Tandem NonStop™ & NonStop II™ Systems

PATHWAY™ Programming Manual

ABSTRACT: This manual describes the SCREEN COBOL language used in the PATHWAY Transaction Processing System by PATHWAY application programmers.

PRODUCT VERSION: PATHWAY E06

OPERATING SYSTEM VERSION: GUARDIAN A05 (NonStop II System)
GUARDIAN E06 (NonStop System)

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

Part No. 82059 E00

April 1983
Printed in U.S.A.

PRINTING HISTORY

82059	A00	December 1980	Original printing.
	B00	October 1981	Revised.
	C00	April 1982	Rewritten.
	D00	October 1982	Revised.
	E00	April 1983	Revised.

The following are trademarks or servicemarks of Tandem Computers Incorporated:

AXCESS	ENABLE	EXPAND	PERUSE	TMF
BINDER	ENCOMPASS	GUARDIAN	TANDEM	TRANSFER
CROSSREF	ENFORM	INSPECT	TAL	XRAY
DDL	ENSCRIBE	NonStop	TGAL	XREF
DYNABUS	ENVOY	NonStop II	THL	
EDIT	EXCHANGE	PATHWAY	TIL	

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 revision adds the following new information:

- the RECEIVE clause that causes SCREEN COBOL programs to accept data from devices other than the terminal keyboard (applies only to the T16-6530 terminal)
- for T16-6530 terminals, a SCREEN COBOL program can assign a RETURN KEY function through the SPECIAL-NAMES paragraph
- the RECONNECT MODEM statement that manages PATHWAY terminal connections across a dial-in switched line
- the SEND statement UNDER PATHWAY and AT SYSTEM phrases that enable communication between PATHWAY processes running in separate PATHWAY systems or on separate Tandem systems
- the STOP RUN statement that stops an executing program
- the SCREEN COBOL compiler command SYMBOLS that causes the SYMSERV process to build a program symbol table used by INSPECT
- the BINDER program is now used to store the user-defined checking and conversion procedures.

CONTENTS

FIGURES

TABLES

PREFACE

This manual is one of three manuals that describe the three major tasks associated with the PATHWAY Transaction Processing System. These manuals and tasks are:

- *PATHWAY Operating Manual* — Configuring the PATHWAY environment.
- *PATHWAY Programming Manual* — Programming the application that runs within the PATHWAY environment.
- *PATHWAY Programming Aids* — Using the utilities provided to create and modify screen definitions and to control application program object files.

This manual, which concerns PATHWAY application programming only, describes the SCREEN COBOL language. Section 1 presents an overview of the PATHWAY transaction processing environment. Section 2 describes the organization of a SCREEN COBOL source program and details the various rules of the language. Sections 3 through 6 describe the four sections that comprise a SCREEN COBOL program. Section 7 describes source program compilation. Section 8 illustrates a sample PATHWAY application.

The manual is for personnel who are responsible for developing a SCREEN COBOL requester program to define and control terminal displays in the PATHWAY environment. It is recommended that readers using this manual have a knowledge of Tandem system programming concepts.

The following publications contain information related to PATHWAY:

CROSSREF User's Manual
INSPECT Interactive Symbolic Debugger User's Guide
GUARDIAN Operating System Programming Manual
PATHWAY Operating Manual
PATHWAY Programming Aids
Transaction Monitoring Facility (TMF) Reference Manual
Transaction Monitoring Facility (TMF) System Management and Operations Guide for NonStop Systems
Transaction Monitoring Facility (TMF) System Management and Operations Guide for NonStop II Systems

SYNTAX CONVENTIONS IN THIS MANUAL

The following is a summary of the characters and symbols used in the syntax notation in this manual.

