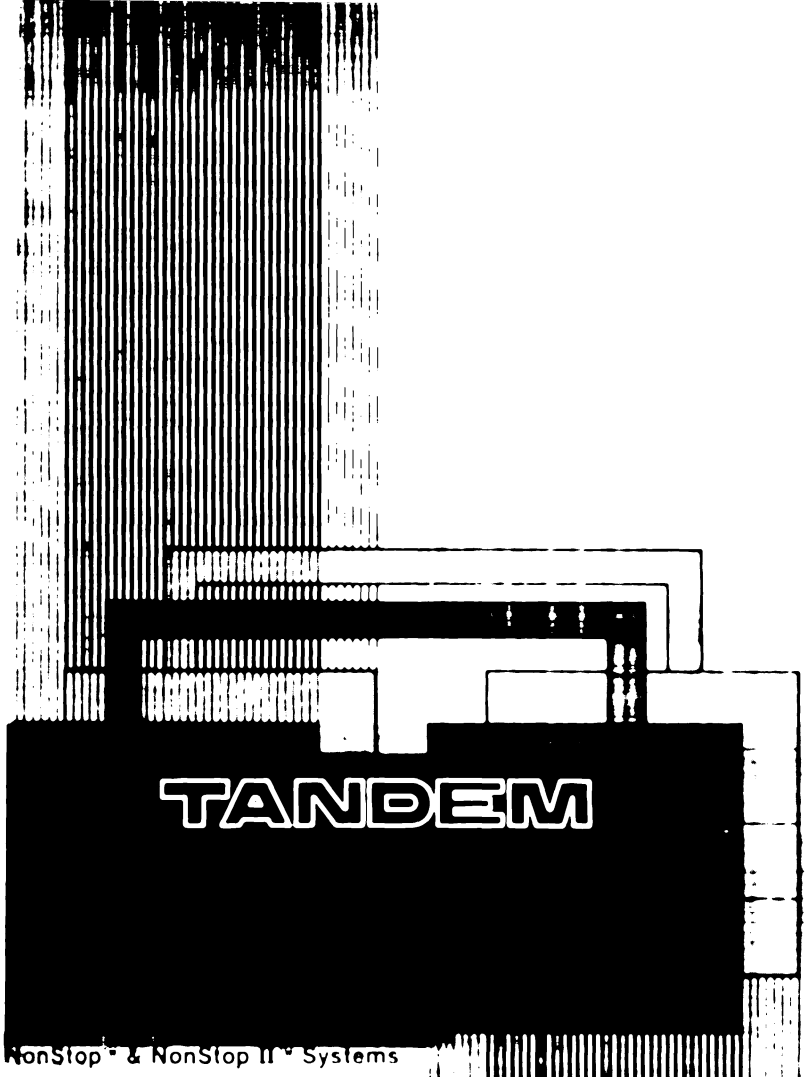


BINDER™
User's Manual

82314

Tandem NonStop (TM) and NonStop II (TM) Systems

BINDER (TM) User's Manual

ABSTRACT: Describes BINDER operations and commands needed to manipulate object files.

PRODUCT VERSION: BINDER A02

OPERATING SYSTEM VERSION: GUARDIAN A06 (NonStop II systems)
GUARDIAN E07 (NonStop systems)

Throughout this document, all references to "NonStop II systems" indicate the software that runs on Tandem NonStop II processors and/or NonStop TXP processors.

Tandem Computers Incorporated
19333 Vallco Parkway
Cupertino, California 95014-2599

Part No. 82314 B00

December 1983
Printed in U.S.A.

DOCUMENT HISTORY

Edition	Part Number	Operating System Version	Date
1st Edition	82314 A00	GUARDIAN A04/E05	October 1982
2nd Edition	82314 B00	GUARDIAN A06/E07	December 1983

New editions incorporate all updates issued since the previous edition. Update packages, which are issued between editions, contain additional and replacement pages that you should merge into the most recent edition of the manual.

The following are trademarks or servicemarks of Tandem Computers Incorporated:

AXCESS	BINDER	CROSSREF	DDL	DYNABUS
EDIT	ENABLE	ENCOMPASS	ENCORE	ENFORM
ENSCRIBE	ENTRY	ENTRY520	ENVOY	EXCHANGE
EXPAND	FOX	GUARDIAN	INSPECT	NonStop
NonStop 1+	NonStop II	NonStop TXP	PATHWAY	PERUSE
SNAX	Tandem	TAL	TGAL	THL
TIL	TMF	TRANSFER	XRAY	XREF

INFOSAT is a trademark in which both Tandem and American Satellite have rights.

HYPERchannel is a trademark of Network Systems Corporation.

IBM is a registered trademark of International Business Machines Corporation.

New And Changed Information

This manual is the second edition of the BINDER User's Manual. It includes the following changes:

- Section 1, "Overview," has minor technical corrections as well as editorial changes.
- Section 2, "Running BINDER," reflects certain enhancements to BINDER, minor technical corrections, and editorial changes.
- Section 3, "BINDER Commands," reflects enhancements to the BINDER command set, including the new commands CHANGE, RENAME, and VERIFY and new options for the DUMP, LIST, MODIFY, and SET commands; minor technical corrections; and editorial changes.
- Section 4, "BINDER Operation", has minor technical corrections and editorial changes.
- Appendix A, "Basic Commands," is unchanged.
- Appendix B, "BINDER Error Messages," incorporates new error messages and minor technical corrections.
- Appendix C, "Syntax Summary for Session Commands," reflects the enhancements to the BINDER command set documented in Section 3.
- Appendix D, "NonStop II User-Library Information," has minor technical corrections.

CONTENTS

PREFACE.....ix

SECTION 1. OVERVIEW.....1-1
BINDER'S PRIMARY FUNCTION.....1-1
HOW THE BINDER WORKS.....1-2
Compile-time Binding.....1-2
Command-driven Binding.....1-4
DEFINING THE TARGET FILE.....1-5
Ordering the Target File.....1-6
Specifying Input File Names.....1-7
Major Commands.....1-8
EXAMPLES.....1-9
Binding Separately Compiled Object Files.....1-9
Procedure Replacement.....1-10
Specifying Output Listings.....1-11
BINDER'S RELATION TO UPDATE.....1-12
BINDER'S RELATION TO INSPECT AND CROSSREF.....1-13

SECTION 2. RUNNING BINDER.....2-1
BIND COMMAND.....2-2
Interactive Mode.....2-3
Noninteractive Mode.....2-3
PRINTED OUTPUT.....2-4
Statistics.....2-4
Load Maps.....2-6
Entry Point Load Map.....2-6
Data Block Load Map.....2-8
Cross-reference Lists.....2-10

SECTION 3. BINDER COMMANDS.....3-1
Syntax Conventions for Name Lists as Command Elements.....3-3
ADD Command.....3-6
ALTER Command.....3-9
BUILD Command.....3-11
CHANGE Command.....3-14
CLEAR Command.....3-16
COMMENT Command.....3-16

DELETE Command....3-17
DUMP Command....3-18
FILE Command....3-20
INFO Command....3-21
LIST Command....3-24
MODIFY Command....3-27
MOVE Command....3-30
RENAME Command....3-32
REPLACE Command....3-33
RESELECT Command....3-35
RESET Command....3-36
SATISFY Command....3-38
SELECT Command....3-40
SET Command....3-44
SHOW Command....3-47
STRIP Command....3-51
VERIFY Command....3-50

SECTION 4. BINDER OPERATION....4-1
OBJECT FILE STRUCTURE....4-1
Code Blocks....4-2
Entry Point Names....4-2
Code Block Attributes....4-3
Data Blocks....4-3
External Data References....4-4
Types of Data Blocks....4-4
Block Naming Conventions....4-4
OBJECT FILE FORMAT....4-5
Header....4-5
Code Region....4-7
User Code....4-7
PEP and XEP....4-7
Data Region....4-8
INSPECT Region....4-8
BINDER Region....4-8
INPUT STAGE....4-9
Include Lists....4-10
Include Code Block List....4-10
Include Data Block List....4-11
Include Entry Point List....4-11
Omit List....4-11
Refer List....4-12
Search List....4-12
Unresolved Reference Lists....4-13
Modify List....4-13
OUTPUT STAGE....4-14
Target File Characteristics....4-14
Target File Construction....4-15
MIXED LANGUAGE BINDING....4-17
COBOL and FORTRAN....4-17
TAL with COBOL and FORTRAN....4-18
Example of a COBOL MAIN Skeleton Program Unit....4-18

APPENDIX A. BASIC COMMANDS....A-1
FILE NAME EXPANSION....A-1
Expanded Disc File Names....A-1
Process and Device Names....A-2
Command Entry....A-2
BASIC COMMAND DESCRIPTIONS....A-3
ENV Command....A-3
EXIT Command....A-3
FC Command....A-3
HELP Command....A-4
LOG Command....A-5
OBEY Command....A-6
OUT Command....A-7
SYSTEM Command....A-8
VOLUME Command....A-8

APPENDIX B. BINDER ERROR MESSAGES....B-1

APPENDIX C. SYNTAX SUMMARY FOR SESSION COMMANDS....C-1

APPENDIX D. NONSTOP II USER-LIBRARY INFORMATION....D-1
GUARDIAN BINDING OF USER-LIBRARY PROCEDURES....D-1
OBJECT FILE FORMAT....D-2
PREVENTING BINDER RESOLUTION OF LIBRARY CALLS....D-2
Compile-time Binding....D-2
Command-driven Binding....D-2
PROGRAM FILE USE OF LIBRARIES....D-3
LIBRARY FILE RESTRICTIONS....D-3
LIBRARY PROCEDURE DATA....D-4

FIGURES

1-1. External References....1-2
1-2. Compile-time Binding....1-3
1-3. Using BINDER in Command-driven Mode....1-5
1-4. BINDER's Major Lists....1-7
1-5. Procedure Replacement....1-10
2-1. Object File Statistics....2-5
2-2. Alphabetic Load Map for Code Blocks....2-7
2-3. Alphabetic Load Map for Data Blocks....2-9
2-4. Entry Point Cross-reference Listing (FORTRAN)....2-11
2-5. Entry Point Cross-reference Listing (COBOL and TAL)....2-11
2-6. Data Block Cross-reference Listing (TAL)....2-12
4-1. Example of BINDER's Object File Format....4-6

TABLES

1-1. Commonly Used BINDER Commands.....1-8
3-1. BINDER Session Commands.....3-1
3-2. Syntax Conventions for Name Lists.....3-4
4-1. Source Language Names for Blocks.....4-2
4-2. Code Block Attributes.....4-3
4-3. Commands to Create Control Lists.....4-9

PREFACE

This book documents the features of BINDER, one of the program development tools for NonStop 1+ and NonStop II software.

BINDER builds all object files for the COBOL, FORTRAN, and TAL compilers. It can also operate as an independent process when providing object file update and link functions in interactive mode.

This book primarily describes the use of BINDER as a standalone program. Each language manual describes the use of BINDER with the compilers.

Intended users are system and application programmers who want to exercise explicit control over object file manipulation.

The organization of this book is as follows:

- Section 1 provides an introduction to BINDER and its commands. This section should provide sufficient information for users who want only a subset of BINDER functions.
- Section 2 gives information on using BINDER in command-driven mode. This section also describes output listings.
- Section 3 provides syntax descriptions and examples of the BINDER commands.
- Section 4 describes the format of object files and the block naming conventions used by the BINDER. It also describes both stages of BINDER operation and mixed-language binding.
- Appendix A describes those commands commonly supported by Tandem software that can be invoked from BINDER.
- Appendix B lists error messages that may occur during BINDER operation.
- Appendix C provides a syntax summary.
- Appendix D provides information about the implementation of user libraries with BINDER.

PREREQUISITES

This book assumes user knowledge of at least one of these languages: COBOL, FORTRAN, or TAL.

In addition, familiarity with editing, compiling, and running programs in the Tandem environment is essential.

The following manuals cover the requirements:

- Introduction to Tandem NonStop Computer Systems
- COBOL Programming Manual
- FORTRAN 77 Reference Manual
- Transaction Application Language Reference Manual
- EDIT Manual
- GUARDIAN Operating System Command Language and Utilities Manual

In addition, the following books can be helpful:

- Tandem NonStop II System Description Manual
- Tandem NonStop System Description Manual

SYNTAX CONVENTIONS IN THIS MANUAL

The following is a summary of the conventions used in the syntax notation in this manual.

Notation	Meaning
UPPERCASE LETTERS	Uppercase letters represent keywords and reserved words; you must enter these items exactly as shown.
lowercase letters	Lowercase letters represent variables that you must supply.
Brackets []	Brackets enclose optional syntax items. A vertically aligned group of items enclosed in brackets represents a list of selections from which you may choose one or none.
Braces {}	Braces enclose required syntax items. A vertically aligned group of items enclosed in braces represents a list of selections from which you must choose only one.
Ellipsis	An ellipsis immediately following a pair of brackets or braces indicates that you can repeat the syntax items, enclosed within the brackets or braces, any number of times.
Punctuation	All punctuation marks and symbols not described above must be entered precisely as shown. If punctuation mark or symbol appears enclosed in quotation marks, it is not a syntax descriptor; it is a required character, and you must enter it as shown.

SECTION 1

OVERVIEW

The BINDER program development tool allows you to examine, modify, and combine object files written in COBOL, FORTRAN, or TAL. BINDER and the code it builds run on GUARDIAN operating systems starting with these releases:

- NonStop II system, release A04
- NonStop 1+ system, release E05.

BINDER'S PRIMARY FUNCTION

Procedures in source code programs often contain references to other procedures. These are known as external references.

A major part of the binding process is resolving external references, that is, locating entry points in other object files if they exist. BINDER does external-reference resolution by including a copy of any compiled code and data required by the reference. This is called satisfying the reference.

If the procedure referred to isn't available, the entry point reference is not satisfied. In the example shown in Figure 1-1, Program B can still be in the planning stages when Program A is compiled. In this case, the reference to Procedure Z remains unsatisfied.

As in Figure 1-1, the references can be to procedure entry points that are in the same source file, or they can be to a library of procedures shared by many users. In either case, the referenced entry point names are considered external to the procedure.

If an object file does satisfy the reference, BINDER can copy the referenced code and data into the new object file. If the referenced code isn't found in a file, BINDER leaves the entry point reference for later resolution. Resolution can occur later in another binding operation or in a call that isn't satisfied until program load time.

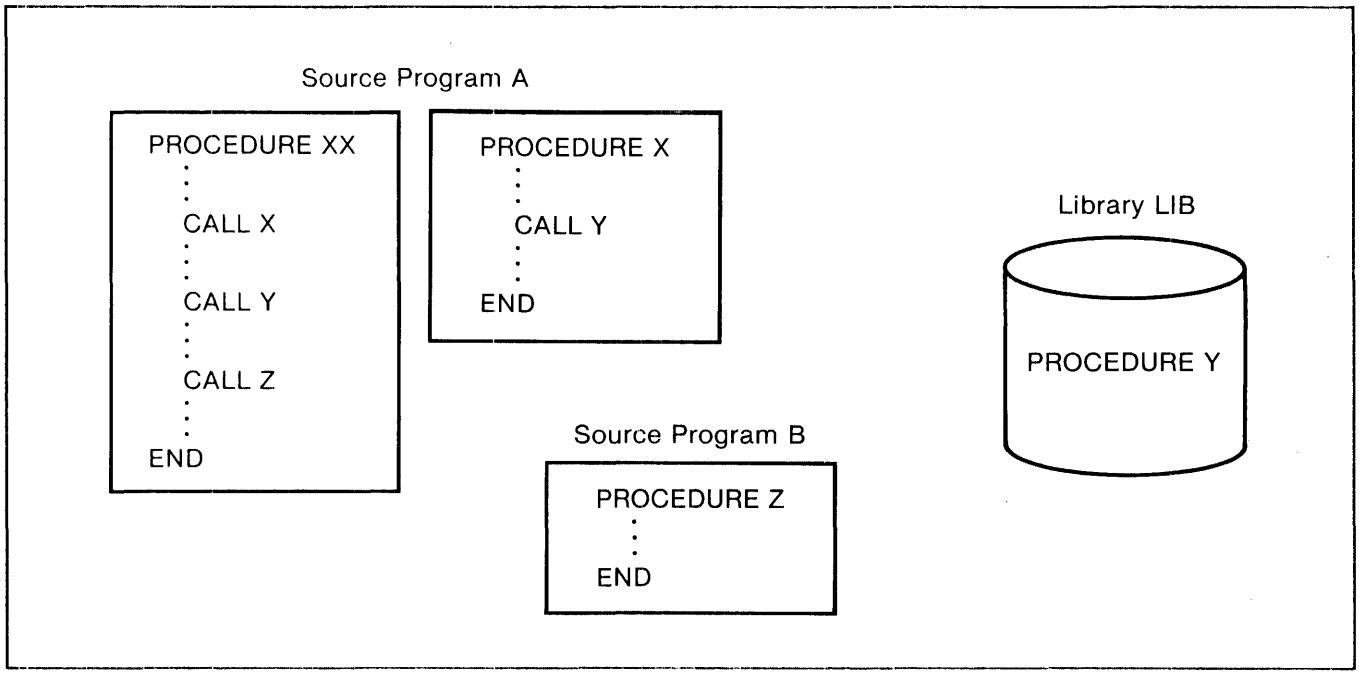


Figure 1-1. External References

HOW THE BINDER WORKS

BINDER operates as either of two processes:

- BINSERV, a process that builds object files in cooperation with the compiler processes; BINSERV's output is controlled by compiler directives; or
- BIND, an independent process that is command-driven.

In either case, both the input and the output files are in standard object file format. This allows you to choose the language that best suits your application and to link program units written in different languages to form an application. For example, a server program of COBOL MAIN can have calculations in FORTRAN subprograms and use TAL procedures for recursive functions, block moves, and scans of data.

Compile-time Binding

All object files that BINDER manipulates originate either from the compile-time process BINSERV or from previous BIND sessions. The new object files produced can then serve as input to further binding operations, either by BINSERV or BIND. Because of this interrelation, it is helpful to be familiar with compile-time binding.

Before the release of BINDER, the COBOL, FORTRAN, and TAL compilers each built their object files in a unique format. Now, BINSERV has been integrated with all three compilers, producing object files in a new standard format. To use BINDER's interactive link and update functions, you must recompile older programs.

BINSERV's action with the compilers is transparent to users, as shown in Figure 1-2.

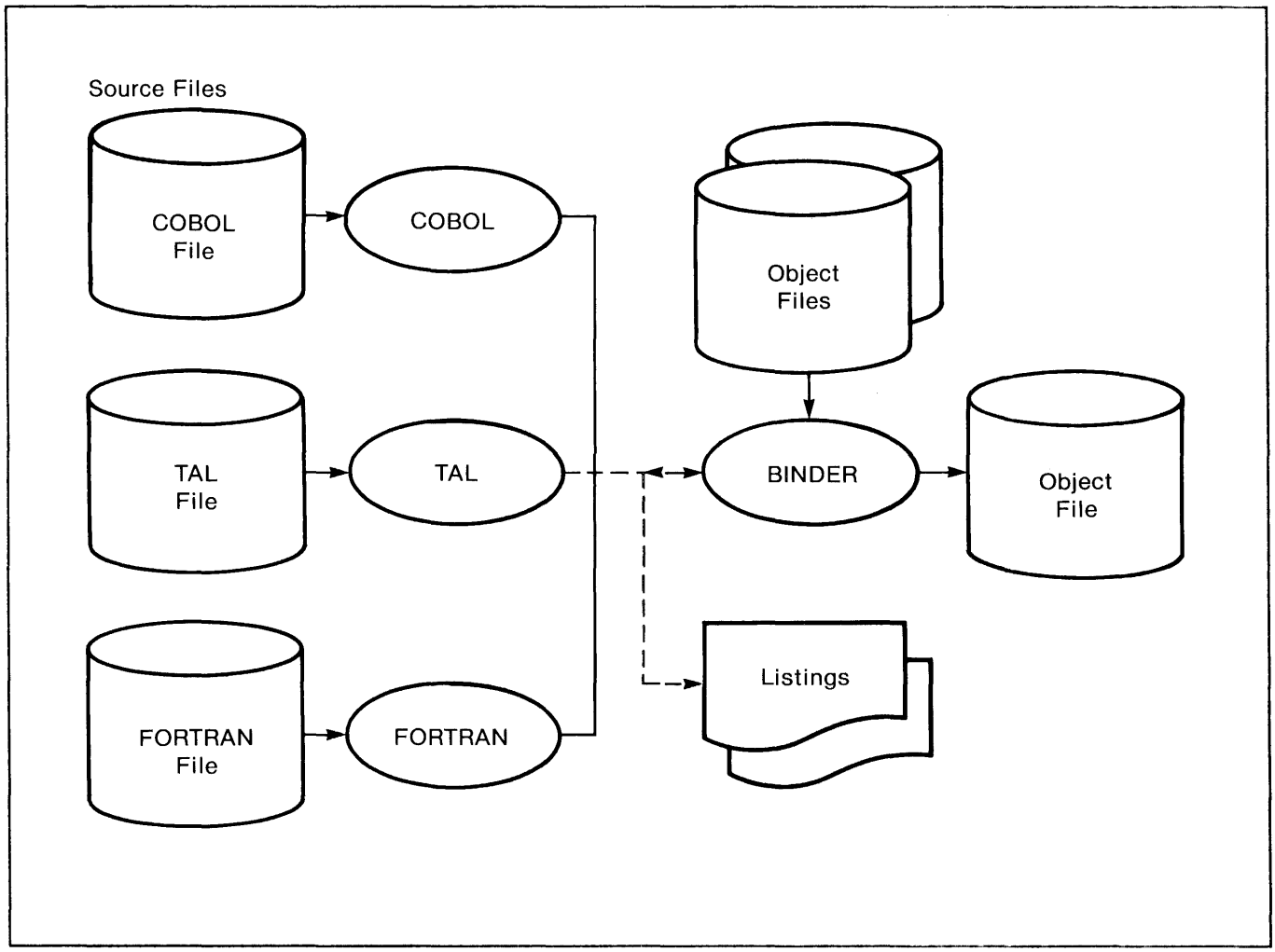


Figure 1-2. Compile-time Binding

The familiar compile-and-run command sequence has the same effect as when the compilers built object files. For example, the following commands behave exactly as before:

```
COBOL /IN mainsrc, OUT listfile/cobsrvr
RUN cobsrvr
```

Command-driven Binding

Both the input files manipulated by BINDER and the output file built by BINDER are object files. To distinguish between the two, the output file is called a target file.

BINDER allows you to perform any of the following functions:

- build a target file from separate object files
- build a target file for use as a library of sharable procedures
- display object file contents
- modify the target file either by patching code or data or by replacing complete blocks
- request consistency checks for parameters or for common data blocks
- reorder code blocks in a target file
- specify external references that should remain unresolved
- produce optional load maps and cross-reference listings
- reduce the size of object files by stripping them of BINDER information and symbol tables
- specify a user run-time library for an object file to be run on a NonStop II system.

Figure 1-3 illustrates BINDER operation in command-driven mode.

