not use this switch and there is a copy of the file in the working directory, the file will not be accessed.
/A	Abort on errors instead of issuing a warning and continuing.
/REV	Causes VID to be interpreted as a revision number.
/S	Suppress standard output that informs user of actions performed by TCS.
/V=<int>	If integer is greater than 1, TCS informs you if changes have been made to another basis since the file was last checked out. See preceding section, "Changes to Other Bases."

Table 2.3 ACCESS function switches

The TCSPRINT Command

To obtain information from the TCS master file, use the TCSPRINT command.

The format of the command is

) TCSPRINT[/*switches*] *filename*[/*version*] <. . .>

where

/*switches* is one or more of the switches listed in Table 2.4.

filename is the name of the original file.

/VID is the version identifier.

Switch	Function
/L=*file*	Specifies the listing file. If the file does not exist, TCS creates it. If it does exist, it is appended to.
/L	Redirects output to @LPT.
/V=<*num*>	when num = 0 If the file is checked out, prints which version is checked out, when it was checked out, and who checked it out. If the file is not checked out, prints nothing. 1 Prints list of all versions, who created each, and dates of creation. 2 In addition to (1), prints comments added at each checkin. 3 In addition to (2), prints comments describing each basis, including the default (original file description), and revision information.
/body	Prints the text of the module indicating what was changed at the last checkout, and when. Also prints comment associated with this version. (When you use the /BODY switch the /V switch has no effect).
/rev	Interprets VID field as revision number.
/after=date	Specifies date for determining which file modification information to print. TCS prints only modifications which have occurred after date.

Table 2.4 TCSPRINT switches

The following example shows the use of the TCSPRINT command to determine which files, if any, are checked out. Other examples of the TCSPRINT command appear in Chapter 3.

Determining Which Versions Are Checked Out

Since all versions created by ACCESS and CHECKOUT receive the same filename, you cannot tell by looking at the list of files in the working directory which versions, if any, are checked out. To determine if a file is checked out, use the command

)TCSPRINT/v=0 <*filename*>

If a version is checked out, TCS reports which version is checked out, when it was checked out, and by whom. If no version is checked out TCS does not print anything and the CLI prompt returns.

You can use the CLI !FILENAMES pseudo-macro in an argument to obtain a list of all files that are checked out in the current working directory.

The format of the command is

```
) TCSPRINT/v=0 <[!FILENAMES +.TCS]>
```

The !FILENAMES psuedo-macro is described in the *MP/AOS Command Line Interpreter (CLI)* DGC No. 069-400201.

Setting the Revision Number

Use the REVMARK command to assign revision numbers to versions of the master file. For a discussion of revision marks and their use, see "Identifying Versions" in Chapter 1.

You may use the REVMARK command repeatedly to assign more than one revision number to the same version. See the second example below.

NOTE: *If you intend to use the master file in conjunction with the BUILD program, see the discussion of revision marks in the note in the "Reconstructing a Version" section of Chapter 2 before using this command.*

The format of the command is

) REVMARK[/*rev*] *filename*[/VID]=<*revision*>

where

rev Indicates that the VID field is to be interpreted as a revision number. See the example below.

<*revision*> is the revision number that you wish to assign. The format of the revision number <*maj*>.<*min*> where *maj* and *min* are integers in the range 0 to 255 representing the major and minor revision levels, respectively.

VID is the VID of a version other than the last cycle in the default basis.

Examples

The command

```
)REVMARK inter.sr/3=1.00
```

assigns revision number 1.00 to the third cycle of inter.sr.

The command

```
)REVMARK/rev program/3.3=4.3
```

assigns revision number 4.3 to the version that has revision number 3.3. This version now has two revision numbers, Rev 4.3 and Rev 3.3. Both refer to the same version.

Creating a New Basis

To create a new basis in an existing master file, use the NEWBASIS command. The format of this command is

) NEWBASIS/USER=*username*[/*rev*] *modulename*/VID

where

username is your name or the name to be recorded as creator of the basis

rev is the revision. See "Identifying Versions" in Chapter 1.

filename/VID is the version of the file to be used as the starting point of a new branch.

When you enter the NEWBASIS command, TCS prompts you with the message

```
Creating new basis for <filename>/VID
```

and the prompt

```
New basis name:
```

When you have entered the name, TCS types

```
Basis description?
```

and two right parentheses as a prompt. The description may be as long as you like; its function is to inform future users of the reasons for the creation of this basis. Terminate the description with a third right parenthesis on a line by itself.

Figure 2.2 illustrates the creation of a branch from an existing version. In this instance, the second cycle of the original basis was used as the starting version of the new branch. In Figure 2a, the master file has only one basis. Cycle 3 is the sole version derived from cycle 2. In Figure 2b both cycle 3 of the original basis and cycle 1 of the new basis are derived from cycle 2 of the original basis.

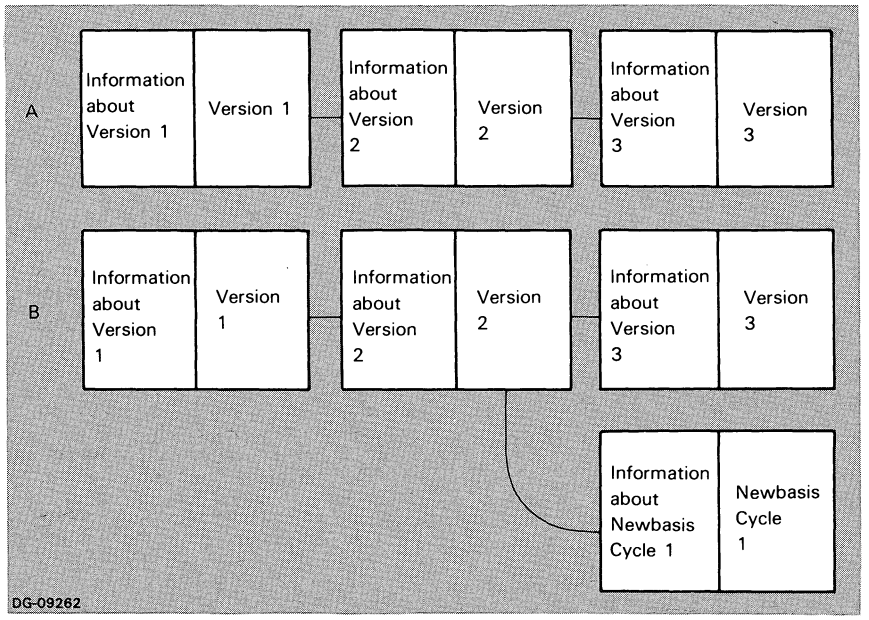


Figure 2.1 Creating a new basis

Example

3

The following example shows the use of each of the TCS commands. It shows the placing of a file under TCS, the CHECKOUT/CHECKIN cycle, the creation of a new basis, the revision-marking of a version of a file, and the use of the TCSPRINT command to obtain information about each version represented in the master file.

Each command used in the example is fully described in Chapters 2; you will find the example in this chapter easier to follow if you read that chapter first.

The example illustrates the use of TCS to maintain a poem though various revisions. Along the way one version becomes the basis for a new poem.

Creating the Master File

Figure 3.1 OPUS_1/.1

The poet, whose name is Jackson, has a rudimentary version of the poem in a file called OPUS_1, which is displayed in Figure 3.1.