Notation	Meaning
UPPERCASE LETTERS	Uppercase letters represent keywords and reserved words.
lowercase letters	Lowercase letters represent variable entries to be supplied by the user.
Brackets [ ]	Brackets enclose optional syntax items. A vertically aligned group of items enclosed in brackets represents a list of selections from which one, or none, can be chosen.
Braces { }	Braces enclose required syntax items. A vertically aligned group of items enclosed in braces represents a list of selections from which exactly one must be chosen.
Ellipses ...	Ellipses immediately following a pair of brackets of a pair of braces indicate the enclosed syntax can be repeated any number of times.
Ellipses preceded by a comma ,...	Ellipses preceded by a comma and immediately following a pair of brackets or braces indicate that the enclosed syntax can be repeated a number of times and requires a comma separator before each repetition.
Punctuation	All punctuation and symbols other than those described above must be entered precisely as shown.

SECTION 1

PATHWAY ORGANIZATION

SCREEN COBOL is a principal component of PATHWAY, the ENCOMPASS product that simplifies the development and control of on-line transaction processing applications. A transaction is a basic unit of work defined by the organization that uses the computer system. Transactions typically originate at computer terminals and require access to a data base, either to search for information or to modify existing information. A terminal operator in the PATHWAY transaction processing environment enters queries and data from a terminal according to a specific screen format; the format is defined and controlled internally by the SCREEN COBOL application program.

A warehouse inventory system represents a typical transaction processing application. A terminal operator performs read transactions when querying the inventory data base to determine the quantity on hand of specific items. As new items are received and existing items are shipped, a terminal operator performs update transactions when modifying data in the data base to reflect current inventory.

A transaction to be processed within the PATHWAY system is entered from a terminal and passed to a requester process. The requester sends messages to a server process to perform functions on the data base. The server completes the requested data base function and replies to the requester. The requester can, in turn, send a reply back to the terminal as acknowledgement that the transaction has completed.

As a SCREEN COBOL programmer, you develop the requester program that defines the screen formats and controls for terminals operating within the PATHWAY environment.

SYSTEM COMPONENTS

The following are components of a PATHWAY system:

- PATHWAY Monitor process (PATHMON)—The central controlling process for PATHWAY.
- PATHCOM process—The command interface to PATHMON.
- SCREEN COBOL—The procedural language that is used to define and control terminal displays.
- Terminal Control Processes (TCPs)—The requesters that interpret SCREEN COBOL object code and send messages to server processes.

- Server Processes—The processes that implement data base oriented requests and send replies to the requesters.
- Transaction Monitoring Facility (TMF)—The data management product that is available for use in PATHWAY to maintain the consistency of a data base and provide the tools for data base recovery.
- PATHWAY Programming Aids—The utility program, PATHAID, that you use to create or modify screen definitions. The utility program, SCUP, you use to access and manipulate compiled programs in SCREEN COBOL object files.
- INSPECT—The interactive symbolic program debugging tool that you can use to examine and modify SCREEN COBOL programs.

PATHWAY Monitor Process

The PATHWAY Monitor (PATHMON) is the Tandem-supplied process that supervises and controls the PATHWAY system. This process controls the existence, the state, and the interrelations of the other processes and devices within PATHWAY. PATHMON assumes responsibility for the life of each process, from definition and start-up through operation and termination.

PATHMON maintains configuration and status information, starts and stops TCPs, starts and stops server processes, and grants links from the TCPs to the server processes.

PATHCOM Process

PATHCOM is the Tandem-supplied process that provides the command interface to PATHMON. PATHCOM executes the file of commands that are entered by the PATHWAY system designer to describe terminals, TCPs, and servers to the PATHWAY system. These commands describe which terminals are controlled by each TCP, describe the capacity of the PATHWAY system by indicating the maximum number of entities that can exist, indicate the starting and stopping of processes and terminals, and request the display of status and statistical information.

SCREEN COBOL

SCREEN COBOL is the language you use to write the requester program. Your program defines the display format, the application of editing checks and data conversion, the relationships between screen fields and data items, and the flow of messages to PATHWAY servers. You always design the SCREEN COBOL program as if to control a single terminal.

A SCREEN COBOL program performs three basic functions:

- Displays screens and data through execution of DISPLAY statements.
- Allows data to be entered from the terminal through execution of an ACCEPT statement.
- Sends messages to a server process through execution of a SEND statement.