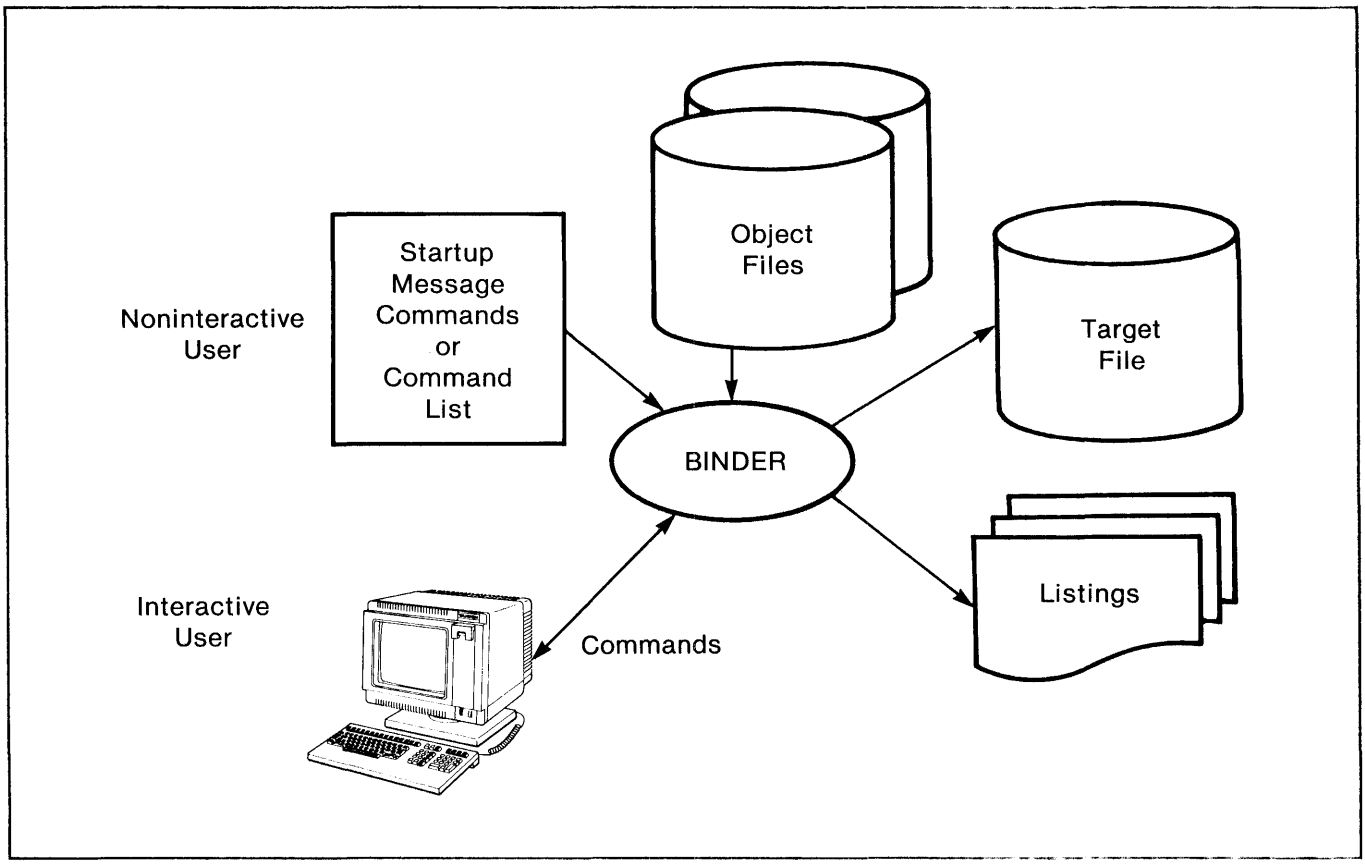


Figure 1-3. Using BINDER in Command-driven Mode

DEFINING THE TARGET FILE

BINDER commands specify the contents of the target file by giving the names of code blocks and data blocks to include. If modifications to the code or data blocks are specified, the changes are made in the target file only.

The concept of a "block" is fundamental to the BINDER. Blocks are the smallest units that can be separately relocated. Both code instructions and data are organized into blocks.

Some block names are directly derived from the source code; for example, a COBOL program unit with a PROGRAM-ID of NAMEX results in a code block named NAMEX. If it declares data in Working-Storage, the resulting data block is also named NAMEX; if it declares data in the new Extended Storage section, that data block is named NAMEX^. There is also a program unit control block (PUCB) named NAMEX#. To take another example, suppose input to the TAL compiler is as follows:

- first declaration is NAME unit^name
- global data declared without a BLOCK construct
- sharable data declared in BLOCK data^name
- procedure declared as PROC proc^name

The resulting code block is named PROC^NAME. There are two data blocks: #GLOBAL and DATA^NAME. If private data is declared, a private data block called UNIT^NAME (from the initial NAME declaration) is created. If there is secondary storage, a secondary data block called .DATA^NAME is also created.

An analogous situation exists for FORTRAN.

The compilers' output listings show all the block names for code and data. (See the appropriate language reference manual for specifics on block naming.) Section 4 gives comparative information for source language constructs and the resulting blocks. It also discusses object file structure and block naming conventions in each language.

Ordering the Target File

The order in which commands specify code block names can be important. In general, BINDER assumes that the order specified is the preferred order that code blocks should appear in the object file. On the other hand, the ordering of data blocks in the target file cannot be established beforehand.

Running BINDER interactively facilitates display of BINDER's interpretation of your commands. BINDER builds lists of target file specifications from the commands and from input object files. The lists and their contents are:

- Three include lists - one each for the code blocks, data blocks, and entry point names to include in the target file. These lists reflect the order of name entry and the order in which BINDER builds the target file.
- Omit list - contains external references (to entry points) that BINDER is not to resolve, regardless of the names that are included.
- Refer list - holds pairs of entry point names; the first name satisfies references to the second. For example, an entry point to a code stub (that is, a block skeleton) can satisfy references to code that isn't ready.
- Modify list - contains an entry for each patch for a code or data block.

- Two unresolved reference lists - one for entry points and one for data blocks. As you enter each name on an include list, BINDER removes entries from the unresolved reference lists if the entries are satisfied by the new name.

- Search list - holds disc file names of object files to search for unresolved references. Since BINDER searches these files in the order in which you name them, BINDER uses the first block encountered if two files contain versions of the same block.

Not all lists are needed for all uses of BINDER, though at least one include list is required for the binding process. (Normally, an unresolved reference list is present during some part of the session.) Each of these lists is described in greater detail in Section 4.

The lists that are commonly present are shown in Figure 1-4.

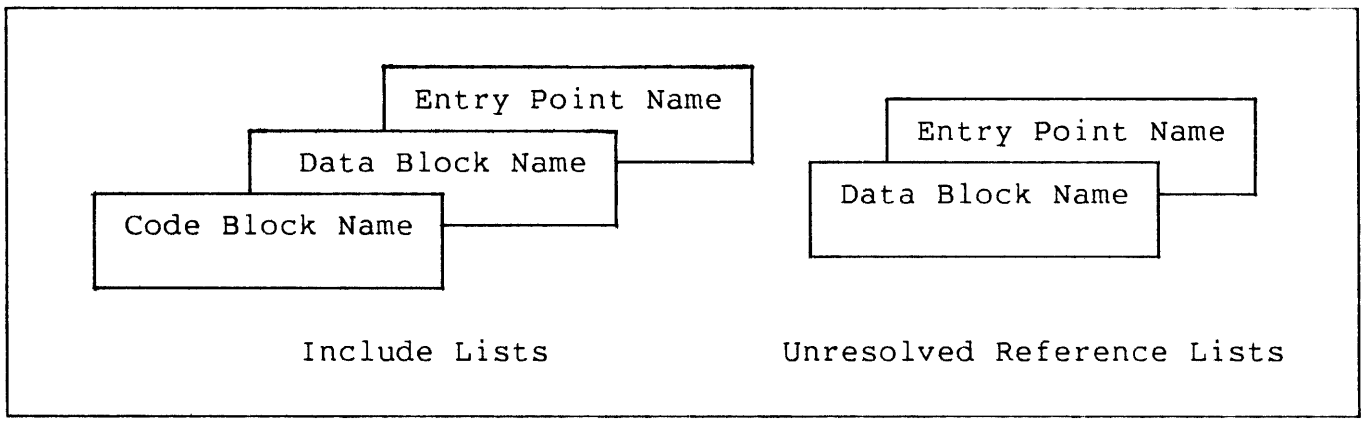


Figure 1-4. BINDER's Major Lists

Specifying Input File Names

BINDER also needs to know where to find the blocks for the include lists. Therefore, you must give the disc file name of the correct object file before BINDER accepts the name for an include list.

The file that BINDER uses as the default for block retrieval is the current file. Either a block must be in the current file or the file must be specified when the block is added. The current file designation can be changed as often as required by using the FILE command.

Major Commands

BINDER commands allow a broad range of object file manipulations. Table 1-1 introduces commands most commonly used.

Table 1-1. Commonly Used BINDER Commands

ADD	names the blocks and entry points to include; replaces blocks and entry points to delete.
BUILD	creates the target file after input is completed.
CLEAR	deletes input information without building a file.
COMMENT	enters text commentary for log listings.
DELETE	removes block names from the include lists.
DUMP	displays object file contents in ASCII, HEX, ICODE, OCTAL, or DECIMAL
FILE	gives a default disc file name for the current file to use for retrieval of code or data blocks.
INFO	displays information about names on the include and unresolved reference lists.
LIST	specifies load maps and object cross-reference listings.
REPLACE	names replacements for code and data blocks already on the include lists.
SELECT	specifies controls for satisfying references or building the target file; also selects BINDER services such as parameter checking or compacting the code area.
SET	sets object file characteristics of the target file.
SHOW	displays the current file name and values in effect for SELECT and SET; also displays the modify list.
STRIP	deletes BINDER and INSPECT information from the object files.

Besides its own command set, BINDER provides the basic commands like OBEY, HELP, and LOG. Automatic file name expansion occurs for all file names. Refer to Appendix A for more information about the basic support.

EXAMPLES

The examples in this section show some common functions and different methods of command entry. (Section 3, however, remains the primary reference for examples of command use.) Command keywords are shown in uppercase, but BINDER accepts lowercase as well for all commands.

The GUARDIAN Command Interpreter starts BINDER in response to a BIND command. As with other "run" commands, you can enter BINDER commands:

- as the BIND command-list
- as the BIND IN command-file, naming a file for command input
- in response to a BINDER prompt (the commercial @).

Binding Separately Compiled Object Files

Suppose two object files contain the code and data blocks needed to make up a large program. Each object file contains multiple program units or procedures, all of which are needed in the target file. (For this discussion, the source language doesn't matter.)

You can bind the separate blocks by running the BINDER and then entering commands in response to the @ prompt.

```
:BIND
@ADD * FROM objfilex
@ADD * FROM objfiley
@BUILD trgfilez
```

"ADD *" causes all the code and data blocks from an object file to be included in the target file. Therefore, if objfilex contains PROC A, PROC B, and PROC D, and objfiley contains PROC C, PROC E, and PROC F, then trgfilez contains all six PROCs.

If more than one object file includes the same code block name or data block name, BINDER retains the first one it encounters and displays a warning message.

Procedure Replacement

This example and Figure 1-5 show replacement of a FORTRAN subprogram in objfilea by a new subprogram in objfileb. In this case, a command file contains the BINDER commands.

```
FORTRAN /IN newsub2, OUT listfile/objfileb
BIND  /IN cmdfile, OUT listfile/
RUN trgfilec
```

cmdfile contains the following BINDER commands:

```
COMMENT - all entries from objfilea are needed
ADD * FROM objfilea
COMMENT - sub2 replaced by new sub2 in objfileb
REPLACE CODE sub2 FROM objfileb
BUILD trgfilec
```

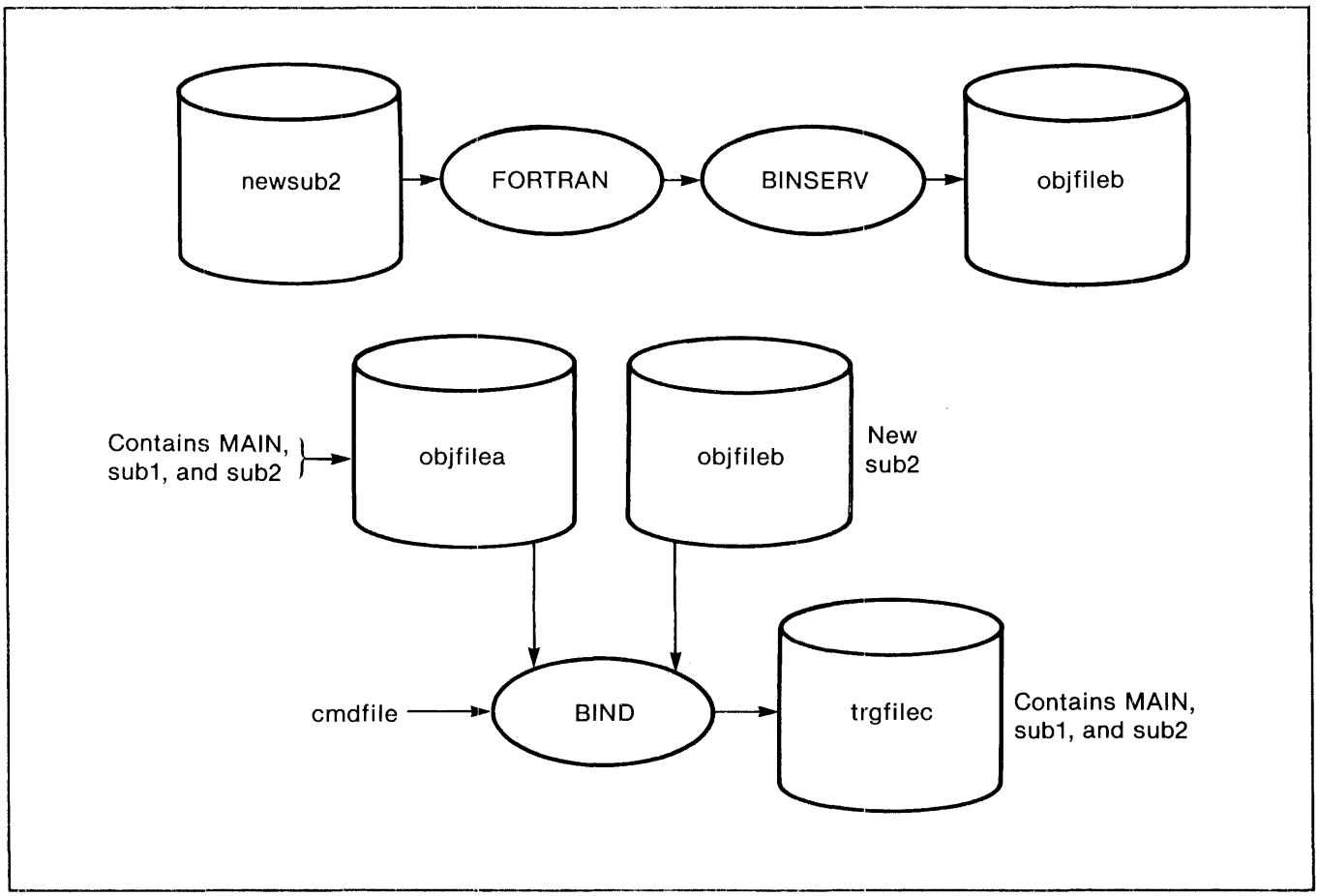


Figure 1-5. Procedure Replacement

Specifying Output Listings

You can use the LIST command (or the LIST option of the SELECT or BUILD commands) to specify load maps in alphabetic or location order. These commands also have options for object file cross-reference information.

Either of the following command lines specifies that BINDER should produce a map of objfilec.

```
:BIND / OUT listfile / FILE objfilec ; LIST LOC

:BIND / OUT listfile / LIST LOC FROM objfilec
```

"LIST LOC" specifies that a block map in location order and a PEP list should be printed to listfile. In this case, the map is created instead of the alphabetic map normally produced for every target file built by BIND.

BINDER'S RELATION TO UPDATE

The UPDATE tool previously provided object file edit and link functions. COBOL and FORTRAN programmers, however, were restricted in its use. The list below gives the BINDER commands corresponding to UPDATE functions.

UPDATE Command	BINDER Command
ADD	ADD *, DELETE (see Note 1 below)
ADD, KEEP	ADD *
ADD procs	ADD CODE procs, DELETE
BUILD file	BUILD file, LIST * OFF, SATISFY OFF
BUILD file, MAP	BUILD file, LIST LOC ON, SATISFY OFF
BUILD file, XREF	BUILD file, LIST XREF ON, SATISFY OFF
DATA pages	SET DATA pages
DEL procs	DELETE CODE procs
DEL q, ABSENT p	SELECT REFER q TO p DELETE CODE q
DUMP proc	DUMP CODE proc
EXIT	EXIT
FILE file	FILE file
LIST	INFO INCLUDE *
MAIN proc	ALTER proc, MAIN ON
MOD proc	MODIFY CODE proc offset, value
MOD *D	MODIFY DATA dataname
SET R, proc	ALTER proc, RESIDENT ON
SET NR, proc	ALTER proc, RESIDENT OFF
SET P, proc	ALTER proc, PRIV ON

NOTE

1) In TAL, when there are no named data blocks, you may want to use ADD CODE *, DELETE in place of ADD *, DELETE. You should also ADD * from the object file containing all the global data needed. (In object files with unnamed data blocks the global data block has the name #GLOBAL. An ADD *, DELETE replaces the #GLOBAL block from the first object file with the #GLOBAL block from the second object file; however, you probably want the #GLOBAL block from the first object file.)

2) Refers will be ignored if the SATISFY OFF parameter of the BUILD command is used.

BINDER'S RELATION TO INSPECT AND CROSSREF

The BINDER, INSPECT, and CROSSREF tools are designed to complement each other in the program development cycle.

INSPECT allows you to debug a program or multiple programs symbolically. That is, you can specify debugging commands using the same symbols you used in the source code. INSPECT must have symbol tables in the object module to allow this feature. The compilers then provide the symbol information to BINDER at compile time. Refer to the INSPECT Users Manual for a description of this product.

You can make incremental changes to code or data blocks in an object file using BINDER. (Source files do not reflect these changes until you recompile.) During an INSPECT session, however, you can only change process data, not code.

To ease the debugging process, CROSSREF provides symbol reference information in the detail that you specify. Note that the cross-reference information available from the BINDER is for object files (entry points and common data blocks only). CROSSREF's listings are source code symbols. Refer to the CROSSREF Users Manual for further information.

Besides the source languages BINDER supports, INSPECT and CROSSREF also support SCREEN COBOL. SCREEN COBOL builds object files in the format needed in PATHWAY's interpretive environment.

SECTION 2

RUNNING BINDER

BINDER requires object files compiled with compatible versions of the COBOL, FORTRAN, or TAL compilers. These are:

- COBOL release E00 or later
- FORTRAN release E00 or later
- TAL release E01 or later.

NOTE

Because of a change in the object file format to support extensible procedures, previous releases of BINDER do not accept object files built by the current BINDER release. The current BINDER, however, supports all object files built by previous versions of BINDER.

BINDER also requires sufficient disc space for temporary files and the target file. If insufficient space exists, BINDER does not retain any of the file-definition tables. BINDER temporary files are placed on the default volume in effect when BINDER is started (or if PARAM SWAPVOL is used, on the volume specified).

Object files built by the compile-time BINDER require additional space when compared with object files built by previous releases of the compilers. The additional space contains BINDER tables and, optionally, symbol tables used for symbolic debugging. As a guideline, allowing an additional 50% of disc space to compile a file using the new compiler versions is sufficient without symbol tables. Allow another 150% of old disc requirement if symbol tables are needed. The symbol tables can be deleted after the debugging cycle is complete. If desired, you can delete both symbol and BINDER tables by using the STRIP command, described in Section 3. You may want to use the BACKUP program to save the files containing these tables before using the STRIP command.

BIND COMMAND

The BIND syntax is standard for starting a process under the GUARDIAN Command Interpreter.

```
BIND [ / [ IN command-file ] [ , OUT list-file ] / ]

  [ command [ ; command ] ... ]
```

IN command-file

specifies a file containing BINDER commands. If this parameter is omitted, BINDER prompts the current input file of the Command Interpreter, normally the home terminal.

OUT list-file

specifies the file to receive output listings. If this parameter is omitted, output is directed to the current output file of the Command Interpreter, normally the home terminal.

command

is any BINDER command. If specified, BINDER executes the commands and terminates without opening or reading the command file.

Considerations

- BINDER complies with the COMINT command PARAM SWAPVOL, creating its work files on the swap volume specified. For a description of the PARAM command, refer to Volume 2 of the GUARDIAN Operating System Programming Manual.
- BINDER complies with the COMINT ASSIGN messages that override such default characteristics of the IN and OUT files as record length. For a description of ASSIGN messages, refer to Volume 2 of the GUARDIAN Operating System Programming Manual.

Interactive Mode

BINDER operates in interactive mode if neither the IN parameter nor a list of commands is included in the COMINT BIND command. BINDER prompts with the commercial at sign @ and continues to do so until an EXIT command is entered.

More than one BINDER command can be entered in response to the prompt. Multiple commands, however, must be separated by semicolons. For example:

```
@ADD * FROM file1; REPLACE sub2 FROM file2
```

is equivalent to:

```
@ADD * FROM file1
@REPLACE sub2 FROM file2
```

Noninteractive Mode

If commands are specified by means of an IN file, BINDER executes the listed commands, terminates, and returns control to the GUARDIAN Command Interpreter. For example:

```
:BIND /IN building, OUT listfile/
```

causes BINDER to accept and execute the commands in the file named building and to direct the output to the file named listfile. BINDER terminates when end-of-file is reached or the EXIT command is encountered in the building file.

No prompts or error messages are displayed at the home terminal if a command is incorrectly or incompletely specified. Error messages are put in the OUT file.

As with interactive mode, multiple commands separated by semicolons can be entered in a single command line.

PRINTED OUTPUT

Output from BINDER includes warning and error messages (described in Appendix B) and the following listings about the target file that accompany successful completion of a BUILD command:

- target file statistics
- load maps in alphabetical order (default)
- load maps in location order (optional)
- cross-reference lists for entry points and data blocks (optional)

Examples of these listings are given in Figures 2-1 through 2-6. In addition, output is generated in response to the DUMP, INFO, LIST, and SHOW commands (described in Section 3).

The output from the BUILD command is automatically printed to the list file, if one is specified; otherwise, it is displayed at the home terminal. In interactive mode, use the OUT parameter to capture the output of a single command.

Statistics

After a BUILD command is executed, BINDER automatically produces statistics for the constructed target file. Figure 2-1 shows an example of these statistics.

```
BINDER - OBJECT FILE BINDER - T9621A02 - (01DEC83) SYSTEM \ANY

Object file name is $SVOLL.PRIVATH.OBJECT
User library file name is $SVOLL.PRIVATH.LIB
Number of Binder errors = 0
Number of Binder warnings = 1
Code size = 14642 words      (See item 6 below.)
  Gap at 32K = 0 words
  PEP = 28 words
  Procedures = 14614 words
  XEP = 49 words
Primary data = 3 words
Secondary data = 847 words
Code area size = 15 pages
Data area size = 4  pages
```

Figure 2-1. Object File Statistics

From the top down, the statistics display provides the following information:

1. the BINDER banner
2. name of the constructed object file
3. name of the object file to be used as an execution-time user library, if one exists
4. number of error messages issued
5. number of warning messages issued
6. code size total in number of words; itemized listing for the gap size at the 32K boundary and, if applicable, for: procedure entry point (PEP) size, size of global read-only arrays, and storage occupied by code blocks. Note that the code size total does not include the XEP size.
7. external entry point (XEP) size
8. number of words of primary data space
9. number of words of secondary data space
10. number of pages required for code area allocation
11. minimum number of pages required for data area allocation.

Load Maps

Separate load maps are produced for entry points and data blocks. BINDER produces alphabetic maps by default. Using the LIST, BUILD...LIST or SELECT LIST commands, you can specify maps ordered by location in addition to, or instead of the alphabetic maps. (Lines are folded if the output line length is less than 132.)

ENTRY POINT LOAD MAP. The following information is included in load maps for entry points.

Label	Description
PEP	PEP table number
BASE	base address of the code block defining the entry point
LIMIT	end address of the code block defining the entry point
ENTRY	address of the entry point
ATTRS	attributes of the entry point (main, privileged, callable, resident, interrupt, and variable number of arguments)
NAME	entry point name
DATE/TIME	timestamp for compilation of the block
LANGUAGE	source language of the block: TAL, COBOL, or FORTRAN
SOURCE FILE	disc file name of source code for the block

Figure 2-2 illustrates a load map for an alphabetically ordered code block.

```
ENTRY POINT MAP BY NAME

PEP    BASE      LIMIT     ENTRY    ATTRS      NAME
                    DATE        TIME    LANGUAGE    SOURCE FILE
005    000454    001371    000470             ATTEMPTPTDISPLAY
                    10/01/82   15:36   COBOL        $LNG.RPDT.QNCBA
002    000006    000040    000014   M         COBOL-QUEENS-SOLUTION
                    10/01/82   16:09   COBOL        $LNG.RPDT.QNCBM
004    000212    000453    000220             SOLUTIONDISPLAY
                    10/01/82   15:56   COBOL        $LNG.RPDT.QNCBS
003    000042    000211    000202             SOLUTION^IN^TAL
                    10/01/82   16:06   TAL          $LNG.RPDT.QNTLS
```

Figure 2-2. Alphabetic Load Map for Code Blocks

DATA BLOCK LOAD MAP. The following information is included in load maps for data blocks.