To put this file under TCS control he types

```
) NEWTCS/USER=jackson OPUS_1
```

TCS responds with the prompt

```
Describe Opus_1
 ))
```

to which he responds

```
))Opus 1 is the poetic expression of an abiding obsession.
)))
```

Entering a third right parenthesis terminates the description and causes the system to respond

```
the file contained 13 lines.
```

By using the CLI FILESTATUS command, the poet verifies that the working directory no longer contains Opus_1 and does contain Opus_1.TCS.

The Editing Cycle

In order to make changes to the poem, Jackson now causes TCS to create a copy of Opus_1 in the working directory. To do so he uses the command

```
) Checkout/user=Jackson opus_1
```

TCS responds with the message

```
Checking out :UDD:JACKSON:WORKSPACE:OPUS_1/.1
The file contains 13 lines
```

The /.1 appended to the file name indicates that the checked-out version of the file is cycle .1, the first version. The working directory now contains both Opus_1 and Opus_1.TCS.

Jackson now uses a text editor to change the poem. Figure 3.2 shows the poem after the addition of a second stanza. To incorporate these changes in the master file, he now enters the command

```
) Checkin/user=Jackson opus_1
```

TCS responds with the prompt

```
Checking in :UDD:JACKSON:WORKSPACE:opus_1/.2
Describe the change:
))
```

In response to the prompt, the poet types

```
)) I've added a second stanza. Somehow it doesn't feel right.
)))
```

Again, typing the right parenthesis terminates the interaction. The working directory now contains only Opus_1.TCS.

TCS confirms that the changes have been logged with the message

```
Lines unchanged: 12  inserted: 5  deleted: 0
```

```
Crystal Love

My love is crystal
My love is ferric oxide
I place my love in a winter shed
I dress my love in fox hide

I love my love abidingly
Even though her hair is thin
Swing to the left, swing to the right
Go Fight Win!
```

DG-09264

Figure 3.2 OPUS_1/.2

Creating Copies of a Version

Later, Jackson decides to print out a copy of the original, one stanza, version. To create a copy in the working directory, he types

```
) access Opus_1/.1
```

TCS displays the message

```
Accessing :UDD:JACKSON:WORKSPACE:OPUS_1/.1
The file contains 13 lines.
```

The working directory now contains a file called OPUS_1 that is no longer under TCS. Changes made to this file cannot be recorded in the master file. The ACCESS command creates a new copy of the file and leaves the original copy intact. (This command is used by the BUILD program, as explained in Part II).

Note that the /user switch is not needed with the ACCESS command.

Creating a New Basis

After a few more revisions, the poet decides to try switching the gender of the beloved. In order to create two parallel versions of the poem he creates a new basis, starting from the third version (of the original basis), which is displayed in Figure 3.3.

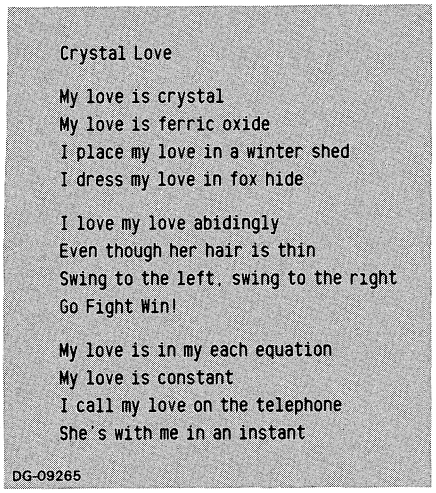


Figure 3.3 OPUS_1/.3

To create the new basis he types

```
) Newbasis/user=jackson Opus_1/.3
```

TCS responds with the message

```
Creating a new basis for Opus_1/.3
New basis name:
```

The poet responds

```
) Him
```

TCS then displays the prompt

```
)) Basis description?
```

and the poet supplies the description

```
)) The same poem, but with a masculine beloved
)))
```

In order to actually create the new poem, the poet must checkout the new basis and edit the file. To do so he enters the command

```
) Checkout/user=jackson Opus_1/him
```

TCS displays the message

```
Checking out :UDD:JACKSON:WORKSPACE:Opus_1/HIM.1
The file contains 23 lines
```

Using the FILESTATUS command, the poet can verify that the working directory now contains two files, Opus_1.TCS and Opus_1. As always, the new version of the file bears the same file name as the original version. The basis name and cycle number are not part of the filename recognized by the operating system; they are internal markings used only by TCS.

In order to check the file in, the programmer must remember to specify the basis name (but not the cycle number). For example, typing

```
) Checkin/user=jackson Opus_1
```

generates the message

```
You did not check out that version
```

In order to check the file back in, the poet must type

```
) Checkin/user=jackson Opus_1/him
```

Final Versions

Figures 3.4 and 3.5 show the latest versions in the two bases. The section entitled "Printing information about versions" shows the use of the TCSPRINT command to trace the evolution of each of these forms.

```
Love's Constancy

My love is crystal
My love is ferric oxide
I place my love in a mineral case
I dress my love in fox hide

My love has faultless circuitry
She never needs a solder
I place my love in a winter shed
I feed my love fodder

My love is in my each equation
My love is constant
I call my love on the telephone
She's with me in an instant
```

DG-09266

Figure 3.4 OPUS_1/.4

```
Oh, Love!

My love is crystal
My love is ferric oxide
I place my love in a winter shed
I dress my love in fox hide

I love my love abidingly
Even though his hair is thin
Swing to the left, swing to the right
Go Fight Win!

My love is in my each equation
My love is constant
I call my love on the telephone
He's with me in an instant
```

DG-09267

Figure 3.5 OPUS_1/HIM.3

Setting a Revision Level

Suppose now that Jackson decides that the latest version in the default basis is relatively stable. He wishes to assign a revision level to the poem in order to co-ordinate it with other poems to be used in an anthology. The poet may revise the poem for a subsequent anthology, and at that time assign a revision number to another version. To assign revision level 1.00 he types

```
) REVMARK Opus/=1.0
```

TCS confirms that the revision level has been set with the message

```
Revision marking :UDD:JACKSON:WORKSPACE:OPUS/.4
```

Printing Information about Versions

The following section uses listings generated by using the TCSPRINT command to trace the development of the master file in the preceding example.

Please refer to Chapter 3 for a fuller discussion of the TCSPRINT command switches.