These functions are illustrated in Figure 1-1.

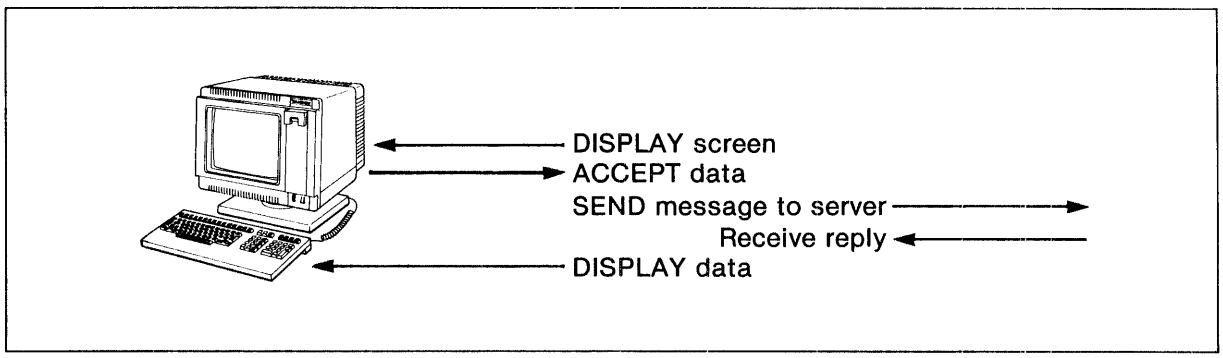


Figure 1-1. SCREEN COBOL Functions

Terminal Control Process

A Terminal Control Process (TCP) is a Tandem-supplied program that interprets the code generated by the SCREEN COBOL compiler for multiple SCREEN COBOL programs. A TCP assumes responsibility for the physical terminal I/O operations, performs field validation based on edit patterns in the SCREEN COBOL application program, maintains separate data areas and control information for each terminal under its control, handles the conversion of data between external and internal representations, and sends messages to sever processes on behalf of the SCREEN COBOL application.

Server Process

A server process is a program written in COBOL, FORTRAN, MUMPS, or the Tandem Transaction Application Language (TAL) to implement the data base oriented requests and replies in the form of transaction messages. These messages are generated by the SCREEN COBOL application and sent by a TCP. You design the server to receive and interpret the requests, perform data base I/O functions according to these requests, and send appropriate replies back to the TCP.

A server is configured to be a member of a particular server class. The server class itself has specific characteristics that are defined within the PATHWAY configuration. Individual servers within a server class are simply copies of a single program; PATHCOM creates new servers from a single server program according to configuration criteria.

Transaction Monitoring Facility

The Transaction Monitoring Facility (TMF) is a data management product that maintains the consistency of a data base and provides the tools for data base recovery. TMF requires that monitored data files be flagged for auditing. TMF audits a file by maintaining before and after images of changes to these files. These images provide the basis for transaction backout, which cancels the effects of a partially completed transaction, and data base rollforward, which restores a data base to a consistent state after a catastrophic failure.

PATHWAY systems that use TMF must have server classes configured to operate on audited files. Servers that are configured as TMF servers can read, lock, and change records in audited files. Servers that are not configured as TMF servers can only read audited files.

Terminal program units that communicate with TMF servers via the SCREEN COBOL SEND verb must be configured in PATHCOM for TMF. TMF is invoked by execution of a SCREEN COBOL BEGIN-TRANSACTION verb, at which time the terminal program enters what is called transaction mode. The terminal program remains in transaction mode until execution of a SCREEN COBOL END-TRANSACTION (or ABORT-TRANSACTION) verb. These verbs start and end a sequence of operations that are treated as a single transaction by TMF. An additional verb is available to restart a transaction. When transaction mode begins, TMF assigns the transaction a unique identifier called a TRANSID. When concurrent terminal programs are in transaction mode, TMF distinguishes transactions by TRANSIDs.