Label	Description
BASE	base address of the data block (a word address, even for COBOL Extended Storage)
LIMIT	end address of the data block (a word address, even for COBOL Extended Storage; if blank, the block is empty)
TYPE	type of the data block (own, common, or special)
MODE	word or string
NAME	name of the data block
DATE	date of the compilation
TIME	timestamp for the compilation
LANGUAGE	source language of the block
SOURCE FILE	disc file name of the source code

Figure 2-3 illustrates a load map for a data block ordered alphabetically. Note the larger base and limit fields of the final entry, indicating that it is a COBOL extended data block.

```
DATA BLOCK MAP BY NAME
 BASE     LIMIT     TYPE       MODE      NAME
                      DATE     TIME       LANGUAGE    SOURCE FILE
000000    000017    SPECIAL    WORD      #G0
                     8/31/83   09:26      COBOL      $LNG.RPDT.CBLSKEL
100254              SPECIAL    WORD      #HIGHBUF
                     8/31/83   09:26      COBOL      $LNG.RPDT.CBLSKEL
100000    100114    SPECIAL    WORD      #RUCB
                     8/31/83   09:26     COBOL       $LNG.RPDT.CBLSKEL
000061    000071    SPECIAL    WORD      #STACK
                     8/31/83   09:26     COBOL       $LNG.RPDT.CBLSKEL
000020    000060    OWN        STRING    ATTEMPTDISPLAY
                     8/31/83   09:26     COBOL       $LNG.RPDT.CBLSKEL
100115    100253    SPECIAL    WORD      ATTEMPTDISPLAY#
                     8/31/83   09:26     COBOL       $LNG.RPDT.CBLSKEL
0001000000  0001000307   OWN   STRING    CBLSKEL^
                     8/31/83   09:26     COBOL       $LNG.RPDT.CBLSKEL
```

Figure 2-3. Alphabetic Load Map for Data Blocks

Cross-reference Lists

The cross-reference lists are produced only if explicitly selected by the LIST XREF command. Cross-reference lists are generated for both entry points and data blocks. The entry point listing consists of:

- entry point name
- name of the code block containing the reference
- location of each reference.

Figures 2-4 and 2-5 show partial examples of entry point cross-reference lists.

```
ENTRY POINT CROSS REFERENCE

REFERENCED ENTRY POINT     REFERENCING CODE BLOCK   WORD OFFSET
                                                    OF REFERENCES

FORMATCONVERT
                           FORMATTER                00305
FORMATEDIT
                           FORMATTER                02003 02065
                                                    07644
FORMATNUMIN
                           FORMATNUMIN              00150
                           FORMATTER                00466 00571
                                                    00627 01213
                                                    07204
FORMATTER^
                           PRICE                    00125 00135
                                                    00155 00174
```

Figure 2-4. Entry Point Cross-reference Listing (FORTRAN)

```
ENTRY POINT CROSS REFERENCE

REFERENCED ENTRY POINT     REFERENCING CODE BLOCK   WORD OFFSET
                                                    OF REFERENCES

ATTEMPTDISPLAY             SOLUTION^IN^TAL          00110
CLIB^DISPLAY               ATTEMPTDISPLAY           00117
                           SOLUTIONDISPLAY          00230
CLIB^INIT                  COBOL-QUEENS-SOLUTION    00006
CLIB^STOP                  COBOL-QUEENS-SOLUTION    00031 00032
SOLUTIONDISPLAY            SOLUTION^IN^TAL          00064
SOLUTION^IN^TAL            COBOL-QUEENS-SOLUTION    00030
```

Figure 2-5. Entry Point Cross-reference Listing (COBOL and TAL)

The data block cross-reference listing consists of:

- data block name (either a common block name or the TAL special block, #GLOBAL)
- location and storage type (word, byte) of referenced identifier
- name of each code block containing block references
- location of the reference.

Locations are word offsets in octal from the base of the block.

```
COMMON BLOCK CROSS REFERENCE

REFERENCED COMMON BLOCK   OFFSET TYPE  REFERENCING      WORD OFFSET
                                       CODE BLOCK       OF REFERENCES

APPLY^DATA          000000 000000 WORD  APPLY            00000 00047
                                                        00573 00637
                                                        00733 00736
APPLY^PUBLIC        000000 000000 WORD  IN^INITIALIZE    00516
                    000000 000001 WORD  AP^PERFORM       00357
                                        AP^REDIRECT      00214
                                        AP^START^LOG     00157
                    000000 000002 WORD  APPLY            01305
                                        AP^START^LOG     00112 00120
```

Figure 2-6. Data Block Cross-reference Listing (TAL)

SECTION 3

BINDER COMMANDS

BINDER commands select the object files to search for input, define and build the target file, and query the status of BINDER's controls.

This section describes the BINDER commands in alphabetic order. The basic commands that are supported by BINDER are described in Appendix A.

Table 3-1 is a summary of all BINDER commands, whether they are described in this section or in Appendix A.

Table 3-1. BINDER Session Commands

Command	Command Description
ADD	inserts new names or replaces old names on the include lists; replaced names are deleted.
ALTER	changes attributes of entry points.
BUILD	creates the target file.
CHANGE	patches the attribute values in an already created object file.
CLEAR	returns BINDER to the original state without creating the target file.
COMMENT	enters comments to appear in output listing.
DELETE	removes names from include lists and removes associated changes from the modify list.

→

DUMP	displays all or part of the contents of a code or data block.
ENV	displays the current settings of process environment controls.
EXIT	stops the BINDER process.
FC	edits or repeats the previous command line.
FILE	sets the default object file for the ADD, DUMP, LIST, and REPLACE commands.
HELP	displays the BINDER commands and syntax.
INFO	displays information about code blocks, entry points, and data blocks in the include and unresolved reference lists.
LIST	specifies options for load maps and entry point cross-reference listings.
LOG	starts or stops the recording of BINDER input commands and output.
MODIFY	changes values of code or data block locations in the target file.
MOVE	reorders code blocks in the target file.
OBEY	directs BINDER to read commands from the named disc file.
OUT	names the file to receive output listings.
RENAME	renames a code or data block.
REPLACE	names code or data as replacements if the names are on the include lists.
RESELECT	restores one or more BINDER controls to the original state.
RESET	restores one or more object file attributes to the default values.
SATISFY	attempts immediate resolution of all external references.

→

SELECT	sets options for BINDER operation control.
SET	sets object file characteristics to use in building the target file.
SHOW	displays collected information: current file, modify list, and controls from the SELECT and SET commands.
STRIP	deletes BINDER and INSPECT regions from an object file.
SYSTEM	sets the default system name for expanding file names.
VERIFY	verifies a code or data value in an object file.
VOLUME	sets the default volume and subvolume for expanding disc file names.

Syntax Conventions for Name Lists as Command Elements

Many BINDER commands allow lists of names for entry points, code blocks, or data blocks to be used as part of the command line syntax. Be sure to note the syntax conventions for forming these lists given in Table 3-2.

Within lists, BINDER allows ranges of names, so you can specify ranges of blocks or entry points (on ADD, DELETE, and REPLACE commands, for example).

Name ranges can apply to input object files. If so, the range is the span of blocks or entry points between the first name and the second name. The names must be given in order by location in the file.

You can also specify ranges of names as on include lists (on ALTER, INFO, and MOVE commands, for example). In this case, a range is determined by the order in which the names were entered.

The INFO command displays these include lists. Examples of partial INFO displays follow Table 3-2.

Table 3-2. Syntax Conventions for Name Lists

Element	Definition
code-block	one of: COBOL program unit FORTRAN program or subprogram TAL proc
data-block	one of: COBOL program unit's Working-Storage or Extended-Storage (i.e., its OWN block) FORTRAN COMMON or SAVE DATA TAL BLOCK or implicitly-named global
block-name	valid language identifier for a code block or for a data block
block-name-range	for code or data, one of: block-name block-name TO block-name * TO block-name block-name TO *
block-name-list	for code or data, one of: block-name-range (block-name-range [, block-name-range] ...) *
entry-name	a valid language identifier for a primary or secondary entry point name
entry-name-range	one of: entry-name entry-name TO entry-name * TO entry-name entry-name TO *
entry-name-list	one of: entry-name-range (entry-name-range [, entry-name-range] ...) *
*	all members of the current include list or input object file
name TO name	all members from the first name to the second name of the include list or object file

→

* TO name	all members from the start of the include list or object file to the given name
name TO *	all members from the given name to the end of the include list or object file

Below are examples of partial displays generated by the INFO INCLUDE command. Example 1 shows display for entry points only. Example 2 shows all three include lists; the unresolved reference list is not shown here.

1.
```
@INFO INCLUDE ENTRY *
INCLUDE ENTRY: 26 ENTRIES
 NAME                                    OFFSET   ATTRIB
 PRICE                                       43   M
 FLIB^INIT                                    0
 FORMATTER                                 4450   V
 FORMATTER^                                4454   EV
   .   .   .   .   .   .   .   .   .   .   .   .   .
```

2.
```
@INFO *
INCLUDE CODE: 24 ENTRIES
 NAME                                     SIZE
 PRICE                                     289
 FLIB^INIT                                  26
   .   .   .   .   .   .   .   .   .   .   .   .   .
                                           4672
INCLUDE ENTRY: 26 ENTRIES
 NAME                                    OFFSET   ATTRIB
 PRICE                                       43   M
 FLIB^INIT                                    0
 FORMATTER                                 4450   V
 FORMATTER^                                4454   EV
   .   .   .   .   .   .   .   .   .   .   .   .   .

INCLUDE DATA: 5 ENTRIES
 NAME                                     SIZE
 #G0                                         3
 #LOWBUF                                   608
 #RUCB                                      77
 COMMON#POINTERS                             4
 FOO                                        10
```

ADD Command

The ADD command inserts names in the three include lists: include entry point list, include data block list, and include code block list.

```
ADD { CODE entry-name-list  } [ FROM file-name ] [ , DELETE ]
    { DATA block-name-list  }
    { *                     }
```

CODE entry-name-list

specifies code to be included in the target file. Associated code blocks, entry points, and own blocks are added to the appropriate include lists for each entry point named. Valid forms of entry-name-list are given in Table 3-2.

DATA block-name-list

specifies FORTRAN COMMON blocks or TAL BLOCKs to include in the target file. Valid forms of block-name-list are given in Table 3-2. This parameter is not pertinent to COBOL.

*

specifies that all code and data blocks in the file are added to the applicable include lists.

FROM file-name

is the disc file name of an object file to use. The default is the current file.

DELETE

specifies that any previously inserted occurrences of names added by this command should be deleted.

To ensure that the target file is built correctly, add names to the lists in the order that BINDER should build the target file. BINDER adds names to the include lists in the order that ADD commands specify them and then uses the include list order to build the target file.

When you specify an entry name or data block name for a block not already on the include list, ADD adds the name to the end of the list, whether the DELETE option is specified or not. When you specify an entry name or data block name for a block already on the include list, ADD adds the name to the end of the list only if you also specify the DELETE option. Otherwise, BINDER ignores the ADD command and issues a warning message indicating that the specified entry point or data block is already in the include list.

If a specified entry point or data block is not in the specified FROM file, an error message is issued and execution of the ADD command is halted.

Naming an entry point in an ADD command automatically inserts the containing code block in the include code block list. Usually, data blocks are added implicitly (by means of references in included code). Nevertheless, you can add FORTRAN COMMON blocks or TAL BLOCKs explicitly. This enables you to define a different set of initial values, if desired.

For COBOL files, ADD CODE is equivalent to ADD *, since COBOL files do not have separate data blocks.

ADD results in resolution of all previously unresolved external references that are satisfied by the added entry point or data block. (If the name is on the omit list, external references remain unresolved.)

NOTE

It is useful to remember that the REPLACE command performs functions similar to those performed by ADD,DELETE. (ADD,DELETE adds a specified entry name to the end of the include list, deleting the previous occurrence of the entry; REPLACE removes the previous occurrence of the entry name and inserts the new reference in its place.) If an error occurs when you attempt an ADD command, you may be able to accomplish your goal by using a REPLACE command.

Examples of the ADD command:

1.
```
@FILE oldfile
@ADD *
@ADD CODE block-1 TO block-5 FROM objfile
@COMMENT  newfile contains oldfile and
@COMMENT  block-1 to block-5 of objfile
@BUILD newfile
```

2.
```
@FILE oldfile
@SELECT REFER proc to proc2
@SELECT OMIT proc
@ADD CODE *
@COMMENT  -  UPDATE equivalent is "DEL proc, ABSENT proc2"
@COMMENT  -  after entering the FILE and ADDs
@BUILD newfile
```

3.
```
@FILE oldfile
@ADD * FROM oldfile
@ADD CODE block-1 FROM objfile, DELETE
@BUILD newfile
@COMMENT  -  newfile contains block-1 from objfile and all of
@COMMENT  -  oldfile except any code block named block-1.
```

ALTER Command

The ALTER command changes attributes of code blocks and entry points in both include code block and include entry point lists. Attributes that can be changed are CALLABLE, MAIN, PRIVILEGED, and RESIDENT. MAIN can only be changed for TAL procs. Refer to the TAL Reference Manual description of "Procedure and Subprocedure Declaration" for attribute information.

```
ALTER entry-name-list , alter-spec [ , alter-spec ] ...
```

entry-name-list

identifies one or more primary or secondary entry points on the corresponding include lists. See Table 3-2 for the syntax of list formation.

alter-spec

specifies a code attribute. It is one of:

```
CALLABLE { ON | OFF }
LIKE entry-name
MAIN { ON | OFF }
PRIV { ON | OFF }
RESIDENT { ON | OFF }
```

CALLABLE { ON | OFF }

specifies whether privileged entry points are callable by nonprivileged procedures. Since CALLABLE applies only to privileged code, BINDER automatically sets PRIV ON for a entry point with CALLABLE ON.

LIKE entry-name

specifies that the entry points in entry-name-list are to have the same attributes as LIKE entry-name, which must be on an include list. If LIKE occurs, it overrides any preceding parameters of this ALTER command.

→

MAIN { ON | OFF }

specifies whether the entry points in entry-name-list are to have the MAIN attribute. MAIN can be specified only for TAL code; COBOL and FORTRAN MAIN characteristics are set permanently at compilation time.

PRIV { ON | OFF }

specifies whether entry points in entry-name-list are run in privileged mode. If PRIV ON is set, the procedure can be called only by procedures that also run in privileged mode.

RESIDENT { ON | OFF }

specifies whether code blocks are to reside in main memory during the entire time the process is running. (BINDER automatically applies RESIDENT to the code block containing the named entry point.)

Examples of the ALTER command:

1.
```
@ADD CODE sub^1 FROM objfile1
@ADD CODE sub^2 FROM objfile2
@ALTER sub^1, LIKE sub^2
@COMMENT  -  the attributes of the code block named sub^1 are
@COMMENT  -  changed to match the attributes of sub^2.
```

2.
```
@ALTER * TO sub^5, RESIDENT ON
@COMMENT  -  the RESIDENT attribute is changed to ON for all
@COMMENT  -  code blocks and entry points from the beginning of
@COMMENT  -  the include lists through entry name sub^5.
```

3.
```
@ALTER (sub^1, sub5 TO sub^8), CALLABLE OFF
@COMMENT  -  the CALLABLE attribute is changed to OFF for
@COMMENT  -  entry names sub^1 and sub5 through sub^8 on the
@COMMENT  -  include lists.
```

BUILD Command

The BUILD command is entered after completely defining the target file with other commands. BINDER then constructs the target file using the code block names and data block names from the include lists. After the build, the BINDER returns to the initial state.

```
BUILD [ file-name ] [ ! ]  [, set-specification    ]  ...
                           [, select-specification ]
```

file-name

is a valid file name for the target file. file-name must not previously exist unless the ! option is used. If file-name is omitted, BINDER uses the default file name OBJECT in the default volume and subvolume.

!

specifies that BINDER should purge any previously existing file named file-name.

set-specification

specifies an object file attribute to use for the target file. This specification overrides, for this command only, any value previously established for that attribute. set-specification, as defined for the SET command, is one of:

```
{DATA       }
{STACK      } value [ PAGES | WORDS ]
{EXTENDSTACK}
INSPECT { ON | OFF}
LIBRARY file-name
LIKE file-name
PEP number
SAVEABEND { ON | OFF}
SYMBOLS { ON | OFF}
```

→

select-specification

specifies a BINDER control or option to use in building the object file. This specification overrides, for this command only, any value previously established for that parameter. select-specification, as defined for the SELECT command, is one of:

```
CHECK check-spec
COMPACT { ON | OFF }
LIST list-spec
OMIT entry-name-list
REFER refer-list
SATISFY { ON | OFF}
SEARCH file-list
```

When you enter BUILD with SATISFY ON, you allow BINDER to resolve any remaining external references. In this case, BINDER accesses the user files on the search list and, if it finds any of the unresolved entry points, adds the corresponding code to the target file it is building. Entering BUILD SATISFY OFF prevents BINDER from trying to resolve remaining unresolved references.

BINDER creates the target file in a temporary file and gives the target file one of the following names:

1. file-name, if a file by that name did not exist already
2. file-name, if a file of that name previously existed, ! was specified, and the purge was successful
3. OBJECT, if no file-name was specified (default), or if file-name already existed and was not purged
4. ZZBInnnn, where nnnn is a random numeric identifier, if the naming attempts for file-name and OBJECT fail.

NOTE

If the old object file happens to be running when a new object file of the same name is specified with the ! option, BINDER does not purge the old file but renames it ZZBInnnn (where nnnn is supplied by BINDER).

The build fails if BINDER cannot name the target file. When this happens, BINDER issues an error message and, in interactive mode, prompts for input. The temporary file containing the target file is lost as well as the information from previous commands. You must respecify the target file contents by reentering definition commands. If the build failed because of insufficient disc space, you must correct the condition before the build can succeed. The only error condition that does not cause BUILD to clear the lists (and that, consequently, does not require you to begin again) is ILLEGAL SYNTAX.

Examples of the BUILD command:

1.
```
@COMMENT - in this example, object file "filename"
@COMMENT - is to have code for one entry point "epnamel"
@COMMENT - replaced by code with the same name from newfile
@ADD * FROM filename
@REPLACE CODE epnamel FROM newfile
@BUILD objfile, SEARCH (libl, lib2)
```

2.
```
@COMMENT -  this example is equivalent to UPDATE ADDs and BUILD
@ADD CODE sub^1 TO * FROM objfilel
@ADD CODE * FROM objfile2, DELETE
@COMMENT -  use the SATISFY OFF option to suppress resolution
@COMMENT -  of code external references
@BUILD newfile, SATISFY OFF
```

CHANGE Command

The CHANGE command permits the attribute values of an already created object file to be amended or patched. In this respect, the CHANGE command is analogous to the SET command, which specifies attribute values for a target file before it is built. The file being patched cannot be the current file in BINDER. If you attempt to use CHANGE on the current file, you must then execute a CLEAR command and start again. Otherwise, the file remains the current file and repeated attempts to CHANGE result in the same error.

```
CHANGE { DATA value [ PAGES | WORDS ]      } IN object-file
       { INSPECT  { ON | OFF }             }
       { LIBRARY file-name                 }
       { SAVEABEND  { ON | OFF }           }
```

DATA value [PAGES | WORDS]

specifies the amount of data space to be allocated for the object file. The default data space allocated is the maximum number of data pages in any of the files from which data is included, or the amount needed to hold all the data blocks plus an estimate of the stack space needed for local storage, whichever is larger. Either a decimal value or an octal value (preceded by %) is accepted. The default unit for the value is PAGES. (One PAGE is 1024 WORDS.)

INSPECT { ON | OFF }

specifies whether the INSPECT program or the DEBUG program is chosen for debugging when the object file is executed. The default is OFF; that is, the DEBUG program is used. INSPECT OFF automatically causes BINDER to set SAVEABEND OFF. (The COMINT SET INSPECT and RUN commands both allow overriding the INSPECT option.)

LIBRARY file-name

specifies the name of a NonStop II user library to be associated with the object file at run time. This file name can be overridden at run time by using the LIB parameter in the COMINT RUN command. The default is no user library.

→

SAVEABEND { ON | OFF }

specifies whether a save file is to be created if the process terminates abnormally during execution. BINDER automatically sets INSPECT ON if SAVEABEND ON is on. The default is OFF.

Each successive CHANGE command specifying one of these parameters overrides the previous specification. The SHOW command can be used to determine the current values of the file attributes.

Examples of the CHANGE command:

1. @CHANGE SAVEABEND ON IN object

2. @CHANGE LIBRARY libfile IN object

CLEAR Command

The CLEAR command returns BINDER to the original state without building an object file. The BINDER clears its internal lists (include, omit, refer, search, unresolved reference, and modify) and the current file established by a FILE command.

```
CLEAR
```

COMMENT Command

Use COMMENT to enter descriptive text to appear in the output listing.

```
COMMENT [ text ]

text
  is a string of characters.
```

If COMMENT is entered on a multi-command line, the COMMENT must be the last command on the line.

DELETE Command

The DELETE command removes named blocks from the include lists. If any MODIFY commands were previously specified for these names, the changes are also removed from the modify list.

```
DELETE { CODE block-name-list }
       { DATA block-name-list }
       { *                    }
```

CODE block-name-list

specifies code blocks to be deleted. Valid forms of block-name-list are given in Table 3-2.

DATA block-name-list

specifies data blocks to be deleted. Valid forms of block-name-list are given in Table 3-2.

*

specifies that all blocks are to be deleted.

DELETE has these effects:

- external references to deleted blocks are placed on the unresolved reference lists
- external references from deleted blocks are removed from the unresolved reference lists (only if no other blocks refer to those names)
- parameter checking for deleted blocks is discontinued.

If DELETE * is specified, the SELECT and SET specifications remain in effect. These lists are cleared:

- all include lists
- the modify list
- the unresolved reference list.

DUMP Command

The DUMP command displays all or part of the contents of a code or data block from the current object file. The display is unformatted. (There is no default for the block name.)

```
DUMP  {CODE code-block-name}   {offset  [, count]}
      {DATA data-block-name}   {        [TO *  ]}
                               {*                }

  [ dump-spec-list ] [ FROM file-name ]
```

CODE code-block-name

is a code block in either the FROM file-name or the current file.

DATA data-block-name

is a data block in either the FROM file-name or the current file.

offset

is an offset in octal from the base of the block.

count

specifies in octal the number of words to display starting from offset. The default is one word.

TO *

specifies that the rest of the block starting at offset is to be displayed.

*

specifies that the entire block is to be displayed.

→

dump-spec-list

is one or more format specifiers in the form:

```
dump-spec
( dump-spec [ , dump-spec ] ... )
```

dump-spec is one of:

ASCII
HEX
DECIMAL
ICODE
OCTAL

Use ICODE with the CODE code-block-name parameter only. The default is OCTAL.

FROM file-name

is the disc file name of an object file. The default is the current file.

DUMP displays the current file contents. Therefore, you cannot use DUMP to display file contents changed with a MODIFY command. (MODIFY is only effective for the target file.) DUMP DATA does not give useful information for COBOL data blocks.