History of Cycle Creation

Figure 3.6 shows the listing generated using the TCSPRINT command and no switches. It lists the dates of creation and the username for each version on the master file.

```
TCS -- Rev. 0.26    Thursday June 24, 1982    10:27:18 AM
The header of OPUS_1
Cycle 5 was created on 22-Jun-82 at 22:18:08 by JACKSON
Cycle 3 of HIM was created on 11-Jun-82 at 00:23:10 by JACKSON
Cycle 4 was created on 11-Jun-82 at 00:15:02 by JACKSON
Cycle 2 of HIM was created on 11-Jun-82 at 00:09:50 by JACKSON
Cycle 1 of HIM was created on 11-Jun-82 at 00:05:06 by JACKSON
Cycle 3 was created on 11-Jun-82 at 00:02:38 by JACKSON
Cycle 2 was created on 10-Jun-82 at 23:12:46 by JACKSON
Cycle 1 was created on 10-Jun-82 at 23:08:36 by JACKSON
```

DG-09268

Figure 3.6 TCSPRINT (no switches)

Line-by-line Comparison

Figure 3.7 shows the listing generated using the TCSPRINT command with the /body switch. It shows the changes made to the third cycle of the default basis to generate the fourth cycle. The letter *i* next to a line indicates that the line was inserted; the letter *d* indicates a deleted line.

```
TCS -- Rev. 0.26    Thursday June 24, 1982    10:48:38 AM

:UDD:JACKSON:OZ:WORKSPACE:OPUS__1/.4 was created on 11-Jun-82 at 00:15:02

I've finally found the second stanza! ''winter shed'' changed to
''mineral case'' in first stanza. New name too.

  1.
  2.
  3.
  4.
d          Crystal Love
i 5.       Love's Constancy
  6.
  7.
  8.
  9. My love is crystal
 10. My love is ferric oxide
d         I place my love in a winter shed
d         I dress my love in fox hide
d
d         I love my love abidingly
d         Even though her hair is thin
d         Swing to the left, swing to the right
d         Go Fight Win!
i 11.     I place my love in a mineral case
i 12.     I dress my love in fox hide
i 13.
i 14.     My love has faultless circuitry
i 15.     She never needs a solder
i 16.     I place my love in a winter shed
i 17.     I feed my love fodder
   18.
   19.    My love is in my each equation
   20.    My love is constant
   21.    I call my love on the telephone
   22.    She's with me in an instant
   23.
```

DG-09269

Figure 3.7 TCSPRINT/BODY

Cycle Comments

Figure 3.8 shows the listing generated using the TCSPRINT command with the /HEADER/v=2 switch combination. This list contains the history of cycle information that is contained in Figure 3.7, as well as the comments about changes made during each editing cycle.

```
TCS -- Rev.0.26    Thursday June 24, 1982    10:34:50 AM

The header of OPUS_1
Revision 1.000 is cycle 4

Cycle 5 was created on 22-Jun-82 at 22:18:08 by JACKSON
->added blank line

Cycle 3 of HIM was created on 11-Jun-82 at 00:23:10 by JACKSON
->I'm going to create a new poem around the second stanza. I've
->changed the name to Oh, Love!

Cycle 4 was created on 11-Jun-82 at 00:15:02 by JACKSON
->I've finally found the second stanza! ''winter shed'' changed to
->"mineral case" in first stanza. New name too.

Cycle 2 of HIM was created on 11-Jun-82 at 00:09:50 by JACKSON
->Feminine pronouns changed to masculine.

Cycle 1 of HIM was created on 11-Jun-82 at 00:05:06 by JACKSON

Cycle 3 was created on 11-Jun-82 at 00:02:38 by JACKSON
->I added a third stanza. The second still doesn't feel right.

Cycle 2 was created on 10-Jun-82 at 23:12:46 by JACKSON
->I've added a second stanza. Somehow it doesn't feel right.

Cycle 1 was created on 10-Jun-82 at 23:08:36 by JACKSON
```

DG-09270

Figure 3.8 TCSPRINT/V=2

Cycle and Basis Comments

Figure 3.9 shows the listing generated using the TCSPRINT command with the /HEADER/v=3 switch. It contains the information contained in Figure 3.8 as well as the comments that describe each basis.

```
TCS -- Rev. 0.26    Thursday June 24, 1982    10:35:28 AM

The header of OPUS__1
Revision 1.000 is cycle 4

Basis HIM
->The same poem, but with a masculine beloved

Default basis
->Opus 1 is the poetic expression of an abiding obsession.

Cycle 5 was created on 22-Jun-82 at 22:18:08 by JACKSON
->added blank line

Cycle 3 of HIM was created on 11-Jun-82 at 00:23:10 by JACKSON
->I'm going to create a new poem around the second stanza. I've
->changed the name to Oh, Love!

Cycle 4 was created on 11-Jun-82 at 00:15:02 by JACKSON
->I've finally found the second stanza! ''winter shed'' changed to
->''mineral case'' in first stanza. New name too.

Cycle 2 of HIM was created on 11-Jun-82 at 00:09:50 by JACKSON
->Feminine pronouns changed to masculine.

Cycle 1 of HIM was created on 11-Jun-82 at 00:05:06 by JACKSON

Cycle 3 was created on 11-Jun-82 at 00:02:38 by JACKSON
->I added a third stanza. The second still doesn't feel right.

Cycle 2 was created on 10-Jun-82 at 23:12:46 by JACKSON
->I've added a second stanza. Somehow it doesn't feel right.

Cycle 1 was created on 10-Jun-82 at 23:08:36 by JACKSON
```

DG-09271

Figure 3.9 TCSPRINT/V=3

Part 2

The Build Program

Context of the BUILD Program

1

The following documentation describes the function and implementation of BUILD, which is a program that speeds and simplifies the construction of products through transformation of files. A product may be any file or collection of files, such as a program, a collection of programs, a library, a book chapter, or a bill of materials. The BUILD program is closely associated with the Text Control System TCS, which is described in Part 1 of this manual. This documentation is written for users who are comfortable with TCS.

The BUILD documentation is in three chapters. The first presents an overview of what BUILD does, introduces a terminology, and summarizes BUILD program logic. The second chapter explains how to use BUILD, and the third contains an example.

Problems of Product Construction

The construction of a product is often a multistage process involving the transformation and combination of other files. For example, the construction of a program file involves the transformation of files through assembling and compiling, and the combination of the resultant object modules through binding. The end file is dependent on all the files involved in its construction. Intermediate files are dependent on some files but not on others. You direct the construction of a file through a series of CLI commands, which may be grouped into a CLI macro.

The construction, and especially the reconstruction, of files in this way presents two classes of problems. The first class arises from the possibilities for error and redundancy inherent in a complex file construction. It is common to do extra operations (redundancy) or to forget to perform needed operations (error). The second class arises from the need to coordinate many programs or modules that exist in different versions.

Avoiding Error and Redundancy

The complexity of the file-construction procedure presents a problem with two aspects. In the first place, it is easy to make a mistake when you are organizing the procedure and giving the instructions for its execution. In the second place, the combinations and transformations take time and computer resources. The construction of a large program through compiling, assembling, and binding can take hours.

To make sure that all modifications to constituent files are taken into account you may "start from scratch" every time -- that is, re-assemble or recompile every source module, and so forth. Such an exhaustive approach (in many cases the only practical one) is, of course, the most time- and resource-consuming. What's worse, logical errors in a program-building macro can crop up when the macro has run halfway to completion.

Co-ordinating Revisions

A related problem involves the synchronization of revision numbers. If you were creating a program of revision number x, you might want all the modules used in its creation to have revision number x. Remembering revision levels is not practical when the number of modules is large. On the other hand, keeping copies of all modules to make a given revision may require a high disk storage overhead.

The BUILD Solution

The BUILD approach to avoiding the error and redundancy problems described above is to calculate the minimal number of transformations necessary to reconstruct the product(s). The approach to the problem of revision co-ordination is to provide an invisible interface to the TCS program, and to use TCS to co-ordinate revision levels.

You supply BUILD with a description of the file-construction procedure. BUILD uses this description to chart out the dependency relationships among files. BUILD determines which files have been modified and constructs a CLI macro to implement the shortest procedure that will reflect all of the modifications.