For information about TMF, refer to the *Introduction to Transaction Monitoring Facility (TMF)* and the *Transaction Monitoring Facility (TMF) Users Guide.*

PATHWAY Programming Aids

PATHWAY programming aids include the PATHAID screen builder and the SCREEN COBOL Utility Program (SCUP). The PATHAID screen builder allows you to create and modify screen definitions for use in PATHWAY applications. SCUP allows you to manipulate SCREEN COBOL object files; you can issue commands to display information about programs, change the accessibility of programs, copy programs from one object file to another, delete programs, and reclaim file space by compressing object files.

For information about PATHWAY programming aids, refer to the *PATHWAY Programming Aids* publication.

CROSSREF

CROSSREF is a program development tool that produces a list of program references used by SCREEN COBOL programmers to aid in debugging programs. This list contains screen names, paragraph names, data variables, and other program identifiers. In addition, this list describes where and how the identifiers are used throughout the program. The SCREEN COBOL compiler produces the CROSSREF listing and writes it to the output file for the compile.

To obtain a CROSSREF listing, compile your SCREEN COBOL program with the CROSSREF compiler command. The CROSSREF compiler command is described in the "Compilation" section of this manual.

For a complete description of how to use CROSSREF, refer to the *CROSSREF Users Manual.*

INSPECT

INSPECT is an interactive symbolic program debugging tool that you can use to examine and modify SCREEN COBOL programs. INSPECT runs as a separate process that communicates through the TCP with the SCREEN COBOL program running on a PATHWAY terminal. By issuing commands to INSPECT you can control and modify an executing program.

INSPECT uses a symbol table file for the SCREEN COBOL program. To generate a symbol table file when the program is compiled, you must specify the SYMBOLS compiler command either in the program source code or in the compiler run command. The SYMBOLS compiler command is described in the "Compilation" section of this manual.

Before you can use INSPECT, the PATHWAY system must be configured for communication with INSPECT. The PATHCOM commands that let the TCP and the terminals communicate with INSPECT are described in the *PATHWAY Operating Manual.*

For a complete description of how to use INSPECT, refer to the *INSPECT Interactive Symbolic Debugger User's Guide.*

APPLICATION CONFIGURATION

A PATHWAY system appropriate for executing an application is configured through PATHCOM commands. These commands are issued by the PATHWAY application designer to define the capacity and the environment of the PATHWAY system and describe the characteristics and start-up information of the PATHWAY processes and devices. PATHMON maintains this information in a file called PATHCTL.

PATHMON builds the PATHCTL file the first time PATHWAY is configured. PATHMON always uses this file when starting a PATHWAY system after a normal shutdown or a total system failure.

The structure of a PATHWAY system is illustrated in Figure 1-2. The various components and files that comprise the processing environment are briefly described as follows:

- Commands are input to PATHCOM from a terminal, obey file, or another process to initiate PATHMON and establish the PATHWAY configuration.
- PATHMON controls the interrelations of the processes and devices. PATHMON also reports errors and changes in status to a log terminal or log file according to logging commands submitted to PATHCOM.
- The TCPs perform application operations according to the SCREEN COBOL object program, provide general control of the assigned terminals, and route messages to available servers.
- The servers access and update the data base files and reply to messages.

When TMF is configured and terminals are operating in transaction mode, data files auditing is performed.

COMMUNICATION BETWEEN PROCESSES

TCPs and servers communicate with each other by exchanging transaction messages and transaction replies through an interprocess file. Messages and server classes are referenced in the SCREEN COBOL SEND statement, which is executed by the TCP.

When a SCREEN COBOL program communicates with a server class in a different PATHWAY system, the program becomes location sensitive. That is, the SCREEN COBOL SEND statement must indicate the Tandem system on which the external server is running and the name of the external PATHMON (the PATHMON running in a different PATHWAY system from the requesting SCREEN COBOL program) controlling the server class.