Examples of the DUMP command:

1.
```
@COMMENT - dump all of block1
@DUMP CODE block1 * HEX
```

2.
```
@COMMENT - dump 8 words from word 12  of block5
@DUMP DATA block5 14,10 ASCII FROM objfile
```

3.
```
@COMMENT - be sure that the current file corresponds to the dump
@COMMENT - request; in this example, the current file is
@COMMENT - changed.
@FILE file1; ADD CODE block1 TO block5; FILE file2; ADD *
@DUMP CODE block3
 ***** ERROR ***** Block does not exist in file2
@DUMP CODE block3 FROM file1
```

FILE Command

The FILE command establishes the current file for subsequent ADD, DUMP, LIST, and REPLACE commands. The current file remains in effect until another FILE, CLEAR, or BUILD command is successfully executed. There is no default for the current file.

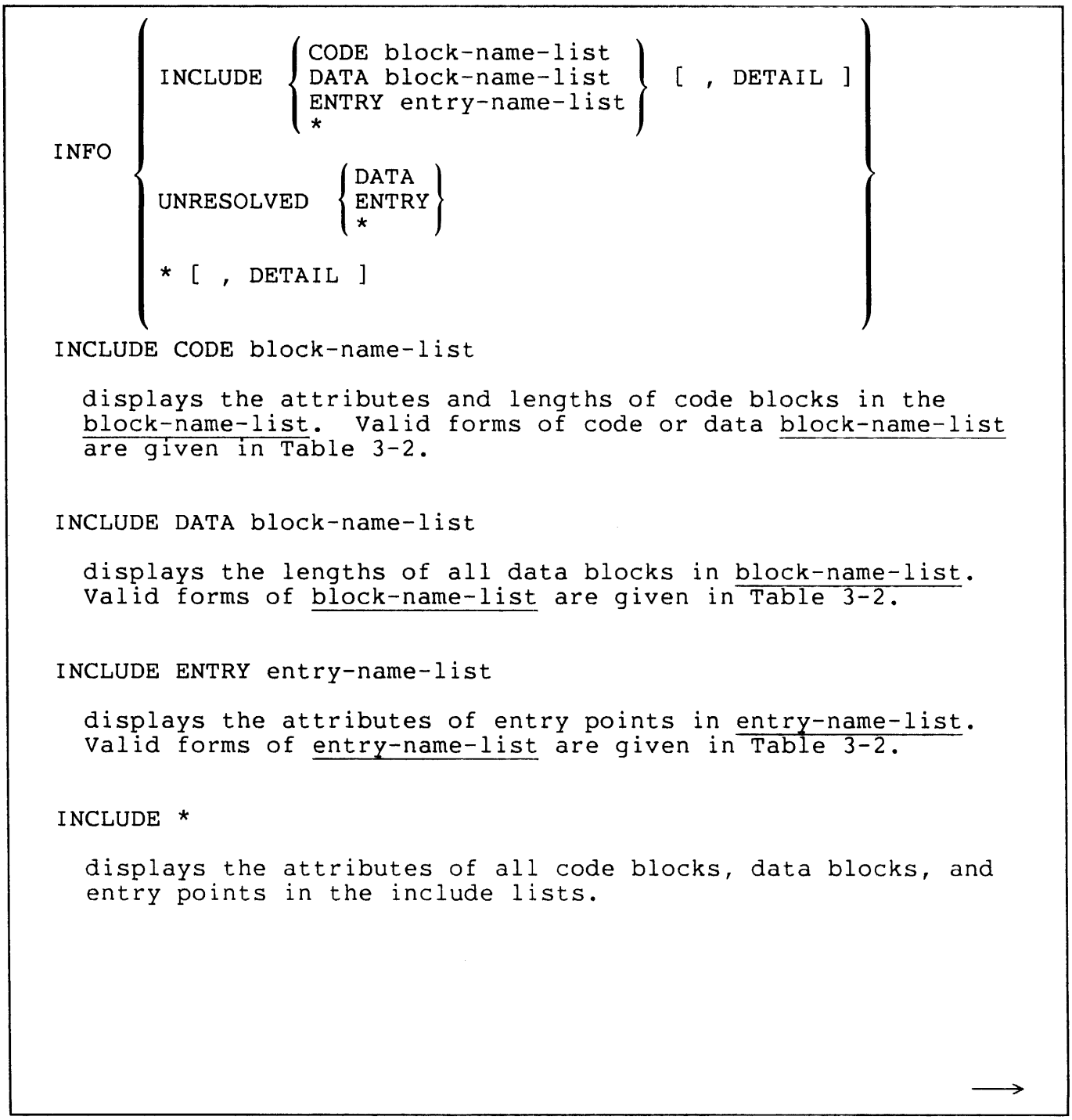
```
FILE file-name

file-name

  is the file name of the object file.
```

INFO Command

The INFO command displays information about code blocks, entry points, and data blocks in the include lists and the unresolved reference lists.



INCLUDE CODE block-name-list

displays the attributes and lengths of code blocks in the block-name-list. Valid forms of code or data block-name-list are given in Table 3-2.

INCLUDE DATA block-name-list

displays the lengths of all data blocks in block-name-list. Valid forms of block-name-list are given in Table 3-2.

INCLUDE ENTRY entry-name-list

displays the attributes of entry points in entry-name-list. Valid forms of entry-name-list are given in Table 3-2.

INCLUDE *

displays the attributes of all code blocks, data blocks, and entry points in the include lists.

→

DETAIL

provides further information for blocks in the include lists such as date and time compiled, source language, and the source file from which the block was compiled.

UNRESOLVED DATA

displays names of data blocks on the unresolved reference list.

UNRESOLVED ENTRY

displays the names and default files of entry points in the unresolved reference list.

UNRESOLVED *

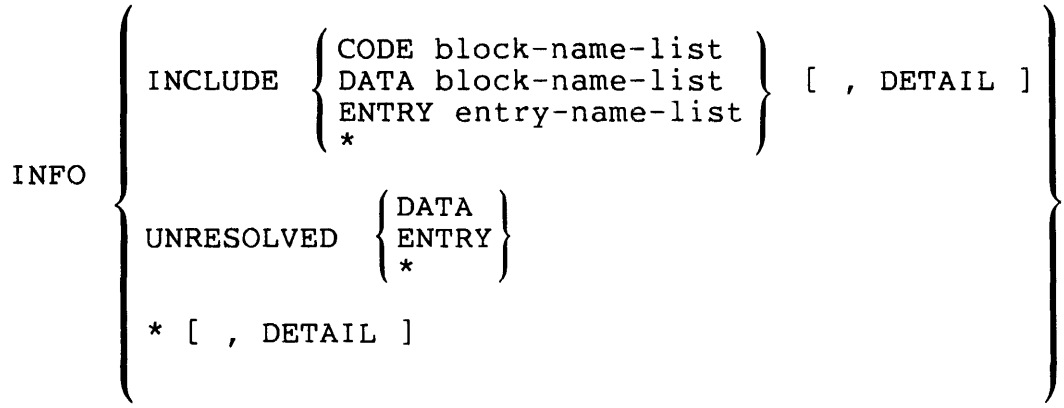
displays the names of entry points and data blocks in the unresolved reference lists.

*

displays all include lists and unresolved lists.

INFO Command

The INFO command displays information about code blocks, entry points, and data blocks in the include lists and the unresolved reference lists.



INCLUDE CODE block-name-list

displays the attributes and lengths of code blocks in the block-name-list. Valid forms of code or data block-name-list are given in Table 3-2.

INCLUDE DATA block-name-list

displays the lengths of all data blocks in block-name-list. Valid forms of block-name-list are given in Table 3-2.

INCLUDE ENTRY entry-name-list

displays the attributes of entry points in entry-name-list. Valid forms of entry-name-list are given in Table 3-2.

INCLUDE *

displays the attributes of all code blocks, data blocks, and entry points in the include lists.

→

DETAIL

provides further information for blocks in the include lists such as date and time compiled, source language, and the source file from which the block was compiled.

UNRESOLVED DATA

displays names of data blocks on the unresolved reference list.

UNRESOLVED ENTRY

displays the names and default files of entry points in the unresolved reference list.

UNRESOLVED *

displays the names of entry points and data blocks in the unresolved reference lists.

*

displays all include lists and unresolved lists.

Examples of the INFO command:

1.
```
@INFO *
INCLUDE CODE: 27 ENTRIES
 NAME                                SIZE    ATTRIB
 PRICE                                289
   .   .   .   .   .   .   .   .   .   .   .

INCLUDE ENTRY: 29 ENTRIES
 NAME                                SIZE    ATTRIB
 PRICE                                 43    M
   .   .   .   .   .   .   .   .   .   .   .

INCLUDE DATA:  5 ENTRIES
 NAME                                SIZE
 #RUCB                                77
   .   .   .   .   .   .   .   .   .   .   .

UNRESOLVED ENTRY: 52 ENTRIES
 NAME                              FILE
 INITIALIZER
   .   .   .   .   .   .   .   .   .   .   .
```

2.
```
@INFO *, DETAIL
INCLUDE CODE: 15 ENTRIES
 NAME                                 SIZE    ATTRIB
 FORMATTER                            4672
  LANG: TAL TIME: 10/01/82 14:48  SYMBOLS: OFF  FILE: $VL.S1.FMT
   .   .   .   .   .   .   .   .   .   .   .

INCLUDE ENTRY: 20 ENTRIES
 NAME                                OFFSET   ATTRIB
 FORMATTER                            4450    V
  LANG: TAL TIME: 10/01/82 14:48  SYMBOLS: OFF  FILE: $VL.S1.FMT
   .   .   .   .   .   .   .   .   .   .   .

INCLUDE DATA:  5 ENTRIES
 NAME                                SIZE
 #RUCB                                77
 LANG: FORTRAN TIME: 10/03/82 19:55 SYMBOLS: OFF FILE: $VL.S1.FN
   .   .   .   .   .   .   .   .   .   .   .
```

LIST Command

The LIST command provides load maps and cross-reference data for entry points and code and data blocks. You must specify an object file with either a previous FILE command or with the FROM parameter of the LIST command.

```
                  ⎧ SOURCE                                    ⎫
                  ⎪                                           ⎪
                  ⎪ CODE ⎧address-list    ⎫                  ⎪
                  ⎪      ⎩block-name-list ⎭                  ⎪
                  ⎪                                           ⎪
                  ⎪ DATA ⎧address-list    ⎫                  ⎪
LIST              ⎪      ⎩block-name-list ⎭                  ⎪   [ FROM file-name ]
                  ⎪                                           ⎪
                  ⎪ list-option [ , BRIEF ]                   ⎪
                  ⎪                                           ⎪
                  ⎪ ( list-option [ , list-option ] ...       ⎪
                  ⎪                        [ , BRIEF ] )      ⎪
                  ⎪                                           ⎪
                  ⎪ *  [ , BRIEF ]                            ⎪
                  ⎩                                           ⎭
```

SOURCE

specifies that the source file be listed for each code and data block used in creating the object file. The output shows, by source file, all code blocks (identified by the letter P) and all data blocks (identified by the letter B) in the object file.

⎧CODE⎫ address-list
⎩DATA⎭

for each address in address-list, lists the block name of the code at the given address in code or data space. The addresses in address-list are word addresses taken from the load map. The 132-character output gives all the information shown in a load map from BINDER.

→

{CODE / DATA} block-name-list

for each block name in block-name-list, lists the base address in code or data space. Valid forms of block-name-list are given in Table 3-2. The 132-character output gives all the information shown in a load map from BINDER.

list-option

specifies the type of map to be displayed:

ALPHA
LOC
XREF
*

ALPHA

displays a load map in alphabetic order. The map lists names and addresses for code blocks and data blocks. The map also gives the language and name of the source file which yielded each block, and the date and time of compilation.

LOC

displays a load map in location order. The map lists names and addresses for code blocks and data blocks. The map also gives source information as for ALPHA.

XREF

displays an entry point and data block cross-reference list.

*

generates all three listings.

BRIEF

requests display of an 80-character load map line rather than the standard 132-character load map line. The truncated line omits the DATE, TIME, LANGUAGE, and SOURCE FILE information for code and data blocks.

→

FROM file-name

specifies the file name of an object file to be mapped. The default is the current file.

The listings are described in more detail in Section 4.

Examples of the LIST command:

1. @COMMENT - The following command outputs to listfile an entry
   @COMMENT - point and data block cross-reference list for the
   @COMMENT - current file (assumed to be already established).
   @LIST / OUT listfile / XREF

2. @COMMENT - The following command outputs to the home terminal
   @COMMENT - all three load maps (ALPHA, LOC, XREF) for oldfile,
   @COMMENT - using the 80-character truncated line.
   @LIST * FROM oldfile, BRIEF

3. @COMMENT - The following command outputs to the home terminal
   @COMMENT - the ALPHA and LOC load maps for objfile, using the
   @COMMENT - standard 132-character line.
   @LIST (ALPHA, LOC) FROM objfile

MODIFY Command

The MODIFY command changes the values of words in the code and data blocks of the target file. No changes are made to existing files.

```
MODIFY  {CODE code-block-name}  [ modify-spec ] [ offset ]
        {DATA data-block-name}

  [ , value ] ...
```

CODE code-block-name

is the name of a code block in the include code block list to be modified. Use the INFO INCLUDE CODE * command to display the include code block list.

DATA data-block-name

is the name of a data block in the include data block list to be modified. Use the INFO INCLUDE DATA * command to display the include data block list.

modify-spec

specifies the input format for value and the output format for the current values. The default is OCTAL. modify-spec is one of:

ASCII
HEX
DECIMAL
OCTAL

offset

is the offset in octal from the base of the block of the word to receive value. The default is the base of the block (offset 0).

→

value

is an expression for the replacement of the word contents. BINDER prompts (interactively) for the new value if you omit it and assumes value is in the same format as dump-spec; an ASCII value must be enclosed in quotation marks. If more than one value occurs, BINDER modifies word locations sequentially.

PROMPTING SEQUENCE. If you don't specify value, BINDER prompts for input in this sequence. BINDER displays the address in octal of the location and the current value at that location. The location is offset or the base of block. The display is as shown below where nnnnn is the offset.

```
{CODE} BLOCKNAME+nnnnn (oldval)<--
{DATA}
```

You can select whether the value display is octal, decimal, hex, or ASCII. BINDER accepts your replacement value in the same form as the displayed value. Do not specify a prefix for hex or octal values; BINDER does not accept numbers in the form %nnn.

Prompting continues until a carriage return indicates no further modifications or until the end of the block occurs.

MODIFYING EXTERNAL REFERENCES. BINDER issues a warning message if you modify a CALL or a reference to global data. If the modified target file is included in a subsequent binding operation, BINDER reissues the warning message.

CONSIDERATIONS. To verify changes, use the SHOW MODIFY command. The changes are made only to the target file; the input object file is unchanged.

Examples of the MODIFY command:

1.
```
@COMMENT - this example causes the prompting sequence
@COMMENT - to begin with the base address
@COMMENT - (The resulting display follows the command.)
@MODIFY CODE block-3
CODE BLOCK-3+00000 (012345) <-- 000000
CODE BLOCK-3+00001 (000000) <-- cr ends the MODIFY prompt
```

2.
```
@MODIFY CODE lm^init
CODE LM^INIT+00000 (070402) <-- 70401
CODE LM^INIT+00001 (024700) <-- 24701
CODE LM^INIT+00002 (002005) <-- 2006
CODE LM^INIT+00003 (040001) <-- 40000
CODE LM^INIT+00004 (014404) <-- 14403
CODE LM^INIT+00005 (100777) <-- cr
@COMMENT  -   verify changes using SHOW
@SHOW MODIFY
@COMMENT - BINDER replies with all entries on MODIFY list
MODIFY         5 ENTRIES:
MODIFY          CODE LM^INIT+00000 = 070401      LADR  L+001
MODIFY          CODE LM^INIT+00001 = 024701      PUSH    701
MODIFY          CODE LM^INIT+00002 = 002006      ADDS   +006
MODIFY          CODE LM^INIT+00003 = 040000      LOAD  G+000
MODIFY          CODE LM^INIT+00004 = 014403      BAZ    +003
```

MOVE Command

The MOVE command allows you to relocate code blocks within the target file. For example, if BINDER was unable to fill the gap preceding the 32K boundary because of the order of the include lists or because of code block sizes, you can establish a more efficient order.

You can also use this command to reduce page faults; however, this requires careful analysis. Refer to the XRAY Users Manual for information on obtaining the analytical data needed to analyze program behavior.

```
MOVE entry-name-list { AFTER | BEFORE } entry-name

  [ , entry-name-list { AFTER | BEFORE } entry-name ] ...
```

entry-name-list

is a list of one or more entry names in the include entry name list; either a primary or secondary name is permitted. When a secondary name is specified, BINDER moves the containing block for the secondary name. See Table 3-2 for the valid list formats.

AFTER entry-name

specifies the position in the include entry name list after which entry-name-list should appear; entry-name cannot be within the range of entry-name-list. Primary and secondary entry point names are accepted.

BEFORE entry-name

specifies the position in the include entry name list before which entry-name-list should appear; entry-name cannot be within the range of entry-name-list. Primary and secondary entry point names are accepted.

Examples of the MOVE command:

1.
```
@ADD * FROM objfile
@MOVE block-1 AFTER block-5
@BUILD
@COMMENT  -  the entry name block-1 is moved from whatever
@COMMENT  -  position it currently occupies in the include
@COMMENT  -  list to the position immediately following
@COMMENT  -  block-5.
```

2.
```
@MOVE (block-1, block-7) BEFORE block-2
@COMMENT  -  the entry names block-1 and block-7 are moved from
@COMMENT  -  whatever positions they currently occupy in the
@COMMENT  -  include list to the positions immediately preceding
@COMMENT  -  block-2.
```

RENAME Command

The RENAME command enables the user to rename a code or data block.

```
RENAME  {CODE code-block-name}  TO  name
        {DATA data-block-name}
```

CODE code-block-name

specifies the current name of the code block to be renamed.

DATA data-block-name

specifies the current name of a data block to be renamed.

name

specifies the new name to be assigned to the specified block.

The RENAME command is useful when you must bind together two object files that contain data blocks with the same name but different types. RENAME cannot be used when special data blocks are involved.

Examples of the RENAME command:

1. @RENAME CODE block1 TO blocka

2. @RENAME DATA datax TO datay

REPLACE Command

The REPLACE command inserts replacements for named code or data blocks on the include lists. If the names to be replaced are not already on the include lists, they are ignored.

```
REPLACE { CODE entry-name-list  }  [ FROM file-name ]
        { DATA data-block-list  }
        { *                     }
```

CODE entry-name-list

specifies the entry points in the file that are to replace entry points in the include entry name list. Refer to Table 3-2 for the valid forms of entry-name-list.

DATA block-name-list

specifies the data blocks in the file that are to replace data blocks in the include data block list. Refer to Table 3-2 for the valid forms of block-name-list.

*

specifies that all entry points and data blocks in the file are to replace matching entry points and data blocks in the appropriate include lists.

FROM file-name

identifies the file containing the entry points and data blocks to replace corresponding ones in the include lists. The default is the current file.

NOTE

It is useful to remember that the ADD command (with the DELETE option specified) performs functions similar to those performed by REPLACE. (ADD,DELETE adds a specified entry name to the end of the include list, deleting the previous occurrence of the entry; REPLACE removes the previous occurrence of the entry name and inserts the new reference in its place.) If an error occurs when you attempt a REPLACE command, you may be able to accomplish your goal by using ADD,DELETE. For example, the REPLACE command replaces both the direct and the indirect data blocks in a TAL object file. If a TAL data block that has an associated indirect data block is replaced with a TAL data block or FORTRAN common block that does not have an indirect data block, BIND ends with a fatal error. The ADD command with the DELETE option can be used successfully in this case.

Example of the REPLACE command:

1.
```
@ADD * FROM oldfile
@REPLACE CODE block-1 FROM objfile
@COMMENT  the entry name block-1 now refers to code in objfile
@COMMENT  rather than to the code with that entry name in
@COMMENT  oldfile.
```

2.
```
@COMMENT  objfile was compiled with a ?SEARCH directive
@COMMENT  for library libfile; libfile was recompiled
@COMMENT  and objfile is to be rebuilt using new libfile
@ADD * FROM objfile
@REPLACE * FROM libfile
@BUILD nobjfile
```

RESELECT Command

The RESELECT command returns one or more SELECT command parameter names to the default value. SELECT parameters specify BINDER's operation during execution of BUILD and SATISFY commands.

```
RESELECT  {select-parameter  [ , select-parameter ] ...}
          {*                                            }
```

select-parameter

is a valid parameter name for the SELECT command to return to the default value. select-parameter is one of:

CHECK
COMPACT
LIST
OMIT
REFER
SATISFY
SEARCH

*

specifies that all SELECT command parameters are to be reset to the default values.

The SELECT command parameter defaults or initial states are:

CHECK	BLOCK ON, LIBRARY OFF, PARAMETER ON
COMPACT	ON
LIST	ALPHA ON, LOC OFF, XREF OFF
OMIT	empty list
REFER	empty list
SATISFY	ON
SEARCH	empty list

Examples of the RESELECT command:

1. @RESELECT *

2. @COMMENT - clear omit and refer lists; use listing defaults
   @RESELECT OMIT, REFER, LIST

RESET Command

The RESET command restores the default value to one or more target file attributes that were previously specified with the SET command.

```
RESET  { set-parameter  [ , set-parameter ] ... }
       { *                                      }
```

set-parameter

is a valid SET command parameter name to be reset to the default value. set-parameter is one of:

DATA
EXTENDSTACK
INSPECT
LIBRARY
LIKE
PEP
SAVEABEND
STACK
SYMBOLS

*

specifies that all SET attributes are to be reset to the default values.

The SET command parameter default values are:

DATA	PAGES for all data blocks and stack estimate
EXTENDSTACK	PAGES estimated
INSPECT	OFF
LIBRARY	no user library
LIKE	(no default)
PEP	minimum for entry points in target file
SAVEABEND	OFF
STACK	PAGES estimated
SYMBOLS	ON

Resetting LIKE causes the four parameters DATA, INSPECT, LIBRARY, and SAVEABEND to be reset to the default values.

Examples of the RESET command:

1. @RESET LIBRARY, SAVEABEND
2. @RESET *

SATISFY Command

The SATISFY command attempts immediate resolution of external references to entry points and data blocks in the unresolved reference list. BINDER uses the current include lists and the search list.

```
SATISFY [ select-specification [ , select-specification ] ... ]

select-specification

  specifies a SELECT command parameter and value to be used for
  this statement only. This specification overrides any value
  previously established for that parameter.
```

During execution of a SATISFY command, BINDER searches the object files listed in the search list in an attempt to resolve any unresolved external references listed in the unresolved reference list. Object file names are added to the search list with the SELECT command, but, if the SEARCH file-name parameter is used with the SATISFY command, the files specified will be searched instead, overriding any previously established search list.

In the following example, files objectc and objectd will be searched for external reference resolution and not files objecta and objectb, which have already been added to the search list with the SELECT command.

```
@SELECT SEARCH (objecta, objectb)
@SATISFY SEARCH (objectc, objectd)
```

If BINDER is unable to resolve a reference, the reference remains on the unresolved reference list until resolved by another SATISFY command or until execution of the BUILD command.

Note that any select specification used in a SATISFY command temporarily sets a new value to be used only during execution of that same SATISFY command.

In interactive mode, BINDER prompts for additional target file definition commands following execution of a SATISFY command. Target file generation does not begin until you enter a BUILD command.

Examples of the SATISFY command:

Example 1 commands cause all references to epnamel in oldfile to refer instead to epname2 from newfile.