It is still possible for you to make errors in the specification of the procedure, but BUILD is able to detect certain logical errors before the macro is executed.

You may instruct BUILD to automatically execute the macro or to first make it available for your inspection.

BUILD is designed to interact with the TCS utility. It accesses TCS files when necessary, and uses the TCS revision marking facility to synchronize the revision levels of all files used in the construction of the end product.

Uses of the BUILD Program

In addition to using BUILD to manage the creation of program files from source files, you can use BUILD to manage any procedure that combines files. For example, you could use BUILD to update a documentation set comprised of text files modified by a Speed program. Or you could use BUILD in the construction of a new data base system derived from manipulations and combinations of other data bases.

Keep in mind when reading the following text and examples that BUILD is a general-purpose program not limited to program development.

Definition of Terms

The presentation of the BUILD program requires the introduction of several terms and concepts. Some of these terms are related to the concept of the *dependency tree*, which is a representation of dependency relationships among files. Other terms are related to the BUILD command and output files.

Dependency

A fundamental BUILD concept is *dependency*. A product depends on any file which, if modified, would change the product.

For example, a Pascal object file depends on its corresponding source file, as well as any include files that it uses; an assembler object file depends on the corresponding source file as well as on the permanent symbol file used by the assembler; and a program file depends on all the object modules and libraries used to bind it, including system libraries.

If a macro is used to direct the construction of a product, then the product is also dependent on the macro file. For example, if a macro BX.CLI, is used to bind program X, then changing the macro would obviously change the program X.

Dependency Tree

The *dependency tree* is a mapping of the relationships among the files processed by BUILD. In Figure 1.1, each solid block at a node or terminus represents a file that is a transformation of those connected to it; each line represents a transformation directed by a CLI command or series of commands.

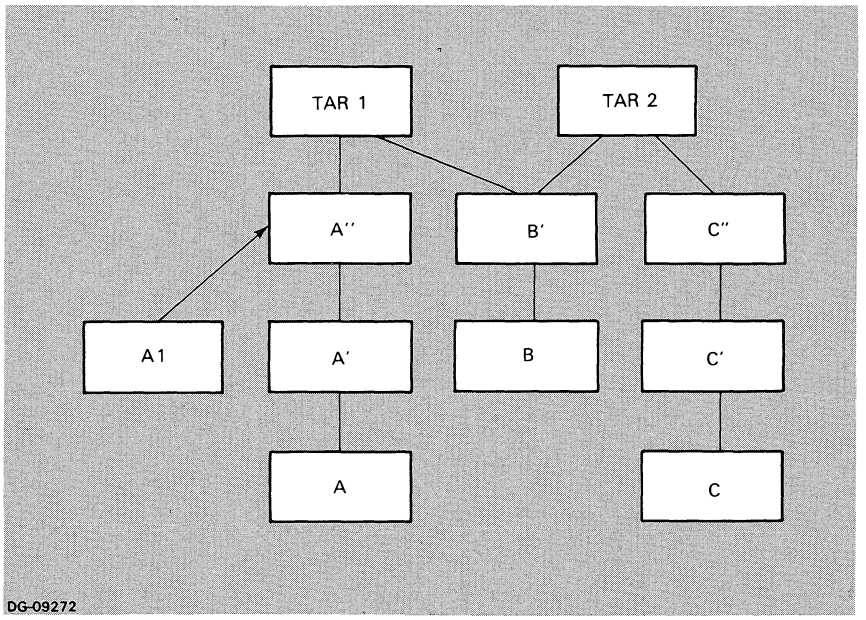


Figure 1.1 The BUILD dependency tree

Files are dependent on those files below them to which they are connected directly or indirectly. Blocks in the topmost echelon are the *target* files that are the object of the construction; blocks in the bottom row are the *level 0* files which are not the result of any transformation mediated by BUILD. Files that are one transformation removed from level 0 are level 1, and so on.

The paragraphs below define the terms used to describe the tree.

Target files are the files that you wish to construct. A dependency tree may have one or many target files. Often, a target file is a program file to be constructed from modules in programming languages. The set of target files comprise the product.

In Figure 1.1, Tar1 and Tar2 are target files.

Constituent files are those files that are transformed and combined to form the target files. In Figure 1.1, constituent files are all the solid blocks except the target files. Constituent files may be either *level 0* or ("intermediate") files.

Level 0 files are those supplied 'as is' to BUILD. They are not the result of any transformation that BUILD controls. In Figure 1.1, A, A1, B, and C are level 0 files.

Intermediate files depend either on level 0 files or on level 0 files. In Figure 2.1, A', A'', B', C', C'' are intermediate files.

BUILD Command File

The command file contains your description of the procedure to create the target files. In it, you specify the names of all the target, level 0, and intermediate files, and the transformations to be performed. BUILD uses the command file to derive the structure of the dependency tree and represents that structure in a *history file.*

The format of the command file and instructions for its creation appear in Chapter 2.

The History File

The *history file* is a file created by BUILD that encodes the structure of the dependency tree, as well as status information about each file used in the construction. By analysis of the history file and by the current status of the files, BUILD determines the minimal number of transformations that need to be performed to create the target file or files.

NOTE: *The history file is printable, and you can therefore examine it if you are curious. You do not need to understand or even look at this file in order to use BUILD. You should never modify it. If you use the BUILD/REVMARK command, the history file is placed under TCS; DO NOT check out the file, as this will cause errors the next time you build a product.*

BUILD Program Logic

The following paragraphs sketch an outline of BUILD's operation. The example in the next section complements the discussion.

The use of BUILD to construct or reconstruct a target file requires a history file and a command file. The next chapter explains how to write the command file and how to instruct BUILD to generate an initial history file.

Using the command file you specify on the command line, BUILD generates a new history file which is a representation of the dependency tree. By comparing the new history file with the previous one, and by inspection of the directories on the seachlist, BUILD determines which files are missing or modified. In order to calculate the impact of missing and modified files, BUILD traces transformation paths from level 0 to target files, and back down.

Then BUILD constructs the shortest macro that will rebuild the missing files, repeat the transformations on modified files, and build the target files. The macro usually includes steps to rebuild some intermediate files, and omits steps to rebuild others. In order to ensure that a file is, or is not, rebuilt, you may use override files as explained in Chapter 2.

Using BUILD

2

This chapter explains how to use BUILD. It describes the procedures for constructing a command file and for creating the initial history file. It also highlights some of BUILD's varied uses.

Among the applications described are

- constructing target files
- synchronizing revision levels
- constructing the BUILD macro without executing it
- exploring consequences of changes to files.

Switch	Effect
/ALL	Preforms the entire construction procedure; does not check for file modifications
/HIST	Constructs a history file. Use this switch to obtain a history file for a first use of BUILD
/L	Redirects output to @LPT
/L=<file>	Redirects output to <file>
/NOBUILD	Creates a procedural macro but does not execute it.
/REVMARK	Assigns revision level specified by VID to all TCS files (target and constituent)
/ACCESS=<value>	When <value> equals ALWAYS, BUILD accesses all TCS files, whether necessary to rebuild the product or not. See Part 1, Chapter 2 for a discussion of ACCESSing.
	When <value> equals NEVER, BUILD does not access any TCS files, whether or not they are needed to rebuild the product.
	When you do not use this switch, BUILD accesses TCS files as needed.
/O	Causes BUILD to use the TCS /O switch when accessing files. See Chapter 2, Part 1 for a discussion of the /O switch.
/USER=<username>	Identifies the user. See the discussion of this switch in the Part 1, Chapter 1.