Specifying the system name and the PATHMON name makes communication possible among multiple PATHWAY systems on the same Tandem system and on different Tandem systems.

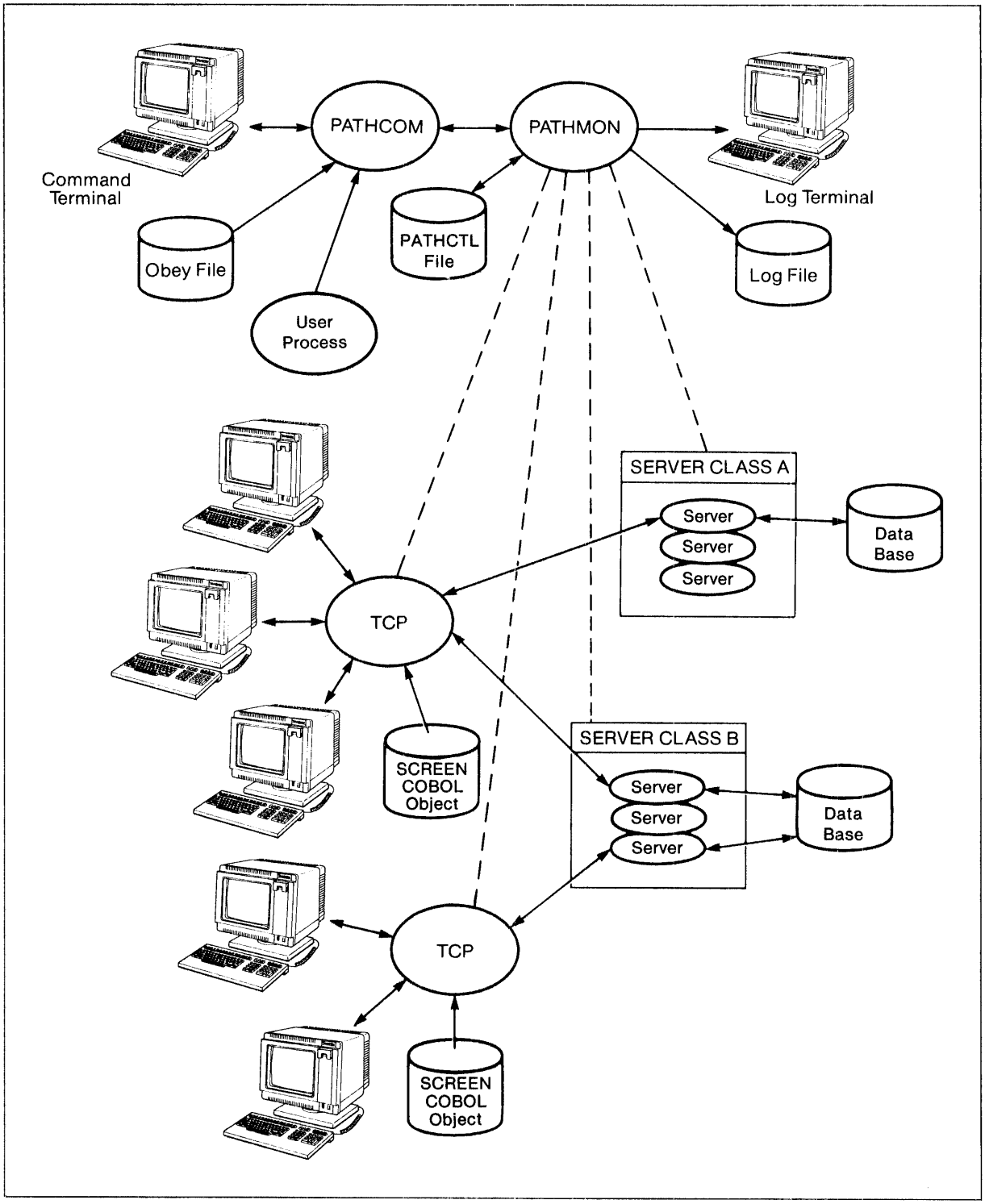


Figure 1-2. PATHWAY System Structure

Transaction Messages

A transaction message is sent by a SCREEN COBOL program to a server. This message consists of data supplied by the SCREEN COBOL SEND statement. The data in the message is a list of SCREEN COBOL data items. Each data item begins immediately after the last byte of the preceding data item. This list can include variable length data items. It should be noted, however, that a message containing a variable length data item cannot be easily decomposed by a server written in COBOL; the only exception is when the variable length data item is the last item of the message.