1.
```
@COMMENT - "caller" inserted on next command calls epnamel
@COMMENT - the satisfy will result in calling epname2
@ADD CODE caller FROM objfile
@SATISFY SEARCH newfile, REFER epnamel TO epname2
@ADD CODE epnamel FROM oldfile
@BUILD newfile !
```

2.
```
@COMMENT - this example leaves entry references unresolved
@COMMENT - since SATISFY OFF (a select-specification) is valid
@ADD * FROM objfile
@SATISFY SATISFY OFF
```

SELECT Command

The SELECT command sets parameter values that control BINDER during execution of subsequent BUILD and SATISFY commands. The values can be overridden by the BUILD and SATISFY commands or be reset to default values by the RESELECT command.

```
                  CHECK   {check-option                                  }
                          {( check-option [ , check-option ] ... )       }

                  COMPACT { ON | OFF }

                  LIST    {list-option                                   }
                          {( list-option [ , list-option ] ... )         }
SELECT            OMIT    {entry-name                                    }   , ...
                          {( entry-name [ , entry-name ] ... )           }
                  REFER   {refer-pair                                    }
                          {( refer-pair [ , refer-pair ] ... )           }

                  SATISFY { ON | OFF }

                  SEARCH  {file-name                                     }
                          {( file-name [ , file-name ] ... )             }
```

CHECK check-option

selects a type of error checking as follows:

```
{BLOCK    }
{LIBRARY  } { ON | OFF }
{PARAMETER}
{*        }
```

BLOCK { ON | OFF }

specifies whether common blocks are checked for consistency in length and addressing (that is, all byte or all word). CHECK BLOCK OFF can be used to check FORTRAN procedures for adherence to the FORTRAN rules for common blocks. CHECK BLOCK ON checks TAL blocks (if a global data block used in several object files changes but only some of the object files have the new one, BINDER detects the length mismatch and issues a warning). The default is ON.

→

LIBRARY { ON | OFF }

specifies whether NonStop II user library checking is in effect when BINDER is building the target file. Any reference to a data block other than a read-only block or any entry point that has the main attribute causes a warning message to be issued. The default is OFF. (Note that BIND does not perform checking on existing libraries, only on libraries that are being built.)

PARAMETER { ON | OFF }

specifies whether parameter lists between code blocks are checked when the target file is built. Each parameter is checked for proper size, type, and mode (value or reference); the return type of functions is also checked. The default is ON.

* { ON | OFF }

specifies whether all check options are to take effect.

COMPACT { ON | OFF }

specifies whether the gap at the 32K boundary is to be filled when the target file is built. A gap occurs when a code block that would be allocated across the 32K boundary is relocated to begin at the 32K boundary. The default is ON.

LIST list-option

specifies the printed output for building the target file:

{ ALPHA | LOC | XREF | * } { ON | OFF }

→

ALPHA { ON | OFF }

specifies whether a load map in alphabetic order is produced. The map lists names and addresses for code blocks and data blocks; source file information for each code block or data block is also listed. The default is ON.

LOC { ON | OFF }

specifies whether a load map in location order is produced. The map lists names and addresses for code blocks and data blocks; source file information for each code block or data block is also listed. The default is OFF.

XREF { ON | OFF }

specifies whether an entry point and data block cross-reference list is produced. The default is OFF.

* { ON | OFF }

specifies whether all list options are selected.

OMIT entry-name

specifies an entry point name to be added to the omit list. The default is an empty omit list. Note that all entry points belonging to a particular code block are omitted if any one of them appears on the omit list.

REFER refer-pair

specifies a pair of entry point names to be added to the refer list. The format for refer-pair is:

entry-name TO entry-name

The default is an empty refer list.

→

SATISFY { ON | OFF }

specifies whether to attempt resolution of remaining unresolved external references for entry points when a BUILD or SATISFY command is issued. Data block references are not affected by SATISFY. OFF suppresses automatic resolution; the default is ON.

SEARCH file-name

is a file name to be added to the search list. The default is an empty search list.

Examples of the SELECT command:

1. @COMMENT - allow gap before 32K; suppress epname resolution
   @SELECT COMPACT OFF, SATISFY OFF

2. @COMMENT - add three file names to the search list; put all
   @COMMENT - check options in effect.
   @SELECT SEARCH (objfile1, objfile2,objfile3), CHECK * ON

3. @COMMENT - add two pairs of entry names to refer list; add
   @COMMENT - one entry name to omit list.
   @SELECT REFER (blk1 TO blk4, blk2 TO blk3), OMIT blk5

SET Command

The SET command specifies attribute values to be associated with the target file. Attribute values can be overridden by the BUILD command or reset to default values by the RESET command.

```
      ⎧ ⎧DATA       ⎫                                          ⎫
      ⎪ ⎨EXTENDSTACK⎬  value [ PAGES | WORDS ]                  ⎪
      ⎪ ⎩STACK      ⎭                                          ⎪
      ⎪ INSPECT { ON | OFF }                                   ⎪
SET   ⎨ LIBRARY file-name                                      ⎬  , ...
      ⎪ LIKE file-name                                         ⎪
      ⎪ PEP value                                              ⎪
      ⎪ SAVEABEND { ON | OFF }                                 ⎪
      ⎩ SYMBOLS { ON | OFF }                                   ⎭
```

DATA value [PAGES | WORDS]

specifies the amount of data space to be allocated. The default data space allocated is the maximum number of data pages in any of the files from which data is included, or the amount needed to hold all the data blocks plus an estimate of the stack space needed for local storage, whichever is larger. Either a decimal value or an octal value (preceded by %) is accepted. The default unit for the value is PAGES; one PAGE is 1024 WORDS.

EXTENDSTACK value [PAGES | WORDS]

specifies the number of pages or words to add to BINDER's estimate. The allocation is for the total; the default is the estimate. Either a decimal value or an octal value (preceded by %) is accepted. (One PAGE is 1024 WORDS.) The default unit for the value is PAGES.

STACK value [PAGES | WORDS]

overrides the estimate for stack space computed by BINDER. The number of pages or words plus the space required for global data blocks is the total number of data pages. (One PAGE is 1024 WORDS.) The default is the space estimated for local storage. Either a decimal value or an octal value (preceded by %) is accepted. The default unit for the value is PAGES.

→

INSPECT { ON | OFF }

specifies whether the INSPECT program is chosen for debugging when the target file is executed. The default is OFF; that is, the DEBUG program is used. INSPECT OFF automatically causes BINDER to set SAVEABEND OFF. (The COMINT SET INSPECT and RUN commands both allow overriding the INSPECT option.)

LIBRARY file-name

specifies the name of the NonStop II user library to be associated with the object file at run time. This file name can be overridden at run time by using the LIB parameter in the COMINT RUN command. The default is no user library.

LIKE file-name

specifies that the DATA, INSPECT, LIBRARY, and SAVEABEND attributes for the target object file are to be set identical to the attributes of a specified object file.

PEP value

specifies the size of the Procedure Entry Point (PEP) table to be allocated for the target file. The default value is the minimum size necessary for the number of entry points in the target file. Any integer value from 1 through 512 is accepted. It may be specified either in decimal or in octal (preceded by %).

SAVEABEND { ON | OFF }

specifies whether a save file is to be created if the process terminates abnormally during execution. BINDER automatically sets INSPECT ON if SAVEABEND ON is on. The default is OFF.

SYMBOLS { ON | OFF }

specifies whether symbol tables in the object files should be retained in the target file. The default is ON.

Only one of the DATA, STACK, and EXTENDSTACK parameters can be set at at a time. Each successive SET command specifying one of these parameters overrides the previous specification.

COBOL extended data block sizes are set at compile time, and BINDER collects all of these blocks into one extended segment, which is automatically allocated at run time by the run-time loader. SET DATA does not affect the size of this extended segment.

Examples of the SET command:

1. @SET SAVEABEND ON

2. @SET SYMBOLS OFF, LIBRARY libfile

SHOW Command

The SHOW command displays the current values for the parameters of the SELECT and SET commands, the current file, and the set of modifications established by the MODIFY command.

After a BUILD is executed, BINDER resets SELECT, SET, FILE, and all lists to the default states. SHOW cannot display information about the constructed target file.

```
        ⎧ FILE             ⎫
        ⎪ MODIFY           ⎪
SHOW    ⎨ SELECT           ⎬
        ⎪ select-parameter ⎪
        ⎪ SET              ⎪
        ⎩ set-parameter    ⎭
```

FILE

requests the name of the current file to be displayed.

MODIFY

displays the current set of modifications established by the MODIFY command. If code has been modified, the values are displayed in ICODE as well as octal.

SELECT

displays the current values for all the SELECT command parameters.

select-parameter

specifies a SELECT command parameter to be displayed:

CHECK
COMPACT
LIST
OMIT
REFER
SATISFY
SEARCH

→

SET

displays the current values for all the SET command parameters.

set-parameter

specifies a SET command parameter to be displayed:

DATA
EXTENDSTACK
INSPECT
LIBRARY
LIKE
PEP
SAVEABEND
STACK
SYMBOLS

Examples of the SHOW command:

1.
```
@COMMENT - the following command results in no display
@COMMENT - because there are no entries on the OMIT list.
@SHOW OMIT
```

2.
```
@SHOW FILE
FILE          \YOUR.$YVOL.YSUBVOL.YFNAME
```

3.
```
@COMMENT - show display at start of BIND process
@SHOW SET
DATA          (0 PAGES)
EXTENDSTACK
INSPECT       OFF
LIBRARY
LIKE
PEP
SAVEABEND     OFF
STACK
SYMBOLS       ON
```

4. @COMMENT - show display at start of BIND process
```
@SHOW SELECT
CHECK        BLOCK ON, LIBRARY OFF, PARAMETER ON
COMPACT      ON
LIST         ALPHA ON, LOC OFF, XREF OFF
OMIT         0 ENTRIES
REFER        0 ENTRIES
SEARCH       0 ENTRIES
```

STRIP Command

The STRIP command removes the BINDER and INSPECT tables from the named object file. Note that STRIP modifies the named file; the file is not copied to a target file. If BINDER and INSPECT tables are required for future analysis, copy the files to another location (for example, magnetic tape).

To delete only the INSPECT symbol tables, use the SET SYMBOLS OFF command and then issue the BUILD command. (This does not change the input file on disc; BINDER copies the file to the target file.)

```
STRIP file-name

file-name

  is the disc file name of an object file whose BINDER region is
  to be deleted; INSPECT symbol tables are also deleted if they
  exist.
```

After the strip, an attempt to use file-name as input to any BINDER operation results in an error message. BINDER regards the file as if it were built by a pre-BINDER version of a compiler or by UPDATE. The STRIP command cannot be used on files containing COBOL Extended Data blocks.

VERIFY Command

The VERIFY command compares the actual value of a code or data word in an object file with a value specified by the user. If the value in the object file is not identical to that specified, the BIND process terminates.

```
VERIFY { CODE code-block-name } [ verify-spec ]  [ offset ] , value
       { DATA data-block-name }
```

CODE code-block-name

is the name of a code block containing a value to be verified.

DATA data-block-name

is the name of a data block containing a value to be verified.

verify-spec

specifies the format for value. The default is OCTAL. verify-spec is one of:

ASCII
HEX
DECIMAL
OCTAL

offset

is the offset in octal from the base of the block used to compute the exact location of the word whose value is to be verified against value. The default is the base of the block (offset 0).

value

is an expression to be verified against the contents of the specified word. An ASCII value must be enclosed in quotation marks.

The VERIFY command can be used during a noninteractive BIND session to verify that the actual value of a code or data word corresponds to the value you expect and to terminate the BIND session when discrepancies are detected. If the VERIFY command yields a discrepant value, BINDER issues the fatal error message "Value specified in VERIFY command not equal to current value" and terminates.

Examples of the VERIFY command:

1. @VERIFY CODE block-3, 17

2. @VERIFY DATA block ASCII 50, "ab"

SECTION 4

BINDER OPERATION

This section discusses the code and data blocks that make up object files, object file format, and the stages of BINDER operation. Lastly, this section also includes requirements for binding blocks that are written in different languages.

Briefly, BINDER operates in two stages to create target files:

1. command input - defines the names of disc files to search for the target file contents, names of code and data blocks to extract from the searched object files, the order in which to insert blocks in the target file, and changes to code and data and reference resolution.

2. target file build - BINDER uses the collected information to create a new object file on disc.

OBJECT FILE STRUCTURE

Remember that target files are object files, just as the input files BINDER uses as reference code are object files. Since the following discussion pertains to object file structure (both for input code files and target files), no distinction is made.

All BINDER operations are performed on object files. For clarity, the following definitions are assumed:

- block - the smallest separately relocatable unit of code or data. Data can be separately compiled in FORTRAN if compiled as COMMON and in TAL if compiled as BLOCK structures, respectively.

- object file - one or more code and data blocks compiled and bound together.

- program - an executable object file. It must contain an entry point with the MAIN attribute (in COBOL, this is a "calling program").

Table 4-1 gives the corresponding COBOL, FORTRAN, and TAL terminology for code and data blocks.

Table 4-1. Source Language Names for Blocks

BINDER BLOCK TYPE	COBOL Equivalent	FORTRAN Equivalent	TAL Equivalent
CODE	Program Unit	Program Unit PROGRAM SUBROUTINE FUNCTION	Procedure PROC
DATA			
Own	Working-Storage Extended-Storage	SAVE and DATA	N/A
Common	N/A	COMMON (named & blank)	BLOCK (named & PRIVATE); global read-only arrays;implicitly- named global data
Special	compiler- generated only	compiler- generated only	N/A

Compilers provide names for all code and data blocks that are unnamed in the source code. "Block Naming Conventions" in this section describes the names for different block types.

BINDER receives from compilers and interactive users the names of disc files to search for specific code and data blocks. When the BINDER finds each block, it copies the code or data block into the target file. The input object file is not affected; it remains on disc in its original state.

Code Blocks

As shown in Table 4-1, a code block can be a COBOL program unit, a FORTRAN program or subprogram, or a TAL procedure. (TAL SUBPROCs cannot be separately compiled.)

ENTRY POINT NAMES. Code blocks can refer to other code blocks at named entry points. These are external references that the BINDER resolves during binding if the addresses are known.

TAL and FORTRAN code blocks can have multiple entry points. If multiple entry points occur, the primary entry point name is also the name of the code block; secondary entry point names are declared by using ENTRY statements.

COBOL program units can have only one entry point.

CODE BLOCK ATTRIBUTES. Especially in TAL, code block declarations can contain attributes that define execution or relocation characteristics. Some attributes can be altered using the BINDER ALTER command. The TAL INTERRUPT, EXTENSIBLE, and VARIABLE attributes, once set at compile time, cannot be changed.

The MAIN attribute is used for COBOL, FORTRAN, and TAL compiled code. After compilation, however, you can alter MAIN for TAL procedures only.

The following table shows code block attributes recognized by BINDER. "yes" indicates that you can explicitly set the attribute in the given language compiler.

Table 4-2. Code Block Attributes

Attribute	COBOL	FORTRAN	TAL	Alterable in a BINDER Session
MAIN	yes	yes	yes	TAL only
CALLABLE			yes	yes
INTERRUPT			yes	
PRIVILEGED			yes	yes
RESIDENT			yes	yes
VARIABLE			yes	

Refer to the TAL Reference Manual discussion of procedure and subprocedure declarations for descriptions of the code attributes.

Data Blocks

BINDER recognizes three types of data blocks: own, common, and special blocks. Unlike code blocks, which always contain code for exactly one program unit, data blocks are simply collections of logically-related data. More than one code block can refer to the same data block through external references.

EXTERNAL DATA REFERENCES. Unlike external references to code blocks, data block references must be resolved during binding. Therefore, you cannot create an object file until all the necessary data blocks are compiled.

You can modify separately compilable data blocks in an object file either by recompiling the block or by using BINDER's MODIFY command. Then, you can build a new object file with the corrected data block during an interactive BINDER session. It is not necessary to recompile the entire program.

TYPES OF DATA BLOCKS. Source code statements define own and common blocks. Compilers generate special blocks that are referred to only by compiler-generated statements. The contents of block types are:

Own blocks consist of data that is accessible only to a single code block.

COBOL	-	Working-Storage and Extended-Storage sections
FORTRAN	-	SAVE or DATA that is not COMMON
TAL	-	none

Common blocks contain data that can be referred to from multiple code blocks.

COBOL	-	no common data; LINKAGE SECTION is dynamically allocated for local data
FORTRAN	-	blank or named COMMON; BINDER treats them the same
TAL	-	BLOCK (named and PRIVATE), global read-only arrays, and implicitly named globals

Special blocks are compiler-generated blocks. Program control blocks such as the run-unit control block (RUCB) and program-unit control block (PUCB) are special blocks.

COBOL	-	program control blocks
FORTRAN	-	program control blocks
TAL	-	none

Block Naming Conventions

Code and data blocks must be specified by name with BINDER commands. The names are listed on output listings from the compilers and from BINDER.

Block names are established as follows:

- Code block names are the source code names for the compilable units: PROGRAM, SUBROUTINE, FUNCTION, PROC, NAME. Unnamed FORTRAN programs are named MAIN^ by the compiler.

- Common data block names are the source code names if specified in a statement. FORTRAN blank COMMON blocks are named BLANK^ by the compiler.

- Own data blocks can have the same name as the corresponding code block in the compilable unit. Extended data blocks,which are also own blocks, have "^" appended to the code block name.

- Special block names are assigned by the COBOL and FORTRAN compilers and are distinguished by having a pound sign (#) for at least one of the characters in the name. Special block names are present in the data block output listings. These names are usually not used in BINDER commands.

  Examples of special block names are: #HIGHBUF, #G0, #PUCB, #RUCB, and COMMON#POINTERS. A COBOL PUCB name has the form PROG-NAME#, where PROG-NAME is the name of the code block.

OBJECT FILE FORMAT

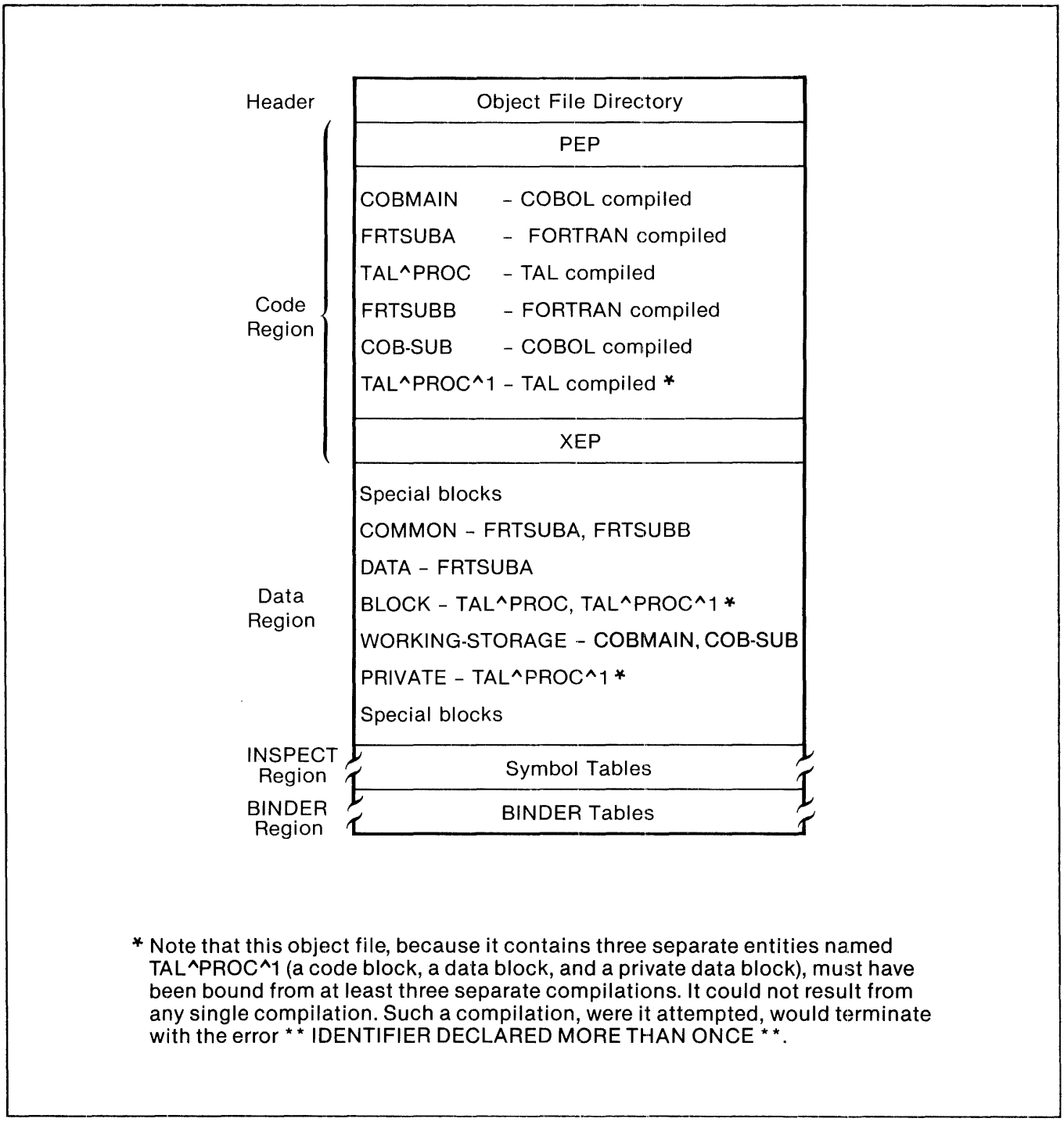
All object files have the same format, regardless of the number of code and data blocks.

Figure 4-1 shows an example object file made up of several blocks copied from different object files. Since both FORTRAN and COBOL compiled code is included, the MAIN program unit must be COBOL. "Mixed Language Binding" in this section describes the restrictions on using COBOL and FORTRAN in the same program.

On disc, an object file can consist of a maximum of 16 extents. The header, code, and data areas must be in the first extent (0). The additional extents (1 through 15) can be used for INSPECT symbol tables and BINDER tables.

Header

The object file header is a block at offset zero containing pointers and descriptive information for other blocks in the object file.



* Note that this object file, because it contains three separate entities named TAL^PROC^1 (a code block, a data block, and a private data block), must have been bound from at least three separate compilations. It could not result from any single compilation. Such a compilation, were it attempted, would terminate with the error ** IDENTIFIER DECLARED MORE THAN ONCE **.

Figure 4-1. Example of BINDER's Object File Format

Code Region

The code region consists of consecutive pages of disc space, starting on a page boundary, in the object file. BINDER output statistics give the exact number of pages to be allocated for the code area at run time. Code region contents, in order, are:

1. PEP table
2. global read-only arrays (TAL only)
3. resident code blocks (TAL only)
4. nonresident code blocks
5. XEP table.

USER CODE. You can define the order of blocks that BINDER should use to build the code region. In building the file, BINDER separates resident code blocks from nonresident code blocks for you.

In the completed target file, nonresident code can come before resident code if compression of the target file requires it. If compressing the file contents, BINDER will move a nonresident block that fits in the gap at 32K even if resident code blocks are placed above the 32K boundary. The SELECT COMPACT OFF command prevents BINDER from performing any compression or filling the gap.

TAL - global read-only arrays are placed in the code region immediately following the PEP. Read-only string arrays cannot straddle the 32K boundary. BINDER moves such an array to above the boundary to avoid straddling the boundary. If this happens, a warning message is issued.

PEP AND XEP. The PEP and XEP tables are GUARDIAN operating system tables whose contents are briefly described here.

PEP - Procedure Entry Point table, contains the entry point addresses for each code block in the code area. The PEP is in the first page of the code area.

XEP - External Entry Point table, is in the last page of the code area and contains an entry for each unresolved external reference. The GUARDIAN software fills in this table at run time. If the object file is a program file, the external references still unresolved are for calls resolved at NEWPROCESS time. These can refer to entry points in the system library or in the FORTRAN or COBOL libraries. On NonStop II systems, they can also refer to entry points in user library segments.

Data Region

BINDER determines the minimum number of pages to be allocated for the data area and reports the number in the output statistics. The data area starts on a page boundary.

BINDER also determines the order for data blocks so you need not specify their order when defining a target file.

INSPECT Region

Symbol tables in the INSPECT region contain information on all symbols in blocks compiled with the ?SYMBOLS directive. An object file can contain symbol tables for some of the blocks and not for others. That is, ?SYMBOLS can be turned on and off on a procedure basis.

During an interactive BINDER session, you can specify whether to retain the symbol tables in the target file. You should retain symbol tables for blocks still in the development cycle if using INSPECT's high-level commands.

Once symbol tables are deleted, you must recompile if they are needed again. Low-level INSPECT commands and DEBUG commands can be used whether symbol tables exist or not. Refer to the INSPECT Users Manual and the DEBUG Reference Manual for debugging information.