Table 2.1 BUILD switches

The Command Line

You execute BUILD by issuing a CLI command. The arguments and switches on this line determine the type of output produced and the files to which output is stored or written.

The format of the command line is

XEQ BUILD[*/sw <command_file>* [/VID] *[override file specification]*

where

SW is one or more of the switches listed in Table 3.1. Certain of these swithes are highlighted in the examples; the more elementary switches are listed only in the table.

<command_file> is the command file. The format of the command file is explained below.

VID is the TCS version identifier. This switch is used in conjunction with the REVMARK switch described below ("Syncronizing Revisions"). Refer to Part 1 for a discussion of TCS.

<override specification> <. . .> are files and their associated switches that you can use to modify the way BUILD constructs the macro. Override files are discussed later in this chapter and the switches are listed in Table 2.3.

The Command File

The command file is a description of the target file construction procedure. Its format is that of CLI macros structured around a skeleton of BUILD key words. Four keywords define *functional areas* of the command file, as explained below. Two other keywords define actions within one of the functional areas. Table 2.2 lists the command file keywords.

Keyword	Function
SEARCHLIST	Marks area of command file to set searchlist for the duration of BUILD
SETUP	Marks area of command file which includes CLI command to be executed prior to determination of the BUILD macro.
BUILD	Marks area of the command file that contains target file construction procedure.
CLEANUP	Marks area of the command file that includes CLI commands to be executed after the execution of the BUILD macro.
FROM	Within BUILD area, identifies constituent files to be used in the construction.
WITH	Within BUILD area, identifies CLI commands to be used to create target and intermediate file.
COMMENT	Identifies area of a command file containing a comment.

Table 2.2 BUILD keywords

Keyword areas in the file are delimited by semi-colons, and new-line characters may appear in a command. BUILD does not distinguish between upper and lower case.

Functional Areas of the Command File

The keywords SEARCHLIST, SETUP, and CLEANUP may each define one area of the command file. All of these keywords are optional. The BUILD keyword is not optional, and may be repeated. The COMMENT keyword is optional and may be repeated. Comments may appear anywhere in the command file; when the other keywords are used they must be in the order in which they are described below.

The WITH and FROM keywords appear within the BUILD area, and set off clauses in the BUILD instructions.

Chapter 3 contains an example of a command file.

SEARCHLIST

The SEARCHLIST area consists of a single command that defines the searchlist for the duration of the BUILD. The format of this command is

SEARCHLIST <*searchlist pathnames*>;

Use commas to separate individual pathnames and a semicolon to terminate. When used, the SEARCHLIST command must appear first in the command file and may appear only once.

SETUP

The SETUP area allows you to specify CLI commands that will be automatically executed by the CLI that BUILD invokes. A typical use of this area is the setting of the default attributes for the files created by BUILD.

Do not use the CLI DIRECTORY command in this or any section of the command file.

BUILD

The BUILD area of the command file describes the actual procedure for the construction of an intermediate or target file. Each area begins with the word BUILD and contains one or more lines of the format

<*target*> FROM <*constituents*> WITH <*CLI commands*>;

where

<*target*> is the name of a target file or intermediate file to be constructed

<*constituents*> are the names of the constituent files used in the construction of <target>

<*CLI commands*> are commands directing the construction of the target file. Multiple commands bay be separated by new-lines. BUILD uses these commands in the construction of the BUILD macro.

There is no limit to the number of BUILD statements that appear in a command file. Each construction listed after the WITH keyword corresponds to a transformation on the dependency tree. Since the object of BUILD is to minimize unnecessary transformations, you should leave BUILD the opportunity of leaving out as many commands as possible.

To do so, keep BUILD lines simple by limiting the number of operations specified in the WITH clause. Keeping this principle in mind, you may list as many operations as you like in the WITH clause.

For example, the following two constructions are equivalent:

```
1) BUILD a.pr FROM a.fr b.ob WITH x fort4 a
                                  x masm a
                                  x bind a b;

2) BUILD a.pr FROM a.ob b.ob WITH X bind a b;
     a.ob FROM a.sr      WITH X masm a;
     a.sr FROM a.FR      WITH X fort4 a;
```

You should use the second construction, however, since the first construction would require two unnecessary transformations *(x fort4 a, x masm a)* in the event that only *b.ob* had been modified. The commands in the BUILD area may be in any order.

CLEANUP

The CLEANUP area of the command file allows you to specify CLI commands to 'do housekeeping' after the execution of the BUILD macro. The format of this area is

CLEANUP <CLI *commands*>;

For example,

```
CLEANUP delete +.LS;
```

would delete all listing files generated by the BUILD.

There may be at most one CLEANUP area in the command file.

Override Files

Override files are files that are marked for special consideration. Using override file switches, you may mark files as modified or unmodified regardless of their actual state, and you may mark files not to be affected by the /REVMARK switch. Each override file must have one and only one switch. Switches are listed in Table 2.3.

Switch	Effect
/U	BUILD considers the file as unmodified
/M	BUILD considers the file as modified
/NR	BUILD does not set the revision level of the file.

Table 2.3 Override file switches

Sample Applications

The following examples show some of the uses of BUILD and illustrate the use of switches, command files, and override files.

Logging the Macro

By using the /L [=*filename*] switch on the BUILD command line you can create a copy of the macro that BUILD creates. The macro shows you the exact sequence that BUILD used to construct the target file. When you use the /L=[*filename*] switch together with the /NOBUILD switch, BUILD produces a copy of the macro without executing it.

Getting Started

Before you can use BUILD to construct a target file, a history file for the procedure must exist. To create an initial history file, use the following procedure.

- Write a command file containing instructions to create the target file.
- Execute BUILD using the /HIST switch. This will cause BUILD to check that the procedure is logically consistent. If it is, BUILD creates an initial history file.
- Execute BUILD using the /ALL/NOBUILD/L switches to produce a macro showing the entire procedure for an exhaustive build. Keep this macro for a reference.
- Execute BUILD without the /NOBUILD switch to build the target file(s).

Updating a File

When you wish to reconstruct a file after modifying one or more constituent files, simply execute the BUILD command using the same command file that you used before.

When you have changed all (or virtually all) of the level 0 files, you may use the /ALL switch. This causes BUILD to forego the inspections that detect missing and modified files, and to execute the entire BUILD procedure.

The /ALL switch is also useful when using a vital file such as an unproven compiler, or an improved compiler.

If you have modified a file but know the changes to be insignificant (say, for example, you are building a program file and have modified only the comments in a source module) you may specify that file as an override file with the /U switch.

Building a New Revision

When you wish to assign the same revision number to the target file and some or all constituent files, use the /REVMARK switch, and the /VID switch on the command file name. Files that are to be revision-marked must be under TCS. BUILD uses TCS to set the revision level of the latest version of the default basis.

NOTE: *Do not CHECKOUT the history file; BUILD automatically updates it. If the history file is checked out by users other than the BUILD program, errors may arise on the next Build.*

If the command file is not under TCS, BUILD places both the command file and the history file under TCS, and assigns the number you specify using the /VID switch as the revision level of the command, history, constituent and target files.

To disable the revision-marking of any file, list that file as an override file and use the /NR switch. This switch is useful for files that are under TCS and shared among products maintained by BUILD.

Reconstructing a Version

When you have used the above-described procedure to synchronize the revision levels of constituent and target files, you may re-create the target files by using the /VID command file switch without using the BUILD /REVMARK switch.

NOTE: *In general you should use this procedure rather than the TCS revmark command. If you set the revision level using the TCS REVEMARK command, TCS will issue a warning when you use BUILD/REVMARK. You may safely disregard this warning.*

Exploring Hypothetical Changes

Since BUILD can trace dependency among files used to create a target file, you can use BUILD as a reference tool to see what files would be affected if you modified one or more files.

To use BUILD in this way, use the following command line:

XEQ BUILD/NOBUILD/L=@TTO <*cmd-file*> <*mod file1*>/M . . ....< *mod filen*>/m

This will cause the BUILD macro to be displayed at the console.

By analysis of the macro you can determine which files have to be rebuilt as a consequence of modifying the files you specify with the override switch.

3 A Build Example

The example in this chapter shows the use of BUILD to create a program file from several Pascal source files. The chapter shows how to construct a BUILD command file to describe the construction of the target file. It describes the actions that BUILD takes when it encounters missing or modified files while preparing the BUILD macro. Finally, it shows typical messages directed to the standard output file during execution of BUILD.

The Problem

The following example shows the use of BUILD to direct the construction of a target file through complilation and binding of input files. In the example, certain of the input files are missing or have been modified since BUILD was last used.

The dependency tree is represented in Figure 3.1. The target files are AB.PR and BC.PR; four level 0 files are A.PAS, A1.PINC B.PAS and C.PAS. A.PAS and C.PAS are maintained under TCS. As illustrated in the figure, the A.PAS.TCS file has been modified since the last use of BUILD, the source B.PAS has also been modified, and neither C.PAS or C.OB exist in the directories on the searchlist.

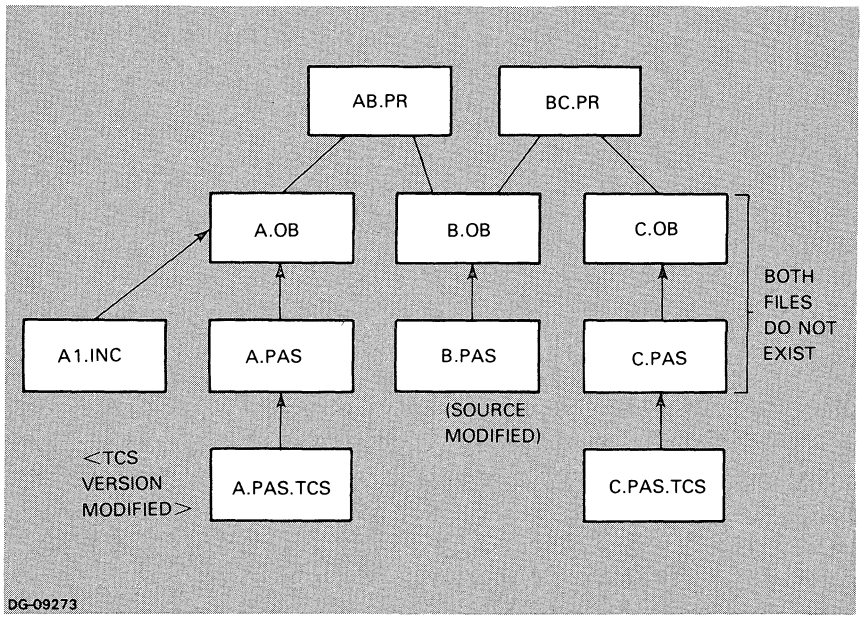


Figure 3.1 Example of a BUILD dependency tree

The Command File

Figure 3.2 illustrates the command file used to direct the construction of the two program files. The file contains two BUILD areas separated by a comment, as well as one SETUP, one SEARCHLIST, and one CLEANUP area.

```
SEARCHLIST
   :util, :udd:don:build:doc;

COMMENT
   Clear the attributes to make sure that no files
   are read or write protected;

SETUP
   Attributes/k;

BUILD
   a.ob from a.pas, a1.pinc with pascal a;
   b.ob from b.pas with b;
   c.ob from c.pas with pascal c;

COMMENT Compilation complete, binding next;

BUILD
   ab.pr from a.ob b.ob with x mbind/p=ab/aos a b pascal.lb;
   bc.pr from b.ob c.ob with x mbind/p=bc/aos b c pascal.lb;

CLEANUP
   Delete +.ls
```

DG-09274

Figure 3.2 BUILD command file

Determination of the Procedure

Refer back to Figure 3.1 to follow this explanation of BUILD logic.

By tracing dependency relationships from level 0 files to the target files it is possible to determine which target files must be rebuilt; having determined that, by tracing from target files down to level 0, it is possible to see which intermediate files must be re-created.

Reading Up

Since A.PAS.TCS has been modified, changes to the A.PAS version must be reflected in all target files dependent on it. Therefore, a new version of A.PAS must be accessed in order that it be compiled to form A.OB. and ultimately AB.PR.

Similarly, since B.PAS has been modified, B.OB must be recreated in order to reflect the modifications. B.OB is input to both AB.PR and BC.PR; therefore, both target files must be rebuilt.

Tracing down

C.PAS.TCS has not been modified; this implies that C.PAS and C.OB if re-created would contain no new information. C.OB is required, however, in order to bind B.OB and C.OB together. Since C.OB is required to build BC.PR, BUILD must re-create it. In order to do so, it must first get a copy of C.PAS, which it does automatically through TCS.

Messages to Console

During the BUILDing of the product, executing programs may send messages to the standard output file. This information is provided solely to keep you apprised of what operations are being performed at the time, and does not replace the log file.

Figure 3.3 illustrates the screen messages sent to the console during the building of AB.PR and BC.PR.

```
) build/o ex

Accessing :UDD:DON:BUILD:DOC:A.PAS/.2

The file contains 10 lines.

Accessing :UDD:DON:BUILD:DOC:C.PAS/.1

The file contains 6 lines

MP/Pascal    Rev 2.30    18-MAY-82    21:11:32

Source file :UDD:DON:BUILD:DOC:A.PAS
Working directory :UDD:DON:BUILD:DOC
Searchlist

:UTIL:TCS,:UDD:DON,:UDD:DON:MACROS,:UTIL,:,:GAMES,:KS:UTIL:PASCAL

11 source lines were compiled in 6 seconds

No Compilation Errors
No Compilation Errors

MP/Pascal    Rev 2.30    18-MAY-82    21:11:40

Source file :UDD:DON:BUILD:DOC:B.PAS
Working directory :UDD:DON:BUILD:DOC
Searchlist
:UTIL:TCS,:UDD:DON,:UDD:DON:MACROS,:UTIL,:,:GAMES,:KS:UTIL:PASCAL

7 source lines were compiled in 4 seconds

No Compilation Errors
No Compilation Errors

MP/Pascal    Rev 2.30    18-MAY-82    21:11:50

Source file :UDD:DON:BUILD:DOC:C.PAS
Working directory :UDD:DON:BUILD:DOC
Searchlist
:UTIL:TCS,:UDD:DON,:UDD:DON:MACROS,:UTIL,:,:GAMES,:KS:UTIL:PASCAL

7 source lines were compiled in 4 seconds

No Compilation Errors
No Compilation Errors

Rebuilt 6 of 8 total files, final products were

   7 :UDD:DON:BUILD:DOC:AB.PR
   8 :UDD:DON:BUILD:DOC:BC.PR

)
```

DG-09275

Figure 3.3 BUILD screen message

Part 3

The FIND Utility

FIND

1

FIND is utility program that examines a textfile for the occurrence of a pattern. You use FIND in place of searches with text editors; FIND is espescially useful when you are working at a low baud rate and displays of entire texts are impractical.