The space required by symbol tables depends on program characteristics. Space requirements for the object file can almost double when data requirements are complex. The STRIP command deletes both BINDER tables and INSPECT symbol tables. Once STRIP is used on a file, BINDER cannot manipulate the file.

BINDER Region

The BINDER region contains a header and BINDER's tables. These are:

- procedure information table
- entry point table
- data block information table.

INPUT STAGE

During the input stage, BINDER accepts commands and collects the information in lists that are used during the output stage. These lists are:

- three include lists
- one omit list
- one refer list
- one modify list
- two unresolved reference lists
- one search list.

When BINDER is started, all lists are empty. After a build is completed, the lists are again empty. Another target file can be defined or the session can be ended.

The unresolved reference lists are automatically maintained by BINDER. Table 4-3 lists the commands to create the other control lists.

Table 4-3. Commands to Create Control Lists

ADD	Inserts names on the include lists.
REPLACE	Changes code blocks and data blocks in the appropriate include lists.
SELECT	Adds names to the omit, search, and refer lists.
MOVE	Reorders the include entry name list.
SATISFY	Resolves entry point names in the unresolved entry point list and data block names in the unresolved data list.
CLEAR	Clears (resets to empty) the include, omit, search, refer, unresolved reference, and modify lists.
MODIFY	Creates the modify list by specifying a set of modifications to be made when the target file is built.

Include Lists

The include lists are ordered lists of code blocks, data blocks, and entry points that are to be included in the target file. Code block, data block, and entry point names are added to the applicable include list by ADD or REPLACE commands in the order specified. Names from subsequent ADD or REPLACE commands are added to the end of the list.

ADD and REPLACE commands can refer to entry points explicitly by name or implicitly as part of a range of entry point names. Inserting a range of entry points specifies that entry points in the file (in physical order) from the beginning to the end of the range are to be added to the include list. All primary entry point names within the range are added to both the include code block list and the include entry point list; secondary entry point names are only added to the include entry point list. Names of own data blocks for the code blocks are added to the include data block list.

If a code block added to the include code block list contains a reference to an entry point or data block that is not in the respective include list, the name is added to the unresolved reference list. BINDER attempts to resolve the reference when a SATISFY or BUILD command is executed. (You can prevent automatic resolution by using the SELECT SATISFY OFF command.)

INCLUDE CODE BLOCK LIST. A code block name is added to the include code block list when one of the following happens:

- an ADD or REPLACE command refers to the code block (primary entry point) name either explicitly or implicitly as part of a range of entry points
- an ADD or REPLACE command refers to the total contents of a disc file containing the code block
- a SATISFY or BUILD command is executed and the following are true:
  - an entry point, referenced by an included code block, is in the unresolved reference list
  - the entry point name is not in the omit list
  - the code block containing the entry point that can satisfy the reference is in a disc file in the search list.

The order of code block names in the include code block list determines the order in which code blocks are allocated during the output phase.

The include code block list can be reordered during the input phase by the MOVE command, which allows code blocks to be ordered in such a way that page faults are avoided during execution. (Refer to the XRAY Users Manual for diagnostic information to use MOVE in this way.)

INCLUDE DATA BLOCK LIST. A data block name is added to the include data block list when one of the following happens:

- the data block is the own block of a code block in the include code block list
- the data block is a common block and an ADD or REPLACE command refers to the data block name either explicitly or implicitly as part of a range of data blocks; ordering is not necessary
- the data block is a common block that is referred to by a code block in the include code block list
- the data block is part of a disc file whose total contents are referred to by an ADD or REPLACE command
- the data block is a special data block required by a code block in the include code block list.

INCLUDE ENTRY POINT LIST. An entry point name is added to the include entry point list if:

- an ADD or REPLACE command refers to the entry point name either explicitly or implicitly as part of a range of entry points, or
- an ADD or REPLACE command refers to the total contents of a disc file containing the entry point. Secondary entry points, as well as the primary entry points, are added to the list.

The order of entry point names in this list corresponds to the order of code block names in the include code block list. An entry point name is positioned in the order of the ADD or REPLACE command that refers to it.

Omit List

The omit list contains the names of entry points that are not to be included in the target file even if referred to by included code. (Likewise, it doesn't matter whether the files on the search list contain the entry points.) The omit list can be used to force an entry point to be satisfied at run time, either from a system library or from a user library on the NonStop II system.

An entry point name that is already in the include entry point list cannot be specified for the omit list.

The SELECT OMIT command specifies entries for the omit list. The omit list can be overridden by specifying a different omit list in the SATISFY or BUILD command. (The SATISFY respecification is in effect only during execution of the SATISFY command.)

Giving the BUILD command implies a SATISFY command. The exception is if SELECT SATISFY OFF is specified, which prevents automatic resolution of entry point references. Note that the SATISFY command refers to entry points and data blocks; SELECT SATISFY refers only to entry points.

Clear the current omit list by using the RESELECT OMIT command.

Refer List

The refer list is a list of pairs of entry point names. The first entry point name of the pair is the existing name used to refer to an entry point (the old name). The second entry point name of the pair is the name of an entry point (the new name) that is to be substituted for the existing entry point name. They cannot have the same names.

When a SATISFY or BUILD command is executed, unresolved entry point references are checked against the refer list. If the reference is to be changed, BINDER makes the change and then attempts to resolve the reference. The refer list does not apply to entry points previously added to the include entry point list.

The refer list is constructed according to specifications in the REFER parameter of the SELECT command. If the REFER parameter is specified in the SATISFY or BUILD command, however, that refer list overrides the refer list of the SELECT command during execution of the SATISFY or BUILD command. The refer list can be cleared by the REFER parameter of the RESELECT command.

Search List

The search list contains the disc file names of object files to search in order to resolve entry point and data block references. This list is used during execution of a SATISFY or BUILD command. The files are searched in the order in which the file names are listed for blocks that define entry points and data blocks in the unresolved reference list.

The search list is constructed according to specifications in the SEARCH parameter of the SELECT command. If the SATISFY or BUILD command includes the SEARCH parameter, however, the specified file names become the search list and, during execution of the SATISFY or BUILD command, that list overrides any search list constructed by the SELECT command.

Unresolved Reference Lists

The unresolved reference lists contain entry point names and data block names that are referred to but are not defined by any block in the include lists. When a SATISFY or BUILD command is executed, BINDER attempts to resolve each entry point name in the unresolved reference list as follows:

1. The refer list is searched for a redirection of the entry point.

2. The omit list is searched; if the name is found in the omit list, the entry point name is not to be resolved. No further action takes place for that reference and the entry point name remains in the unresolved reference list.

3. The file containing the code block that referred to the entry point is searched and then the object files named in the search list are searched for an entry point that satisfies the reference; object files are searched in the order of the file names in the search list. If the entry point is found, the code block that defines the entry point is added to the end of the include code block list and the entry point name is deleted from the unresolved reference list.

4. If adding a code block to the include code block list introduces further unresolved references, the refer list is checked for redirection of those entry points. If an entry point is redirected and can be resolved by the include entry point list, no further action takes place; otherwise, the entry point name is added to the unresolved reference list.

When the target file is constructed, any unresolved entry point reference becomes an external reference that must be satisfied either by a subsequent BINDER operation or at run time.

For the languages currently supported by BINDER, no data block names should appear in the unresolved reference list. When the unresolved reference list contains data block names, BINDER attempts to resolve each reference by searching the file containing the code block that referenced the common block.

Modify List

The contents of an existing code block or data block can be changed when the target file is built. During the input stage, modifications are specified by the MODIFY command. BINDER saves the set of modifications established by this command in the modify list and makes the actual changes when the target file is built. Uninitialized data blocks cannot be modified.

OUTPUT STAGE

The output stage begins when the BUILD command causes an implicit satisfy for unresolved references. During this stage, BINDER builds the target file according to the names in the include lists, writes out listings, and clears all of the internal lists (include, omit, refer, search, unresolved reference, and modify) in preparation for building another object file.

Target File Characteristics

BINDER builds the target file using the attributes entered in the SET command or in the set-specification parameter of the BUILD command. These attributes are:

DATA - specifies the total amount of data space to be allocated for data blocks and for local storage.

STACK - specifies the amount of stack space to be allocated for local storage. This parameter value is added to the amount of space required for all data blocks, and the total amount of data space is then allocated.

EXTENDSTACK - specifies an amount of stack space to be added to the amount of stack space estimated by BINDER. The total amount of stack space and total data block space is then allocated.

Only one of DATA, STACK, and EXTENDSTACK can be used to override the default amount of data space allocated by BINDER; these parameters are mutually exclusive. The default data space allocated by BINDER is the amount of space required for all of the data blocks plus an estimated amount of stack space for local storage.

LIBRARY - (NonStop II systems only) specifies the user library that is to be associated with the object file at run time. If the LIBRARY parameter is not specified, no user library is associated with the object file. This file attribute can be overridden at run time by specifying the LIB parameter in the COMINT RUN command.

PEP - explicitly specifies a larger size for the PEP table. By default, BINDER allocates the minimum amount of space needed for the number of entry points in the object file.

INSPECT - specifies whether the INSPECT program or the DEBUG program is to be used for debugging when the object code is executed. The default is the DEBUG program.

SYMBOLS - specifies whether the symbol tables of blocks on the include lists are to be retained in the target file. (When the ?SYMBOLS compiler directive is used, BINDER builds a symbol table into the object file.) SYMBOLS ON is the default and specifies that the tables are to be retained in the object file.

SAVEABEND - determines whether a save file is created if the process terminates abnormally during execution. BINDER verifies that INSPECT is selected if SAVEABEND is ON. The SAVEABEND default is OFF. The save file is created in the same volume and subvolume as the program and has a name in the format of ZZSAnnnn. This file contains information on the process environment at the point of termination including:

- names of all open files
- a copy of the data space at the time the process terminated
- name of the process and a timestamp for the time of termination.

The INSPECT tool also provides for saving the environment of a process as well as examining save files. Refer to the INSPECT Users Manual descriptions of the SAVE and PR commands.

Target File Construction

When the BUILD command is executed, BINDER creates the target file according to the code block, data block, and entry point names currently in the include lists. The following steps are performed in creating the target file:

1. BINDER attempts to satisfy entry point and data block references in the unresolved reference list.
2. Entry points that cannot be resolved are made external references for the object file.
3. Any uninitialized common block is allocated an area preset to zero.
4. Blocks named in the include lists are copied to the target file.

Code blocks are allocated contiguously in the order in which the code block names appear in the include code block list. (The RESIDENT attribute overrides the include list order.) Code blocks and read-only string arrays cannot straddle the 32K boundary. When this would occur, the code block is relocated to begin at the upper 32K boundary and a gap is left following the previously allocated code block. BINDER issues an informational message when such a gap occurs during code block allocation.

If the COMPACT parameter of the SELECT command is set to ON (the default), BINDER checks each succeeding code block to determine whether it will fit in the gap. When a code block that fits in the gap is found, the code block is allocated in that gap. This can result in nonresident code blocks preceding resident code blocks.

Another SELECT command parameter that affects construction of the target file is the CHECK parameter. This parameter selects the different types of error checking that BINDER provides during the binding operation. The available options are as follows:

- The BLOCK option causes common block declarations to be checked for the same length and the same type of addressing in every code block that refers to the common block. If the length and type do not match for each of the references, a warning message is issued. Default is ON.

- The LIBRARY option causes the following checking to occur when a user library is being constructed on the NonStop II System. Code blocks being placed into the object file are checked for references to common blocks other than read-only blocks. Code blocks are also checked for the MAIN attribute. In either case, a warning message is issued. Default is OFF.

- The PARAMETER option causes parameter list checking between code blocks. It also controls function return type checking. The parameter lists for the respective code blocks are checked for consistency in size, type, and mode. Mode checking refers to whether the parameter is passed by value or by reference. COBOL program units and FORTRAN subprograms receive parameters passed by reference only; TAL procedures can receive parameters passed either by value or by reference. A warning message is issued if the called code block's parameter requirements do not match the passed parameters; BINDER does not insert the called block if it is not already on the include list. Default is ON.

NOTE

The parameter checking in BINDER is stricter than in the old compilers. You may receive parameter mismatch errors when creating an object file with BINDER that you would not receive when creating the object file with a pre-BINDER compiler. In particular, the parameter mismatch messages are likely to occur in mixed-language binding.

MIXED-LANGUAGE BINDING

Code blocks written in different languages can be bound together into a single object file. There are certain restrictions if COBOL and FORTRAN are both used in the same program. Likewise, caution is necessary when mixing TAL procedures with COBOL and FORTRAN. These restrictions arise because of the differences in the run-time environments of COBOL, FORTRAN, and TAL, notably in the area of I/O.

COBOL and FORTRAN

When an object file is created using BINDER commands, the program unit concepts of the COBOL and FORTRAN compilers are retained. Each COBOL code block in the include code block list is a separate program unit. All FORTRAN code blocks that are bound together form one program unit. Based on these concepts, BINDER constructs all the control blocks needed to execute the program. These control blocks include:

- one run unit control block for the target file
- one program unit control block for each COBOL program unit in the target file; the PUCB includes the file control blocks
- one program unit control block for the FORTRAN program unit in the object file
- one file control block for each file (logical unit) in the set of files for the FORTRAN program unit in the object file
- FORTRAN logical unit table.

When COBOL and FORTRAN code is included in the same target file, the following restrictions apply:

- The MAIN program unit must be from COBOL source. (This program unit can be a skeleton program containing only an ENTER statement.) The COBOL MAIN program unit is needed to perform run time initialization. Other code blocks in the target file can be written in COBOL, FORTRAN, or TAL.
- The MAIN program unit initializes all COBOL and FORTRAN file control blocks for the run unit.
- NonStop processes must provide for checkpoints by both FORTRAN and COBOL blocks prior to calling a block in the other language (if checkpointing can occur in both COBOL and FORTRAN blocks). The checkpoint should include the state of files and data.
- COBOL and FORTRAN code should not share files; the two languages frequently require different protocols for device I/O.

- Parameter passing - care is required if replacing FORTRAN code with COBOL code, or COBOL code with FORTRAN.

TAL with COBOL and FORTRAN

TAL offers complete flexibility in performing I/O (via direct calls to the GUARDIAN operating system). Therefore, TAL and COBOL or FORTRAN mixed-language programs must be written with great care if TAL shares files with either COBOL or FORTRAN. TAL procedures must not interfere with the COBOL and FORTRAN protocols, especially SETMODEs.

In mixed-language programs combining TAL with FORTRAN, if FORTRAN performs I/O, the main program must be coded in FORTRAN and must have the MAIN attribute in order to include the special data blocks necessary for I/O. Otherwise, FORMATTER errors occur at run time.

Example of a COBOL MAIN Skeleton Program Unit

As mentioned earlier, if any COBOL blocks are included the block with the MAIN attribute must be a COBOL program unit. Below is a sample "skeleton" program unit. In this example, the COBOL blocks are mixed with TAL blocks.

```
IDENTIFICATION DIVISION.
 PROGRAM-ID. QUEENSCO.
 AUTHOR. PROGRAMMER, JOHN R.
 INSTALLATION. TANDEM COMPUTERS, INC.
 DATE-COMPILED.
ENVIRONMENT DIVISION.
 CONFIGURATION SECTION.
 SOURCE-COMPUTER. TANDEM/16.
 OBJECT-COMPUTER. TANDEM/16.
SPECIAL-NAMES.
    FILE QUEENSO IS TALLIB.

DATA DIVISION.

PROCEDURE DIVISION.
START-UP.
    ENTER TAL SOLVE OF TALLIB.
    STOP RUN.
```

APPENDIX A

BASIC COMMANDS

BINDER supports basic commands and automatic file name expansion within BINDER commands. This appendix describes the basic commands in alphabetic order. They are:

ENV	HELP	OUT
EXIT	LOG	SYSTEM
FC	OBEY	VOLUME

FILE NAME EXPANSION

BINDER assumes that file names supplied for input and output follow GUARDIAN naming conventions. Defaults are supplied by the GUARDIAN Command Interpreter when BINDER is started.

File names are assigned to all disc files and devices. Running processes can be named at your discretion. Refer to the GUARDIAN Operating System Programming Manual for details.

Expanded Disc File Names

Disc files of any type are identified, and located, via the expanded file name. File name expansion assumes the following:

\system-name	identifies a system within a network
$volume-name	identifies a physical disc pack mounted on a disc drive
subvolume-name	identifies a group of related files defined by the user
disc-file-name	identifies a single file in the subvolume.

A fully expanded disc file name has the form:

\system-name.$volume-name.subvolume-name.disc-file-name

If a partial file name is supplied as a command parameter, the file name is expanded into the internal four-part representation. To guarantee correct file name expansion, at least disc-file-name must be supplied.

Process and Device Names

Each process and each device, such as a tape drive or printer, is identified in a similar manner. For example:

```
\YOUR.$TAPE1
```

might specify a particular tape drive on system \YOUR; when operations are already on that system, only $TAPE1 is required.

Command Entry

Multiple commands can be entered on the same line if they are separated by the semicolon character (;). If a COMMENT command is entered, it must be the last command on the line.

To continue a command on the next line, enter the ampersand (&) as the last nonblank character of the current line. The maximum line length, including continuation characters, is 132 characters. The maximum length of a continued command record is 528 characters.

BASIC COMMAND DESCRIPTIONS

ENV Command

The ENV command displays the current settings of program environment parameters.

```
ENV [ LOG    ]
    [ SYSTEM ]
    [ VOLUME ]
```

If no options are specified, the values for all parameters are displayed.

EXIT Command

The EXIT command stops the BIND process. If a previous BUILD command has not been issued, no target file is created.

```
EXIT
```

Entering CTRL/Y also stops the process immediately.

FC Command

The FC command operates the same as the COMINT FC command in allowing editing and repetition of the last command line.

```
FC
```

HELP Command

The HELP command displays BINDER commands and syntax.

```
HELP [command-name                    ]
     [[ < ] param-name [ > ]           ]
```

where

command-name

is a valid BINDER command name.

param-name

is a command parameter as displayed in a HELP command-name command.

If no parameters are used, BINDER displays the names of all commands.

The correct spelling for a particular param name is most easily found by first specifying the command-name parameter for the command the parameter is associated with. For example:

```
@HELP DELETE
DELETE { { CODE | DATA } <block-list> | * }
@HELP block-list
```

LOG Command

The LOG command records the session input and output on a permanent file.

```
LOG  { TO file-name }
     { STOP         }
```

where

file-name

identifies a file to receive the copy of commands and output; if the file does not exist, a disc file is created with name *file-name*.

Logging is initiated when the command specifies a file name. If logging is already in progress, the previous log file is closed and logging begins on the new file. If *file-name* is the same as the previous log file, the LOG command is ignored and logging continues on the same file.

If *file-name* has the form of a disc file and the file does not exist, an EDIT file is created. If the named file is an existing disc file, the output is appended to the file.

The current log file is closed and all logging is stopped when the LOG STOP command is entered.

OBEY Command

The OBEY command causes commands to be read from a specified file.

```
OBEY file-name

where

  file-name

    is the file name of the OBEY file.
```

Commands are read from the named file and processed until an end-of-file is encountered. At end-of-file, the OBEY file is closed and command input reverts to the previous input file, normally the home terminal.

Additional OBEY commands can appear within an OBEY file; OBEY files can be nested to a depth of four.

If the default setting of SYSTEM or VOLUME is changed in an OBEY file, it is effective for all following commands. The setting is not automatically returned to the previous state when the OBEY terminates.

If any part of the specification is invalid, if the file does not exist, or if the file cannot be opened, an error occurs. BINDER displays an error message and prompts for input if the input file is a terminal. If the input file was not a terminal, BINDER terminates.

OUT Command

The OUT command directs the output listing to a specified file. The syntax of the OUT command is:

```
{OUT file-name     }
{/ OUT file-name /}
```

where

file-name

is a file name.

The first form of the OUT command causes permanent redirection of the output.

The second form of the OUT command causes temporary redirection of the output. This form is specified as part of another command and must be positioned immediately after the other command name and before any other part of that command. For example:

```
HELP/OUT file-name/BUILD
```

If the file name has the form of a disc file and the file does not exist, an EDIT file is created. If the named file is an existing disc file, the output is appended to the file.

If the file name is invalid or if the file cannot be opened, an error occurs. An error message is displayed and the command is not executed.

SYSTEM Command

The SYSTEM command sets the default system for expansion of any file names.

```
SYSTEM [ \system ]

where

  system

    is a system name of the form \system.
```

VOLUME Command

The VOLUME command sets the default volume and subvolume for expansion of any file names.

```
VOLUME { $volume                 }
       { [ $volume. ] subvol     }

where

  volume

    is a volume name of the form $volume.

  subvol

    names a subvolume on volume.
```

APPENDIX B

BINDER ERROR MESSAGES

The Program Development Tools (PDT) all have an error file called PDTERROR associated with them. This file must reside on the same volume and subvolume as the PDT products. As part of the standard release, they are automatically on $SYSTEM.SYSTEM. Users who chose to relocate the PDT products on some other volume and subvolume must be sure also to relocate PDTERROR (and PDTHELP). When BINDER detects an error condition, it searches PDTERROR for the corresponding error message (documented below). If PDTERROR is missing, BINDER displays a cryptic message of the form BINDER ERROR nn.

BINDER error messages are listed alphabetically with descriptive information and a message severity code. Message severity codes are as follows:

E Error -- command cannot be executed; correct and reenter.
F Fatal error -- BINDER internal error or temporary file error occurred; notify your Tandem representative.
W Warning -- command was executed; result can be unexpected.

Messages without a severity code just give information on command results. BINDER uses both upper and lower case to display messages.

E ADD/REPLACE/DELETE may not be used on OWN (SAVE) blocks

An OWN (SAVE) block cannot be specified separately from its associated code block. Adding, replacing, or deleting the code block automatically adds, replaces, or deletes, respectively, the own block.

E ADD/REPLACE/DELETE may not be used on PUCB blocks

These commands cannot refer to a COBOL PUCB by name.

E Address overflow

BINDER has run out of virtual memory space. The BIND session may simply be too long, in which case it can be broken up into smaller sessions.

E Alter specification reused

ALTER options cannot be specified more than once in a command line.

W Alter to CALLABLE and not PRIV means CALLABLE and PRIV: entry-point-name

Callable procedures must also be privileged code; BINDER automatically sets PRIV ON if you specify CALLABLE ON. PRIV OFF is overridden in this case. (Note that BINDER action is different from UPDATE.)

F An OWN block or COBOL PUCB has been included in the object file without the associated code for the data block: block-name

When an OWN block or COBOL PUCB is included in an object file, the associated code for the data block must also be included.

E Bad object file format version: file-name

The object file must be in the standard BINDER format. If the code was produced by an earlier, incompatible version of the compiler, you must recompile.

E Block does not exist in file: block-name

No code or data block by the given name was found in the object file specified.

1. Verify block names by using the LIST LOC command, or refer to a current compiler listing

2. Ensure that BINDER searches the correct file by giving the file name on the command or use the FILE command.

E Block has no contents

The data block specified in the DUMP or MODIFY command is not initialized.

E Block is not on the include list

The block specified in the MODIFY command is not in the include list (code or data) indicated by the command.

W Block length/address mode error on data-block-name

The CHECK BLOCK parameter (SELECT command) is set to ON, and common or global declarations are inconsistent in length or addressing mode. The data block reference is unresolved; you can:

1. correct the inconsistency, or

2. use SELECT CHECK BLOCK OFF to allow resolution.

E Cannot add entry point to omit list if already on include list: entry-point-name

An entry point cannot be on both lists. If the include list is wrong, use the DELETE command.

E Cannot alter MAIN attribute of non-TAL PROC code-block-name

FOR COBOL and FORTRAN code blocks, you cannot change MAIN after compilation. (Be sure that the ALTER LIKE command is used only if both code blocks have the same MAIN characteristic.)

W Cannot create file, using OBJECT instead: file-name (error-num)

file-name failed for the target file. error-num is the GUARDIAN file error received. BINDER attempts to use the name OBJECT. Refer to the GUARDIAN Operating System Programming Manual for information on error-num.

E/W Cannot create file: vol.subvol.OBJECT (error-num)

The attempt to use OBJECT to name the target file failed. error-num is the GUARDIAN file error received. BINDER then attempts to create the target file with a name of the form ZZBInnnn. If the attempt fails, the target file definition is lost (and, in interactive mode, BINDER prompts for input). Refer to the GUARDIAN Operating System Programming Manual for information on error-num.

E Cannot satisfy references since include lists are empty

Code and data block names and entry point names must be added to the include lists before a SATISFY or BUILD command is entered. If a BUILD command was entered first, BINDER cleared all previous specifications, such as SELECT, SET, and FILE entries.

E CHANGE file cannot be open (use CLEAR command)

Since the CHANGE command writes to the object file, the object file must not be open when this command is executed. BINDER may be using the file, in which case the CLEAR command closes it.

W Code block already on include list: entry-point-name

Your ADD command named a code block that is already on the include list. An entry point name cannot be on the include list more than once. If two object files contain the same code block name, BINDER uses the first occurrence.

W Code block moved to above 32K to avoid straddling 32K: code-block-name

A code block has been moved to begin at the 32K boundary. If the COMPACT OFF parameter (SELECT command) is in effect, all code blocks that follow on the include list also follow in the code area of the target file. If COMPACT ON is selected, smaller blocks can be relocated to the gap below 32K. (This may result in nonresident code placed before a resident code block.)

This message is issued only once.

E Code space overflow in PROC: code-block-name. Code block length is: length

The program exceeded the 64K maximum of code space available.

W Control data space overflow

COBOL -- largest possible configuration exceeds available memory.

W Data block already on include list: data-block-name

A FORTRAN COMMON or TAL BLOCK of the given name has been inserted on the include list already. BINDER uses the first occurrence of the block name if two object files contain the same name.

E Data block cannot be allocated: data-block-name. Data block trying to fit data-space-location

No room is available for the data block in the correct part of memory for the block type. Reorganize the data space to make room for the specified block. data-space-location can be one of: FIXED POSITION, BELOW 256, BELOW 32K, ABOVE 32K, BELOW 32K LAST, ABOVE 32K LAST, BELOW 32K PENULT, BELOW 64, ANYWHERE, EXTENDED ADDRESS.

E Data reference failed due to relocation: (code-block-name) + (offset) REF TO (data-block-name) + (offset)

TAL -- BLOCK global variable may be moved past TAL limit. code-block-name contains the data reference. Rearrange global variables, or use the ?INHIBITXX directive. Refer to the TAL Reference Manual for information on the ?INHIBITXX directive.

E Effective input record is too long

The total command line cannot exceed 528 characters, excluding continuation characters. Respecify as more than one command.

W Entry point already on include list: entry-point-name

The entry point was already inserted on the include list. If the second occurrence should be the correct one, use the REPLACE command to replace the first occurrence.

W Entry point cannot be resolved due to conflict with another procedure: entry-point-name

The entry point has also been used as a secondary entry point name in another code block that is already on the include list.

W File already on search list file-name

The file name was previously used in a SELECT SEARCH command. The file name is ignored.

E File code not 100: file-name

The indicated file is not an object file. If it is a source file, it must be compiled before being passed to BINDER.

E Identifier too long

An identifier cannot exceed 31 characters.

E Illegal ALTER PROC name: proc-name

The entry point name given as an ALTER LIKE entry point is not on the include list.

E Illegal block range member: name OR *

If the * option was specified, the include list is empty; otherwise, the range specifies a block name that is not in the include list.

E Illegal DUMP offset(s)

The offset and length specified in the DUMP command exceeds the block length. Reenter the command with the * option or with the correct length and offset.

E Illegal DUMP specification

The ICODE dump specification cannot be specified for a data block.

E Illegal log file - ignored

The LOG file name was used for one of: IN file, OUT file, or OBEY file.

E Illegal MOVE PROC name: code-block-name

A MOVE command cannot refer to a block that is not on the include code block list or to a block that is inside a range of blocks to move.

E Illegal obey file - ignored

The OBEY command file name was used as one of: IN file, OUT file, or LOG file.

E Illegal offset for MODIFY/VERIFY command

The offset specified in the MODIFY or VERIFY command is beyond the end of the block.

E Illegal OUT file - ignored

The OUT file name was used as one of: IN file, OBEY file, or LOG file name.

E Illegal SELECT REFER pair - ignored

Both names of a refer pair cannot be the same. Also, a "new" name cannot have been used as the "old" name of another refer pair; likewise, the "old" name cannot be the "new" name of another refer pair.

E Illegal SET value - ignored

The maximum values for SET parameters are:

PEP - 512	DATA EXTENDSTACK STACK	64 pages (65536 words)

E Integer conversion error

Either the number supplied is too large, or an invalid digit was received.

F Internal error at P=%nnnnnn text

This message precedes other information about a BINDER internal consistency error; a stack trace usually accompanies the message. Contact your Tandem representative.

E Invalid file name file-name

The file name does not conform to GUARDIAN standards.

E Invalid subvolume name

The volume or subvolume name is too long or contains an invalid character.

E Invalid syntax

The sequence of input characters does not result in a valid BINDER command. A caret (^) indicates the detected error.

E Invalid system name

The system name is too long or contains an invalid character, or the system does not exist.

F MAIN entry point found in a search file: entry-point-name

In attempting to include a procedure from a search file (because it has been directly or indirectly referenced by the program being compiled), BINSERV has discovered that the procedure has the MAIN attribute. This is not allowed.

W MODIFY overrides reference to another block in block-name

The MODIFY command resulted in a change to a CALL or to a reference to global data. This message is issued both at the time of the MODIFY and if the block is added to an include list in a later BINDER session.

E More than 510 entry points on include list nnn

The procedure entry point table can have a maximum of 510 entries, one for each entry point. You must restructure the program.

E More than 512 entry points on unresolved list nnn

The maximum number of unresolved external references has been exceeded. Some of these unresolved references must be satisfied before the object file can be built.

E Multiple MAIN entry points are not allowed: entry-point-name

Only one entry point may have the MAIN attribute.

E No current file for ADD/REPLACE/LIST/DUMP

You must establish a current file before the named commands can be executed. Either reenter the command with the FROM file-name parameter or use the FILE command. FILE establishes the default file for subsequent commands (or until another FILE is entered).

E No help available for name

If a command parameter is queried, it must be specified in the form known to BINDER's HELP command. Use HELP command-name to verify the parameter forms. HELP with no parameters lists all the BINDER commands.

W No parameter information provided for entry-point-name

COBOL -- A CANCEL for the entry point name was encountered; however, no CALL statement referred to the entry point. Therefore, parameters could not be checked.

E No space left for stack after data block allocation

Word 32767 is used in the word-addressable data space. BINDER cannot allocate the stack. Reorganize the data space.

E Not enough space for XEP table; n words required, m words available

The External Entry Point table cannot be placed in the code space.

W Object file is named $vol.subvol.ZZBInnnn

BINDER assigned an arbitrary name after other file naming attempts failed for the target file. nnnn is a numeric suffix that uniquely identifies the file. vol and subvol are the same as would have been used. This is an informative message only.

E Obey nesting exceeds maximum

Four levels of nesting is the maximum allowed in an obey.

E Old format object file: file-name

1. The file was compiled by an earlier compiler version without BINDER. Recompile using the correct version. Check with your installation's system support personnel.

2. The file previously had BINDER tables but has been stripped (by a BINDER STRIP command). BINDER cannot manipulate the file in any way; recompile.

W Old object file has been renamed to file-name

This warning indicates the new name of the existing object file. The renaming makes way for the new object file to be given the name specified by the user. If the old file is not in use, it is purged. If the old file is in use, it is renamed.

E Old version of INSPECT information for entry-point-name

Symbol table incompatible with the current version of INSPECT. Recompile the program containing entry-point-name.

F Paging file error file-error-message(nnn) or ALLOCATESEGMENT error-code nnn

Refer to the GUARDIAN Operating System Programming Manual for a description of the indicated error type.

W Parameter count mismatch on entry-point-name

SELECT CHECK PARAMETER ON is in effect (default), and the consistency check failed. The parameters required by the named entry point are inconsistent with the call; the inconsistency is in the number of parameters. This message is also issued if calls to the same entry point from separately compiled code blocks are inconsistent in parameter passing.

W Parameter mode mismatch on entry-point-name parameter n

SELECT CHECK PARAMETER ON is in effect (default), and the consistency check failed. The parameters required by the named entry point are inconsistent with the call; the inconsistency is in the mode of the parameter. Parameter mode is by-value, by-reference, or extended-reference. This message is also issued if calls to the same entry point from separately compiled code blocks are inconsistent in parameter passing.

W Parameter type mismatch on entry-point-name parameter n

SELECT CHECK PARAMETER ON is in effect (default), and the consistency check failed. The parameters required by the named entry point are inconsistent with the call; the inconsistency is in the type of the indicated parameter. This message is also issued if calls to the same entry point from separately compiled code blocks are inconsistent in parameter passing. BINDER goes on to inform the user whether the mismatch was between two external declarations of a procedure or between an external declaration and the actual procedure declaration. BINDER detects when at least one parameter is an INT, INT(32), or STRING and informs the user. If only one parameter is an INT, INT(32), or STRING, BINDER designates the parameter that is not any of the three types as "other". If neither parameter is INT, INT(32), or STRING, BINDER outputs more detailed information on the mismatch. This information includes one or more of the following: parameter storage type, parameter length in bits, number of units, and number of digits to the right of the decimal point. Parameter checking can be disabled by setting CHECK PARAMETER OFF.

E Primary global area overflow

FORTRAN -- too many variables in FORTRAN COMMON are referenced. Recompile using the ?EXTENDCOMMON compiler directive.

TAL -- Too many global variables defined.

W Procedure references absolute global data which may be relocated: code-block-name

TAL -- ?RELOCATE compiler directive was used and relocation may have occurred. The #GLOBAL block is not at offset zero. The compiler and command-driven BINDER issue warnings.

E Range members in wrong order: name to name

If ADD or REPLACE command has an incorrect range given, respecify block names in ascending order by location in the object file. Use LIST LOC command or a current map listing to verify the order of blocks in the input file.

If the command refers to a range in an include list, also respecify the command with names in ascending order as on the list. The INFO INCLUDE command can be used.

W Read-only data block moved to above 32K to avoid straddling 32K: data-block-name

TAL -- a string P-relative array was moved.

F Reference to a block expected, but not in, G-relative address area block-name + offset REF TO block-name + offset

BINDER was expecting a G-relative address reference for a block but received an address outside of G-relative space.

W Reference to string P-relative in wrong half of code space may fail: proc-name + offset REF TO block-name + offset

TAL -- BINDER cannot verify that references to the array are valid. Note that proc-name + offset is a fix-up word rather than as indicated in the listing.

E RENAME {code / data} block is not on the include list

The code or data block specified in a RENAME command must be on the include list.

W Return type mismatch on proc-name

SELECT CHECK PARAMETER ON is in effect (default), and the consistency check failed. The return type required by the named procedure is inconsistent with the call. This message is also issued if calls to the same entry point from separately compiled code blocks are inconsistent in parameter passing. Parameter checking can be disabled by setting CHECK PARAMETER OFF.

F Storage file error file-error-message (nnn)

An error occurred in the temporary work file that BINDER uses to hold code and data until the object file is built. Refer to the GUARDIAN Operating System Programming Manual for a description of the error.

E STRIP command cannot be used on this file

A valid old-format object file cannot be constructed for this file.

E STRIP file cannot be open (use CLEAR command)

BINDER is currently using the file for prior commands (for example, FILE). CLEAR resets BINDER to the initial state; if no other opens are current for the file, it can then be stripped.

F Symbol file error: file-error-message (nnn)

An error occurred in the temporary work file that BINDER uses to hold symbol information until the object file is built. Refer to the GUARDIAN Operating System Programming Manual for a description of the error.

F The attributes of two FCBS for the same data block do not match block-name

The attributes of FCBs for the same data block must match. This could be a user error but is more likely a compiler error. Contact your Tandem representative.

F The compiler has used the same unique ID for two different names

This message indicates a BINDER internal error. Contact your Tandem representative.

E The MAIN must be in COBOL since COBOL procedures are present

Mixed-language binding with COBOL code requires that the MAIN program be COBOL to allow correct initialization for the COBOL syntax error code. Consider adding a skeleton program that contains a call to the major entry point. (Only one block with the MAIN attribute is allowed.)

W TNS/II user library violation: MAIN procedure code-block-name

SELECT CHECK LIBRARY ON is in effect and a procedure with MAIN attribute was encountered.

W TNS/II user library violation: referencing data block block-name

A data block with read-write access was added to the include list and CHECK LIBRARY ON is in effect. Since this can affect data memory organization, the message is issued.

E Unsatisfied reference to a data block

Before a BUILD command is entered, the unresolved data reference list must be empty. There are still unresolved data block references. Use INFO UNRESOLVED DATA command to check the list.

E Unterminated continuation line

End-of-file was encountered while scanning for the remainder of a continuation line.

E Unterminated string

The ASCII input to a MODIFY command is missing the closing quotation mark.

F Value specified in VERIFY command not equal to current value: should be %value, is %value

The VERIFY command can be used during a non-interactive BIND session to stop the session if a discrepancy exists between an expected code or data value and the actual value. This message indicates what value did not check out.

W VARIABLE attribute mismatch on proc-name

TAL -- different callers disagreed about the VARIABLE attribute of an entry point, or caller and callee disagreed about the callee's VARIABLE attribute.

F Work file error: file-error-message(nnn)

The work file is the object file being constructed. Refer to the GUARDIAN Operating System Programming Manual for a description of the error type.

APPENDIX C

SYNTAX SUMMARY FOR SESSION COMMANDS

BINDER command syntax is summarized in this appendix. Detailed information for each command is referenced by page number.

```
ADD { CODE entry-name-list  }  [ FROM file-name ] [ , DELETE ]
    { DATA block-name-list  }
    { *                     }
```
3-6

```
ALTER entry-name-list , alter-spec [ , alter-spec ] ...
```
3-9

where alter-spec is one of:

```
CALLABLE { ON | OFF }
LIKE entry-name
MAIN { ON | OFF }
PRIV { ON | OFF }
RESIDENT { ON | OFF }
```

```
BUILD [ file-name ] [ ! ]  [ , set-specification   ]  ...
                           [ , select-specification ]
```
3-11

where set-specification is one of:

```
{ DATA         }
{ STACK        }  value [ PAGES | WORDS ]
{ EXTENDSTACK  }
INSPECT { ON | OFF }
LIBRARY file-name
LIKE file-name
PEP number
SAVEABEND { ON | OFF }
SYMBOLS { ON | OFF }
```

where select-specification is one of:

```
CHECK check-number
COMPACT { ON | OFF }
LIST list-spec
OMIT entry-name-list
REFER refer-list
SATISFY { ON | OFF }
SEARCH file-list
```

```
CHANGE { DATA value [ PAGES | WORDS ]
       { INSPECT { ON | OFF }
       { LIBRARY file-name
       { SAVEABEND { ON | OFF }      } IN object-file
```
3-14

CLEAR — 3-16

COMMENT [text] — 3-16

```
DELETE { CODE block-name-list
       { DATA block-name-list
       { *
```
3-17

```
DUMP { CODE code-block-name } { offset [, count]
     { DATA data-block-name } {       [TO *]
                              { *
```
3-18

[dump-spec-list] [FROM file-name]

where dump-spec-list is:

```
dump-spec
( dump-spec [ , dump-spec ] ... )
```

where dump-spec is:

```
ASCII
HEX
DECIMAL
ICODE
OCTAL
```

```
ENV [LOG    ]
    [SYSTEM ]
    [VOLUME ]
```
A-3

EXIT — A-3

```
FC                                                              A-3

FILE file-name                                                  3-20

HELP  [command-name                ]                            A-4
      [[ < ] param-name [ > ]      ]

      {  INCLUDE  {CODE block-name-list  } [ , DETAIL ] }
      {           {DATA block-name-list  }              }
      {           {ENTRY entry-name-list }              }
      {           {*                     }              }
INFO  {                                                 }       3-21
      {                 DATA                            }
      {  UNRESOLVED     ENTRY                           }
      {                 *                               }
      {                                                 }
      {  * [ , DETAIL ]                                 }

      {  SOURCE                                         }
      {                                                 }
      {  CODE {address-list   }                         }
      {       {block-name-list}                         }
      {                                                 }
      {  DATA {address-list   }                         }
LIST  {       {block-name-list}                         } [ FROM file-name ] 3-24
      {                                                 }
      {  list-option  [ , BRIEF ]                       }
      {                                                 }
      {  ( list-option [ , list-option ] ...            }
      {                          [ , BRIEF ] )          }
      {                                                 }
      {  *  [ , BRIEF ]                                 }

   where list-option is one of:

      ALPHA
      LOC
      XREF
      *

LOG  {TO file-name}                                             A-5
     {STOP        }
```

```
MODIFY  { CODE code-block-name }  [ modify-spec ] [ offset ]          3-27
        { DATA data-block-name }

  [ , value ] ...

  where modify-spec is one of:

    ASCII
    HEX
    DECIMAL
    OCTAL

MOVE entry-name-list { AFTER | BEFORE } entry-name                    3-30

  [ , entry-name-list { AFTER | BEFORE } entry-name ] ...

OBEY file-name                                                        A-6

{ OUT file-name     }                                                 A-7
{ / OUT file-name / }

RENAME  { CODE code-block-name }  TO  name                            3-32
        { DATA data-block-name }

REPLACE { CODE entry-name-list }  [ FROM file-name ]                  3-33
        { DATA block-name-list }
        { *                    }

RESELECT { select-parameter [ , select-parameter ] ... }              3-35
         { *                                            }

RESET   { set-parameter [ , set-parameter ] ... }                     3-36
        { *                                     }

SATISFY [ select-specification [ , select-specification ] ... ]       3-38
```

```
        ( CHECK    {check-option                                    } )
        (          {( check-option [ , check-option ] ... )         } )
        (                                                             )
        ( COMPACT { ON | OFF }                                        )
        (                                                             )
        ( LIST    {list-option                                   }    )         3-40
        (         {( list-option [ , list-option ] ... )         }    )
SELECT  (                                                             ) , ...
        ( OMIT    {entry-name                                  }      )
        (         {( entry-name [ , entry-name ] ... )         }      )
        (                                                             )
        ( REFER   {refer-pair                                  }      )
        (         {( refer-pair [ , refer-pair ] ... )         }      )
        (                                                             )
        ( SATISFY { ON | OFF }                                        )
        (                                                             )
        ( SEARCH  {file-name                                   }      )
        (         {( file-name [ , file-name ] ... )           }      )

      ( {DATA       }                                  )
      ( {EXTENDSTACK}  value [ PAGES | WORDS ]         )
      ( {STACK      }                                  )
      ( INSPECT { ON | OFF }                           )
SET   ( LIBRARY file-name                              ) , ...          3-44
      ( LIKE                                           )
      ( PEP number                                     )
      ( SAVEABEND { ON | OFF }                         )
      ( SYMBOLS { ON | OFF }                           )

      ( FILE             )
      ( MODIFY           )
SHOW  ( SELECT           )                                              3-47
      ( select-parameter )
      ( SET              )
      ( set-parameter    )

STRIP file-name                                                         3-50

SYSTEM [ \system ]                                                      A-8

VERIFY  {CODE code-block-name}  [ verify-spec ]  [ offset ]             3-51
        {DATA data-block-name}

  , value

VOLUME  {$volume                }                                       A-8
        {[ $volume. ] subvol    }
```

APPENDIX D

NONSTOP II USER-LIBRARY INFORMATION

BINDER commands make it easier to use NonStop II user libraries. This appendix contains information needed for effective programming with user libraries. The information in this appendix is an overview and should not be considered exhaustive.

A user library contains procedures that the GUARDIAN operating system binds to the program file at run time. Run-time binding does not include copying the procedure into the program file, and a program file can have one (and only one) user library associated with it. Therefore, a running program can have 128K words of code space: 64K words for the program code space and 64K words for the library code space.

User libraries are sharable by multiple programs. Placing procedures that are commonly used in a user library can reduce the storage required for object code on disc and in main memory, while using the maximum code space.

GUARDIAN BINDING OF USER LIBRARY PROCEDURES

At run time, the GUARDIAN operating system searches the optional user library to resolve each unresolved external reference before searching the system library. (Instructions for specifying user libraries are given under "Program File Use of Libraries" in this appendix.)

GUARDIAN resolves an external reference by changing the call in the program file appropriately, that is, to point to the user library or to the system library. Then, the program file can be run repeatedly without satisfying the references again.

If no user or system library procedure can be found to satisfy a run-time external reference, an error message is displayed as the process is started. When the unresolvable external reference is reached, the process enters the debug state. (That is, if GUARDIAN encounters an unresolvable external reference, it changes the reference into a call to DEBUG.)

OBJECT FILE FORMAT

Since the GUARDIAN operating system binds program files and library procedures at execution time, there are no restrictions on object-file format. A program file and its library file can exist in any combination of new (BINDER) and old (GUARDIAN A03/E04 and before) formats.

PREVENTING BINDER RESOLUTION OF LIBRARY CALLS

If a user library will be used (for example, to avoid code-space overflow), care should be taken to prevent binding of library procedures into the object file before run time.

Compile-time Binding

The compile-time BINDER attempts to resolve external references if the compiler SEARCH directive specifies a list of object files for this purpose. One exception to this in COBOL is COBOL's use of the file CLIBOBJ (see the COBOL Programming Manual).

When a search list is present, the BINDER attempts to resolve all unresolved external references. If compile-time binding does result in a user-library procedure being in the program file, that procedure can be deleted from the program file in an independent BINDER session.

Command-Driven Binding

During a BINDER session, you can specify names of individual entry points that BINDER should not include in an object file. You can also build a user-library file of procedures copied from existing program files; then, you can build more compact program files by omitting the library procedures.

To specify the external references that BINDER should not resolve, you can use these commands as appropriate:

1. SELECT OMIT commands naming entry points of user library procedures. This prevents binding of these procedures in the program file.

2. BUILD command with the SATISFY OFF parameter option. This, however, prevents any attempt at resolution after the target file contents have been specified by other commands.

3. DELETE command to remove user-library procedures that were already bound in the program file.

PROGRAM FILE USE OF LIBRARIES

Only one user library can be associated with a program file at any time. Therefore, all concurrently executing processes created from a single program file use the same library file.

You can specify the user library for a program file by:

1. the new TAL-compiler directive LIBRARY
2. the BINDER command SET LIBRARY
3. the COMINT RUN command LIB option.

The library remains associated with the program file until explicitly changed. (That is, GUARDIAN changes to the program file for external-reference resolution remain when the process completes. The GUARDIAN operating system again resolves the calls left for resolution at NEWPROCESS time if it detects modifications to the program file or the user library when another RUN command occurs.)

To specify a different library with a program file:

1. write access to the program file is required, and
2. all processes created from the program file must have stopped.

LIBRARY FILE RESTRICTIONS

A library file should not contain a MAIN program. Beginning with the A05 release of GUARDIAN, this results in a run-time error.

Library procedures cannot call procedures in the program file.

BINDER does not currently support the inclusion of COBOL programs in user libraries. FORTRAN user libraries are supported as long as there are no DATA statements, SAVE statements or COMMON statements. However, both FORTRAN and COBOL programs can call TAL procedures contained in user libraries.

LIBRARY PROCEDURE DATA

The following restrictions on data declarations should be followed.

1. There is one data space (stack) for a process: the one allocated for the program file.

2. Since the allocation and initialization of global data are specified in the program file, if the library file has global data, it should match the global data of the program file. (The operating system ignores any initialization of global data in the library file.)

3. A TAL library procedure can have its own read-only arrays. Nevertheless, a global read-only array must be in the code space containing references to the array. If both code spaces contain references to such an array, copies of the array must both exist in the library and in the program file.

4. User-library procedures are not allowed to pass read-only array arguments to user code or to system code. In particular, a user-library call to FORMATDATA cannot pass read-only array arguments.