To specify a pattern (text string), you may type the pattern exactly as it occurs, or use one of several templates that match a range of patterns.

When the pattern is detected, FIND prints the line in which the pattern occurs. By using command line switches, you can direct FIND to print other information, such as the line number of each line containing a match, the number of lines containing matches, and a specifiable number of lines following the match.

FIND either recognizes or ignores differences between upper and lower cases, according to your specification. By default it is case-indifferent.

Starting and Stopping FIND

You start FIND by typing the command line explained below. To stop FIND before it has finished inspecting the entire text, type CTRL-C CTRL-A. The format of the command line is

XEQ FIND[/*sw*] <*filename*> <*pattern*>

where SW is one of the switches listed in Table 1.1.

filename is the name of the file to be searched.

pattern is a text string or template. The text string may not contain characters that have special meaning to the CLI. See the discussion below for explanation of pattern templates.

Switch	Action
/L	Directs output to @LPT instead of @TTO
/L=<*filename*>	Appends output to <*filename*>. Creates <filename> if it doesn't exist.
/NCI	Sets FIND to detect differences between upper- and lowercase letters. When this switch is not used, FIND is case-indifferent.
/C	Prints a count of the number of lines containing matches
/N	Prints line numbers preceding each output line.
/S=<*integer*>	Prints each line containing a match, and the number of suceeding line specified by <*integer*>.
/H	Prints a header containing the name of the searched file.

Table 1.1 FIND switches

Definition of Terms

An *element* is a character or a range of characters. Ranges are indicated by two characters separated by a hyphen. For example, A and a-z are elements. In a range, the first character must be less than the second in the ASCII collating sequence.

A *set* is a group of elements with no implied ordinality or cardinality. Sets are indicated by a list of elements separated by commas and enclosed within square brackets.

Escaped characters are alphabetic characters preceded by an escape character, which stands for the control (CTRL) function. The escape character is generated by pressing the escape key on the left of the keyboard, and is represented by a dollar sign.

A *Class* is either a character, an escaped character, or a set.

Pattern Templates

Anchoring a Pattern to a Margin

The vertical bar (|) at the right or left of a pattern matches 0 characters and therefore "anchors" the pattern to the respective margin.

For example,

| Daddy
matches
Daddy is home
but not
My Daddy is home
or
Daddy is home.

Patterns not containing a vertical bar will match any occurrence of the pattern. Using anchored patterns greatly increases FIND's performance.

Alternative Patterns

To specify alternative patterns, list the patterns within parentheses and separated by verical bars. Each line containing any of the patterns is printed.

For example

(kw | art)

matches *Kumkwat* or *artichoke.*

Matching Pattern or Null String

Setting a pattern in curly brackets causes the pattern to match itself or the null string. A null string matches at any character position on the line. For example,

a{b}c

matches *abc* and *ac.*

Matching One or More Occurrences of a Pattern

Setting a pattern in curly brackets followed by a plus sign causes the pattern to match one or more occurrences of the pattern. For example,

a{bc}+

matches *abc*, *abcbc*, *abcbcbc*, etc.

Matching Zero or More Occurrences of a Pattern

Setting a pattern in curly brackets followed by an asterisk causes the pattern to match zero or more occurrences of the pattern. Thus {pattern}+ is equivalent to pattern{pattern}*. For example,

*a{b}*c*

matches *ac*, *abc*, *abbc*, *abbbcdc*, etc.

Matching an Element of a Set

Specifying a set (list of elements separated by commas and set off by square brackets) causes FIND to match any member of the set.

For example, the set

[*a-c*, *k*, *x-z*]

matches lines containing *a* or *b* or *c* or *d* or *k* or *x* or *y* or *z*.

Matching Characters Not Members of a Class

Placing a tilde (~) in front of a class specification causes FIND to match any line that does not contain a member of the class.

For example

~*$i*
matches any line that does not contain a tab (CTRL-I) , and

~*[a-z]*

matches any line that does not contain any letters.

Matching Any Single Character

The number sign (#) matches any single character. For example,

jain#ba

matches *jaineba*, *jainbba*, and *Jainaba*.

Index

O

P

R

S

T

V

DG OFFICES

NORTH AMERICAN OFFICES

Alabama: Birmingham
Arizona: Phoenix, Tucson
Arkansas: Little Rock
California: Anaheim, El Segundo, Fresno, Los Angeles, Oakland, Palo Alto, Riverside, Sacramento, San Diego, San Francisco, Santa Barbara, Sunnyvale, Van Nuys
Colorado: Colorado Springs, Denver
Connecticut: North Branford, Norwalk
Florida: Ft. Lauderdale, Orlando, Tampa
Georgia: Norcross
Idaho: Boise
Iowa: Bettendorf, Des Moines
Illinois: Arlington Heights, Champaign, Chicago, Peoria, Rockford
Indiana: Indianapolis
Kentucky: Louisville
Louisiana: Baton Rouge, Metairie
Maine: Portland, Westbrook
Maryland: Baltimore
Massachusetts: Cambridge, Framingham, Southboro, Waltham, Wellesley, Westboro, West Springfield, Worcester
Michigan: Grand Rapids, Southfield
Minnesota: Richfield
Missouri: Creve Coeur, Kansas City
Mississippi: Jackson
Montana: Billings
Nebraska: Omaha
Nevada: Reno
New Hampshire: Bedford, Portsmouth
New Jersey: Cherry Hill, Somerset, Wayne
New Mexico: Albuquerque
New York: Buffalo, Lake Success, Latham, Liverpool, Melville, New York City, Rochester, White Plains
North Carolina: Charlotte, Greensboro, Greenville, Raleigh, Research Triangle Park
Ohio: Brooklyn Heights, Cincinnati, Columbus, Dayton
Oklahoma: Oklahoma City, Tulsa
Oregon: Lake Oswego
Pennsylvania: Blue Bell, Lancaster, Philadelphia, Pittsburgh
Rhode Island: Providence
South Carolina: Columbia
Tennessee: Knoxville, Memphis, Nashville
Texas: Austin, Dallas, El Paso, Ft. Worth, Houston, San Antonio
Utah: Salt Lake City
Virginia: McLean, Norfolk, Richmond, Salem
Washington: Bellevue, Richland, Spokane
West Virginia: Charleston
Wisconsin: Brookfield, Grand Chute, Madison