INDEX

ADD command 1-8, 3-1, 3-3, 3-6/8, 3-34, 4-9/11, C-1
  errors B-1, B-4, B-7, B-10
  examples of 2-3, 3-7/8, 3-10, 3-13, 3-31, 3-39
  UPDATE equivalent 1-12
Adding code or data
  see ADD command
Adding names to lists 3-6/7, 3-33, 3-40/43, 4-10/12
Address overflow B-1
AFTER parameter of MOVE command 3-30
Allocation of code area pages 4-7
ALPHA option
  of LIST command 3-25
  of SELECT command 3-42
ALTER command 3-1, 3-3, 3-9/10, 4-3, C-1
  errors B-2, B-5, B-6
  examples of 3-10
  UPDATE equivalent 1-12
ASCII option
  of DUMP command 3-19
  of MODIFY command 3-27/28
  of VERIFY command 3-51
Attributes 3-9, 3-21/23, 3-36, 3-44/46, 4-3, B-11
  see also ALTER

BEFORE parameter of MOVE command 3-30
BIND command 1-9, 1-11, 2-2/3
  examples of 1-9, 1-11, 2-3
BINDER
  commands 1-8, 1-12, 3-1/3, 3-6/52, 4-9
  error messages B-1/12
  error statistics 2-5
  output
    cross-reference listing 2-4, 2-8/9
    load map 2-4, 2-6/9, 3-11/12, 3-24/26, 3-40/43
    specifying listings 1-11, 2-6, 3-11/12, 3-24/26, 3-40/43
    statistics 2-4/5, 4-7
  primary function 1-1, 1-4, 4-15/16
  prompt 2-3
  relationship to CROSSREF 1-13
  relationship to INSPECT 1-13
  relationship to UPDATE 1-12

   requirements 2-1
      compatibility with compilers 2-1, B-2, B-8/9
   symbol tables 2-1, 3-50, 4-8, 4-14/15
   tables 2-1
   warning statistics 2-5
   with COMINT ASSIGN 2-2
   with COMINT PARAM SWAPVOL 2-2
Binding
   command-driven 1-4, 1-9, 2-2/3
   compile-time 1-2/3
   examples 1-9/10
   mixed-language 4-17/18
   object files 1-9
   preventing premature D-2
BINSERV 1-2/3
Block
   code
      see also Code block 4-2/3
   common 4-2, 4-4, 4-11
      consistency checking 3-40
      naming 4-4/5
      see also Data block
   data 4-3/4
      see also Data block
   definition of 1-5, 4-1
   importance as a unit of code or data 1-5, 4-1
   name 3-4
      assignment of 4-2, 4-4/5
      list 3-4
      range 3-4
   naming 1-4/5
   naming conventions 4-4/5
   order in target file 1-6
   own
      see Data block
   related to source language constructs 4-2
   relocating
      see MOVE command
   special
      see Data block
   types 4-2
BLOCK option of SELECT command 3-40, 4-16
BRIEF option of LIST command 3-24/25
BUILD command 1-8, 2-4, 3-1, 3-11/13, 3-34, 3-38, 3-40, 4-10,
               4-14/15, A-3, C-1, C-10
   errors 3-13, B-3
   examples of 1-9/10, 3-7/8, 3-13, 3-34, 3-39
   implying a SATISFY operation 4-11/12
   suppressing resolution with D-2
   UPDATE equivalent 1-12
Building a target file 3-11/13, 3-40/43, 4-1, 4-14/16
   error B-12

CALLABLE attribute 3-9, 4-3, B-2
CALLABLE parameter of ALTER command 3-9
CHANGE command 3-1, 3-14/15, C-2
  errors B-3
  examples of 3-15
Changing values of words 3-27/29
  verification of 3-29
CHECK parameter
  of BUILD command 3-12
  of RESELECT command 3-35
  of SELECT command 3-40/41, 4-16
  of SHOW command 3-47
CLEAR command 1-8, 3-1, 3-14, 3-16, 3-20, 4-9, B-3, C-2
  errors B-11
COBOL
  compiler 1-3, 2-1, 4-17
  extended data blocks 1-5, 2-8/9, 3-46, 4-5
  MAIN program unit 1-2, 4-5, 4-17/18, B-11
  PUCB B-2
  use of CLIBOBJ D-2
Code block 4-2/3
  attributes 3-9/10, 4-3
  equivalents 4-2
  load map
    examples of 2-6/7
    type of information included 2-6
  name 4-4/5
  renaming 3-32
  size 4-15, B-4
CODE parameter
  of ADD command 3-6/7, C-1
  of DELETE command 3-17, C-2
  of DUMP command 3-18/19, C-2
  of INFO command C-3
  of LIST command 3-24/25
  of MODIFY command 3-27/29, C-4
  of RENAME command 3-32, C-4
  of REPLACE command 3-33, C-4
  of VERIFY command 3-51, C-5
Code region 4-7
COMINT RUN command
  LIB option D-3
Command summary 3-1/3
Command syntax elements 3-4/5
Command-driven mode 1-4, 1-9, 2-2/3, D-2
  see also Mode, interactive
COMMENT command 1-8, 3-1, 3-16, A-2, C-2
  examples of 3-13, 3-19, 3-28
Common block
  see Block

COMPACT parameter
   effect on target file construction 4-15/16
   of BUILD command 3-12
   of RESELECT command 3-35
   of SELECT command 3-40/41
   of SHOW command 3-47
Compile-time binding 1-2/3, D-2
Compiler errors B-11
Consistency checks
   for common and parameters 3-40/41
Constructing a target file 3-11/13, 3-40/43, 4-1, 4-14/16
Control blocks 4-4, 4-17, B-1
Cross-reference lists 2-10/12, 3-24, 3-42
CROSSREF 1-13
Current file 1-7, 3-47
   clearing 3-16
   displaying contents of 3-18/19
   establishment of 3-20
   name of 3-47

Data
   external references
      see Data block
Data area size 3-44, 4-14, B-8
Data block 3-4
   added implicitly 2-12
   cross-reference list
      examples of 2-12
      type of information included 2-12
   equivalents 4-2, 4-4
   external references 4-3/4
   load map 2-8/9
   names 4-2, 4-4/5
   renaming 3-32
   types of 1-4, 4-2/4
      COMMON 4-2/4, 4-10, 4-13, 4-15
      OWN 3-40, 4-2/4, 4-10
      Special 4-2/5, 4-10
DATA parameter B-6
   of ADD command 3-6
   of BUILD command 3-11, 4-14
   of CHANGE command 3-14, C-2
   of DUMP command 3-18
   of LIST command 3-24/25
   of MODIFY command 3-27
   of RENAME command 3-32, C-4
   of REPLACE command 3-33
   of RESET command 3-36
   of SET command 3-44
   of SHOW command 3-48
   of VERIFY command 3-51, C-5

Data space
   format 4-8
   setting the size of 3-13
   size, setting of 3-44
DEBUG tool 4-8, 4-14, D-1
DECIMAL option
   of DUMP command 3-19
   of MODIFY command 3-27/28
   of VERIFY command 3-51
Default values 3-20, 3-35, 3-47, A-6
Defining a target file 1-5/6, 3-44/46
DELETE command 1-8, 3-1, 3-3, 3-17, B-2, C-2
   errors B-1
   reversing resolution with D-2
   UPDATE equivalent 1-12
DELETE parameter of ADD command 3-6, C-1
Deleting
   BINDER and INSPECT tables 3-50, B-9
   names from lists 3-16, 3-17
   procedures
      see ADD, DELETE commands
DETAIL parameter of INFO command 3-21, C-3
Disc file names A-1/2
Disc space
   BUILD fails for insufficient 3-13
   object file extents 4-5
   requirements 2-1
Display
   command syntax A-4
   contents of an object file 3-18/19
   current parameter values 3-21/23, 3-47/48
   include lists 3-5
   information about blocks & entry points 3-21/23
DUMP command 1-8, 3-2, 3-18/19, C-2
   errors B-2, B-5, B-7
   examples of 3-19
   UPDATE equivalent 1-12

Editing a command line A-3
Emptying a list 3-16
Ending a BINDER session A-3
Entry name 3-4
   list 3-4
   range 3-4
Entry point
   attributes of 3-9/10
   cross-reference list
      examples of 2-10/11
      type of information included 2-11
   load map 2-6/7
   names 4-2, 4-11/13
   postponing binding until run time D-2
ENV command 3-2, A-3, C-2
Error checking 2-5, 3-40/41, 4-16, B-9/10

Error messages B-1/10
Examples
   ADD command 3-7/8, 3-10, 3-13, 3-31, 3-34, 3-39
   binding separately compiled object files 1-9
   BUILD command 3-7/8, 3-13, 3-31, 3-34, 3-39
   CHANGE command 3-15
   COBOL MAIN program 4-18
   displays generated by INFO INCLUDE 3-5
   DUMP command 3-19
   FILE command 3-7/8, 3-19
   INFO command 3-23
   LIST command 3-26
   MODIFY command 3-28/29
   MOVE command 3-31
   multiple commands on one line 2-3
   object file statistics 2-5
   procedure replacement 1-9/10
   REPLACE command 3-13
   RESELECT command 3-35
   RESET command 3-37
   SATISFY command 3-39
   SELECT command 3-8, 3-43
   SET command 3-46
   SHOW command 3-48/49
EXIT command 2-3, 3-2, C-2
EXTENDSTACK parameter B-6
   of BUILD command 3-11, 4-14
   of RESET command 3-36
   of SET command 3-44, 4-14
   of SHOW command 3-48
Extents, disc 4-5
External references
   changes to unresolved list 3-17
   data
      must be resolved 4-3/4
   definition 1-1
   resolution 3-7
      deferred to run time 4-13
      using refer list 4-12/13
      using search list 4-12/13
   satisfaction 3-38/39, 3-43
   unresolved
      recorded in XEP 4-7
      resolved at run time D-1

Fatal error B-1, B-6, B-9, B-11
FC command 3-2, A-3, C-3
FILE command 1-7/8, 3-2, 3-16, 3-20, 3-24, B-2, B-7, C-3
   examples of 1-11, 3-7/8, 3-19
   UPDATE equivalent 1-12
File name expansion A-1/2
FILE parameter of SHOW command 3-47, C-5
FORTRAN compiler 1-3, 2-1, 4-17

FROM parameter
   of ADD command 3-6, C-1
   of LIST command 3-24/26, C-3
   of REPLACE command 3-33, C-4

G-relative address area B-10
Gap at 32K boundary 2-5, 4-7
   cause of 4-15
   filling automatically 3-41, 4-15/16
   filling by means of MOVE command 3-30
   warning message B-3, B-10
Global data
   in user libraries D-3
Global read-only arrays 4-7, 4-15, B-10
   in user libraries D-3

HELP command A-4, C-3
   errors B-8
HEX option
   of DUMP command 3-19
   of MODIFY command 3-27/28, C-4
   of VERIFY command 3-51

ICODE option B-5
   of DUMP command 3-19
Illegal syntax
   effect on BUILD command 3-13
Include lists 1-6, 3-6/7, 3-11, 3-16, 3-17, 3-21/23, 3-27, 3-30,
               3-38, 4-9/11, 4-14, B-2, B-4
   for code blocks 3-9, 3-27, 4-10/11
   for data blocks 3-27, 4-11
   for entry points 3-9/10, 3-30, 4-11
INCLUDE parameters
   of INFO command 3-21
INFO command 1-8, 3-2/3, 3-21/23, B-10, B-12
   examples of 3-5, 3-23
   UPDATE equivalent 1-12
Input stage 4-9
INSPECT parameter
   of BUILD command 3-11, 4-14/15
   of CHANGE command 3-14, C-2
   of RESET command 3-36
   of SET command 3-44/45, 4-14/15
   of SHOW command 3-48
INSPECT tables 3-50, 4-5, 4-8, B-9
INSPECT tool 1-13, 4-8, 4-14/15, B-9
Instruction
   see Code block
Interactive mode 2-3
INTERRUPT attribute 4-3

Language, mixed
   binding 4-17/18
LIBRARY option of SELECT command 3-40/41, 4-16
LIBRARY parameter
   of BUILD command 3-11, 4-14
   of CHANGE command 3-14, C-2
   of SET command 3-44, 4-14, C-5, D-3
   of SHOW command 3-48
Library, user
   see User library
LIKE parameter
   of ALTER command 3-9, C-1
   of RESET command 3-36
   of SET command 3-44
   of SHOW command 3-48
Limits
   code space D-1
   data space D-1
Linkage
   see Binding
LIST command 1-8, 1-11, 2-6, 2-10, 3-2, 3-24/26, B-2, C-3
   errors B-7
   examples of 1-11, 3-26
LIST parameter
   of BUILD command 3-12, C-2
   of RESELECT command 3-35
   of SATISFY command C-4
   of SELECT command 3-40/42
   of SHOW command 3-47
Listing
   of file contents
      see DUMP command
   of include lists
      see INFO command
Listings 2-4/12
   specifying 1-11
Lists
   include 1-6/8, 3-1/8, 3-11, 3-16, 3-21/23, 3-27, 3-30, 3-38,
            4-9/11, 4-14
   modify 1-6, 1-8, 3-1, 3-3, 3-16, 3-17, 3-27/29, 4-9, 4-13/14
   of names
      syntactic rules for 3-3/5
   omit 1-6, 3-7, 3-12, 3-16, 3-35, 3-42, 4-9/14
   refer 1-6, 3-12, 3-16, 3-35, 3-42, 4-12
   search 1-7, 3-12, 3-16, 3-35, 3-38, 3-43, 4-12
   unresolved reference 1-7/8, 3-5, 3-16, 3-17, 3-21/23, 3-38,
                         4-9/10, 4-12/14
Load maps
   alphabetical order 3-25, 3-42
   location order 3-25, 3-42
Load module
   see Code block

LOC option
   of LIST command 3-25
   of SELECT command 3-42
LOG command 3-2, A-5, C-3

MAIN attribute 3-9/10, 4-3, 4-16, B-3, B-7
   disallowed in user library entry points B-12, D-3
MAIN parameter of ALTER command 3-9/10
Maintenance of unresolved reference list 4-9
Mixed-language binding 4-17/18
Mode
   interactive 2-3
   noninteractive 2-3
MODIFY command 3-2, 3-17, 3-19, 3-27/29, 3-47, 4-4, 4-9, 4-13, C-4
   errors B-2, B-6, B-12
   examples of 3-28/29
   UPDATE equivalent 1-12
Modify list 1-6, 1-8, 3-3, 3-16, 3-17, 3-27/29, 4-9, 4-13/14
MODIFY parameter of SHOW command 3-47
Modifying object files 4-13
Module
   see Block, Code block, Data block
MOVE command 3-3, 3-30/31, 4-9/10, C-4
   errors B-6
   examples of 3-31
Moving code blocks 3-30/31

Name lists 3-3/4
Name ranges 3-3/4
Naming the target file 3-12
Noninteractive mode 2-3
Nonresident code blocks 3-10, 4-7, 4-15

OBEY command A-6, C-4
   errors A-6, B-8
Obey files A-6, B-6
OBJECT
   as a default target file name 3-12, B-3
Object file 4-1/8
   construction
      see Target file
   definition of 4-1
   format 4-5/8, D-2
      BINDER region 4-8
      code region 4-7
      code region (PEP) 4-7
      code region (user code) 4-7
      code region (XEP) 4-7
      data region 4-8
      header 4-5
      INSPECT region 4-8

size of 2-1, 2-5
affected by SYMBOLS option of SET 3-45, 4-14/15
reduced by COMPACT option of SELECT 3-12, 3-35, 3-40/41
reducing (see STRIP command)
structure 4-1/5
Object module
see Code block
OCTAL option
of DUMP command 3-19
of MODIFY command 3-27/28
of VERIFY command 3-51
Offset 4-5
Offset option
of MODIFY command 3-27/28
of VERIFY command 3-51
Omit list 1-6, 3-16, 3-35, 3-42, 4-9/14, B-3
OMIT parameter
of BUILD command 3-12, C-2
of RESELECT command 3-35, C-4
of SELECT command 3-40/42, 3-42, C-5, D-2
of SHOW command 3-47
Ordering code and data blocks
and object file format 4-7/8
overview 1-6/7
see also MOVE command
with ADD command 3-6
OUT command 3-2, A-7, C-4
Output stage 4-14/16
Overflow
avoiding code space D-2
Own block
see Data block

Page faults 3-30, 4-10, B-9
reducing 3-30
Pages 4-7, 4-8
required for code/data allocation 3-44, 4-7, 4-8
PAGES option
of BUILD command 3-11
of CHANGE command 3-14
of RESET command 3-36
of SET command 3-44, C-5
Parameter checking errors B-11
PARAMETER option of SELECT command 3-40/41, 4-16
Patching attribute values
of object files 3-14
Patching code or data
see MODIFY
PEP
list 1-11
in object file statistics 2-5
table
description 4-7

PEP parameter
   of BUILD command 3-11, 4-14, C-1
   of RESET command 3-36
   of SET command 3-44/45, 4-14, C-5
   of SHOW command 3-48
Printed output 2-4, 3-41/42, A-7
PRIV parameter
   of ALTER command 3-9/10, C-1
   warning message B-2
PRIVILEGED attribute of entry point 3-9/10, 4-3
Procedure replacement 1-9/10
Program
   definition of 4-1
Program file
   loading D-1
      see also System Desc. Manual (Section 4)
Prompt character for BINDER 2-3

Ranges of blocks and entry points 3-3/5, B-10
Read-only arrays
   global 4-7, 4-15, D-4
Recording session input and output A-5
Refer list 1-6, 3-16, 4-9, 4-12, 4-14
Refer pair 3-42, 4-12, B-6
REFER parameter
   of BUILD command 3-12
   of RESELECT command 3-35
   of SELECT command 3-40/42, 3-42, 4-12
   of SHOW command 3-47
Relocating code blocks 3-30
Removing
   BINDER and INSPECT tables 3-50
   names from lists 3-16, 3-17
RENAME command 3-2, 3-32, C-4
   disallowed with special data blocks 3-32
   errors B-10
Renaming code and data blocks 3-32
Repeating a command line A-3
REPLACE command 1-8, 3-2/3, 3-33/34, 4-9/11, C-4
   errors B-1, B-7, B-10
   examples of 3-34
Replacing code and data blocks 3-33
   see DELETE parameter of ADD
   see REFER option of SELECT command
RESELECT command 3-2, 3-35, 3-40, 4-12, C-4
   examples of 3-35
RESET command 3-2, 3-36/37, 3-44, C-4
   examples of 3-37
RESIDENT attribute 3-9/10, 4-3, 4-15
RESIDENT parameter of ALTER command 3-9/10, C-1
Resolving external references 1-1, 3-7, 3-12, 3-38, 3-43, 4-11/13,
                               4-15, B-3, B-5
Returning to original state 3-16, 3-35
Run-time binding D-1/2

SATISFY command 3-2, 3-38/39, 4-9/12
   errors B-3
   examples of 3-39
SATISFY parameter
   of BUILD command 1-12, 3-12, C-2
   of RESELECT command 3-35
   of SELECT command 3-40/43, 3-43, C-5
   of SHOW command 3-47
SAVEABEND parameter
   of BUILD command 3-11, 4-15
   of CHANGE command 3-14/15, C-2
   of RESET command 3-36
   of SET command 3-44/45, 4-15
   of SHOW command 3-48
Saving a file 3-45, 4-15
Search list 1-7
SEARCH parameter
   of BUILD command 3-12
   of RESELECT command 3-35
   of SATISFY command 3-38
   of SELECT command 3-40/43, 3-43, 4-12
   of SHOW command 3-47
SELECT command 1-8, 2-6, 3-3, 3-38, 3-40/43, 3-47, 4-7, 4-9/12, 4-15, C-5
   error B-11, B-12
   errors B-3, B-5, B-9/10
   examples of 3-43
   suppressing resolution with D-2
   UPDATE equivalent 1-12
SELECT parameter of SHOW command 3-47, C-5
Select-specification parameter 3-12, 3-35, 3-38, 3-47
SET command 1-8, 3-3, 3-14, 3-36, 3-44/46, 4-14, C-5, D-3
   examples of 3-27
   UPDATE equivalent 1-12
SET parameter
   of SHOW command 3-47/48, C-5
Set-specification parameter 3-11, 3-36, 3-44/45, 3-48, B-6
Setting attribute values 3-9/10, 3-44/46
Setting parameter values 3-40/43
SHOW command 1-8, 3-3, 3-29, 3-47/49, C-5
   examples of 3-29, 3-48/49
SOURCE parameter
   of LIST command 3-24
Special block
   see Data block
Specifying input file names 1-7, 3-6/7, 3-33, 3-40/43
Specifying output listings 1-11
STACK parameter
   error message B-6
   of BUILD command 3-11, 4-14
   of RESET command 3-36
   of SET command 3-44, 4-14
   of SHOW command 3-48

Stack space 3-44, 4-14, B-8
Stages of BINDER operation 4-9, 4-14
Storage space
  code D-1
STRIP command 1-8, 2-1, 3-3, 3-50, 4-8, C-5
  errors 3-50, B-11
Subvolume names A-1/2, A-8, B-7
SYMBOLS parameter
  of BUILD command 3-11, 4-14/15
  of RESET command 3-36
  of SET command 3-44/45, 4-14/15
  of SHOW command 3-48
Syntax conventions
  for name lists 3-3/4
Syntax summary C-1/5
SYSTEM command 3-3, A-8, C-5

Tables
  BINDER 2-1, 3-50, 4-7/8, 4-14/15
  INSPECT 3-50, 4-5, 4-8, B-9
  PEP 4-7, 4-14, B-6
  XEP 4-7, B-8
TAL
  compiler 1-3, 2-1
    LIBRARY directive D-3
  procedures
    in user libraries D-4
Target file 1-4, 3-29
  attributes 3-11, 3-44/45, 4-14/15
  characteristics 3-36, 3-44/45, 4-2, 4-14/15
  construction 3-11/13, 3-40/43, 4-1, 4-15/16
  definition of 1-4, 4-1
  naming 3-12
  ordering 1-6/7, 3-6, 4-15
Temporary work file errors B-11

UNRESOLVED DATA parameter
  of INFO command 3-21/23
UNRESOLVED ENTRY parameter
  of INFO command 3-21/23
Unresolved external reference
  resolved at run time D-1
Unresolved reference list 1-7/8, 3-16, 3-17, 3-21/23, 3-38, 4-9/10, 4-12/14, B-12
UPDATE command equivalents 1-12
UPDATE tool 1-12, 3-50, B-2
User library 3-14, 3-45, 4-14, 4-16, D-1/4
  error B-12
  file
    building of D-2
    restrictions D-3
  specifying D-3

VARIABLE attribute 4-3
   error B-12
VERIFY command 3-3, 3-51/52, C-5
   errors B-6, B-12
VOLUME command 3-3, A-8, C-5
Volume name A-1/2, A-8, B-7

Warnings 2-5, B-1
WORDS option
   of BUILD command 3-11
   of CHANGE command 3-14
   of SET command 3-44, C-5

XEP 2-5, 4-7, B-8
XREF option
   of LIST command 3-25
   of SELECT command 3-42

ZZBInnnn
   ultimate default name for target file 3-12, B-3

#GLOBAL
   errors B-10

YOUR COMMENTS PLEASE

Tandem NonStop™ & NonStop II™ Systems
BINDER™ User's Manual
82314 B00

Tandem welcomes your comments on the quality and usefulness of its publications. Does this publication serve your needs? If not, how could we improve it? If you have specific comments, please give the page numbers with your suggestions.

This comment sheet is not intended as an order form. Please order Tandem publications from your local Sales office.

FROM:

Name _____ Date _____

Company _____

Address _____

City/State _____ Zip _____

NO POSTAGE
NECESSARY
IF MAILED
IN THE
UNITED STATES

BUSINESS REPLY MAIL
FIRST CLASS PERMIT NO. 482 CUPERTINO, CA, U.S.A.

POSTAGE WILL BE PAID BY ADDRESSEE

Tandem Computers Incorporated
19333 Vallco Parkway
Cupertino, CA 95014-9990

Attn: Manager—Software Publications

TAPE

TAPE