DG-04976

INTERNATIONAL OFFICES

Argentina: Buenos Aires
Austraiia: Adelaide, Brisbane, Hobart, Melbourne, Newcastle, Perth, Sydney
Austria: Vienna
Belgium: Brussels
Bolivia: La Paz
Brazil: Sao Paulo
Canada: Calgary, Edmonton, Montreal, Ottawa, Quebec, Toronto, Vancouver, Winnipeg
Chile: Santiago
Columbia: Bogata
Costa Rica: San Jose
Denmark: Copenhagen
Ecuador: Quito
Egypt: Cairo
Finland: Helsinki
France: Le Plessis-Robinson, Lille, Lyon, Nantes, Paris, Saint Denis, Strasbourg
Guatemala: Guatemala City
Hong Kong
India: Bombay
Indonesia: Jakarta, Pusat
Ireland: Dublin
Israel: Tel Aviv
Italy: Bologna, Florence, Milan, Padua, Rome, Tourin
Japan: Fukuoka, Hiroshima, Nagoya, Osaka, Tokyo, Tsukuba
Jordan: Amman
Korea: Seoul
Kuwait: Kuwait
Lebanon: Beirut
Malaysia: Kuala Lumpur
Mexico: Mexico City, Monterrey
Morocco: Casablanca
The Netherlands: Amsterdam, Rijswijk
New Zealand: Auckland, Wellington
Nicaragua: Managua
Nigeria: Ibadan, Lagos
Norway: Oslo
Paraguay: Asuncion
Peru: Lima
Philippine Islands: Manila
Portugal: Lisbon
Puerto Rico: Hato Rey
Saudi Arabia: Jeddah, Riyadh
Singapore
South Africa: Cape Town, Durban, Johannesburg, Pretoria
Spain: Barcelona, Bibao, Madrid
Sweden: Gothenburg, Malmo, Stockholm
Switzerland: Lausanne, Zurich
Taiwan: Taipei
Thailand: Bangkok
Turkey: Ankara
United Kingdom: Birmingham, Bristol, Glasgow, Hounslow, London, Manchester
Uruguay: Montevideo
USSR: Espoo
Venezuela: Maracaibo
West Germany: Dusseldorf, Frankfurt, Hamburg, Hannover, Munich, Nuremburg, Stuttgart

Data General

Ordering Technical Publications

TIPS is the Technical Information and Publications Service—a new support system for DGC customers that makes ordering technical manuals simple and fast. Simple, because TIPS is a central supplier of literature about DGC products. And fast, because TIPS specializes in handling publications.

TIPS was designed by DG's Educational Services people to follow through on your order as soon as it's received. To offer discounts on bulk orders. To let you choose the method of shipment you prefer. And to deliver within a schedule you can live with.

How to Get in Touch with TIPS

Contact your local DGC education center for brochures, prices, and order forms. Or get in touch with a TIPS administrator directly by calling (617) 366-8911, extension 4086, or writing to

Data General Corporation
Attn: Educational Services, TIPS Administrator
MS F019
4400 Computer Drive
Westborough, MA 01580

TIPS. For the technical manuals you need, when you need them.

DGC Education Centers

Boston Education Center
Route 9
Southboro, Massachusetts 01772
(617) 485-7270

Washington, D.C. Education Center
7927 Jones Branch Drive, Suite 200
McLean, Virginia 22102
(703) 827-9666

Atlanta Education Center
6855 Jimmy Carter Boulevard, Suite 1790
Norcross, Georgia 30071
(404) 448-9224

Los Angeles Education Center
5250 West Century Boulevard
Los Angeles, California 90045
(213) 670-4011

Chicago Education Center
703 West Algonquin Road
Arlington Heights, Illinois 60005
(312) 364-3045

DataGeneral

Technical Products Publications Comment Form

Please help us improve our future publications by answering the questions below. Use the space provided for your comments.

Title: _____

Document No. 069-400208-00

Yes	No		
☐	☐	Is this manual easy to read?	○ You (can, cannot) find things easily. ○ Other: ○ Language (is, is not) appropriate. ○ Technical terms (are, are not) defined as needed.
		In what ways do you find this manual useful?	○ Learning to use the equipment ○ To instruct a class. ○ As a reference ○ Other: ○ As an introduction to the product
☐	☐	Do the illustrations help you?	○ Visuals (are,are not) well designed. ○ Labels and captions (are,are not) clear. ○ Other:
☐	☐	Does the manual tell you all you need to know? What additional information would you like?	
☐	☐	Is the information accurate? (If not please specify with page number and paragraph.)	

Name: _____ Title: _____

Company: _____ Division: _____

Address: _____ City: _____

State: _____ Zip: _____ Telephone: _____ Date: _____

DG-06895

DataGeneral
Data General Corporation, Westboro, Massachusetts 01580

FOLD FOLD

TAPE TAPE

FOLD FOLD

NO POSTAGE
NECESSARY
IF MAILED
IN THE
UNITED STATES

BUSINESS REPLY MAIL
FIRST CLASS PERMIT NO. 26 SOUTHBORO, MA. 01772

Postage will be paid by addressee:

Data General

ATTN: Technical Products Publications (C-138)
4400 Computer Drive
Westboro, MA 01581

Data General

users group

Installation Membership Form

Name _____ Position _____ Date _____

Company, Organization or School _____

Address _____ City _____ State _____ Zip _____

Telephone: Area Code _____ No. _____ Ext. _____

1. Account Category

- ☐ OEM
- ☐ End User
- ☐ System House
- ☐ Government
- ☐ Educational

2. Hardware

	Qty. Installed	Qty. On Order
M/600		
COMMERCIAL ECLIPSE		
SCIENTIFIC ECLIPSE		
AP/130		
CS Series		
Mapped NOVA		
Unmapped NOVA		
microNOVA		
Other (Specify) _____		
_____		

3. Software

- ☐ AOS
- ☐ DOS
- ☐ MP/OS
- ☐ RDOS
- ☐ Other

Specify _____

4. Languages

- ☐ Algol
- ☐ DG/L
- ☐ Cobol
- ☐ PASCAL
- ☐ Business BASIC
- ☐ BASIC
- ☐ Assembler
- ☐ Fortran
- ☐ RPG II
- ☐ PL/1
- ☐ Other

Specify _____

5. Mode of Operation

- ☐ Batch (Central)
- ☐ Batch (Via RJE)
- ☐ On-Line Interactive

6. Communications

- ☐ HASP
- ☐ RJE80
- ☐ RCX 70
- ☐ CAM
- ☐ XODIAC
- ☐ Other

Specify _____

7. Application Description

○ _____

8. Purchase

From whom was your machine(s) purchased?

- ☐ **Data General Corp.**
- ☐ Other

Specify _____

9. Users Group

Are you interested in joining a special interest or regional Data General Users Group?

○ _____

Data General

Data General Corporation, Westboro, Massachusetts 01580. (617) 366-8911

FOLD FOLD

TAPE TAPE

FOLD FOLD

NO POSTAGE
NECESSARY
IF MAILED
IN THE
UNITED STATES

BUSINESS REPLY MAIL
FIRST CLASS PERMIT NO. 26 SOUTHBORO, MA. 01772

Postage will be paid by addressee:

Data General

ATTN: Users Group Coordinator (C-228)
4400 Computer Drive
Westboro, MA 01581

Data General

Data General Corporation, Westboro, Massachusetts 01580

069-4002