If a server is to process more than one type of message, a data item of the message should contain a field that identifies the type of transaction unless the content of the data itself determines the transaction type.

Transaction Replies

A transaction reply is sent by the server to the TCP. This reply consists of a two-byte binary integer reply code value plus the data. Upon receipt of this reply, the TCP compares the reply code value to the list of reply code values given in the SCREEN COBOL SEND statement. The TCP determines which reply was received and, consequently, determines the structure of the data. (The SEND statement declares the data structure that is associated with each valid reply code value.) The TCP then copies the reply into the SCREEN COBOL program.

DEVELOPING THE APPLICATION

Transactions are processed with requesters and servers. The development of an application involves the design of the entire application and the partitioning of work load between the requester and the servers. The functions of the requester should be kept as simple as possible. SCREEN COBOL requester program development is illustrated in Figures 1-3, 1-4, and 1-5.

PATHAID, the PATHWAY screen builder illustrated in Figure 1-3, can be used to design PATHWAY screens by first laying out a picture of the screen on your terminal. Data field display attributes can be assigned after you design the screen. PATHAID then generates the associated screen description source code for the screen layout from the terminal. You can edit the screen source code and copy the screen definitions directly into a SCREEN COBOL program unit or store the screen code in a screen library file.

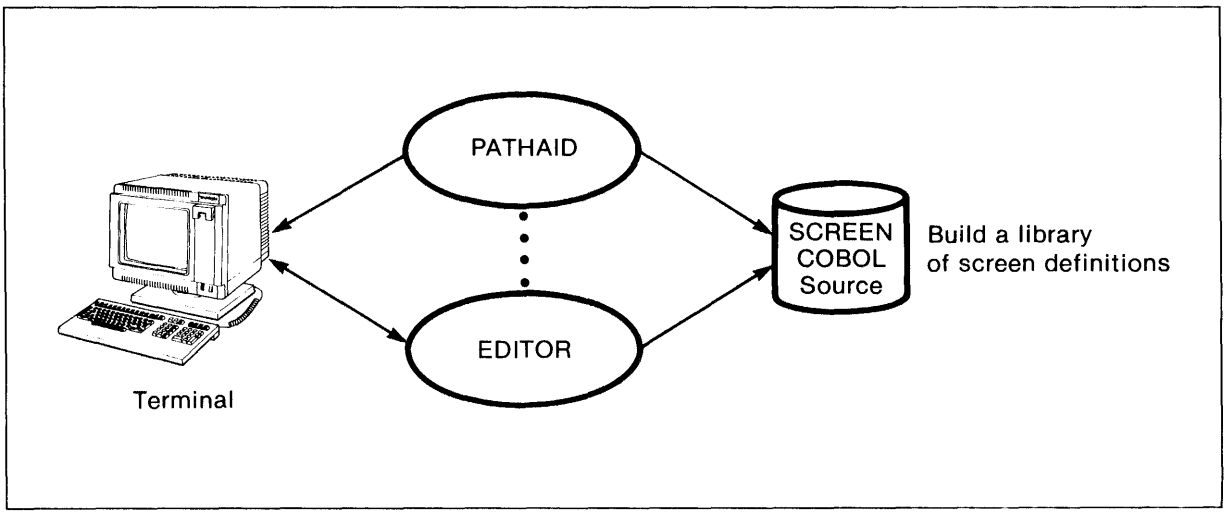


Figure 1-3. Developing Screen Definitions with PATHAID

Figure 1-4 illustrates general SCREEN COBOL program development. You create an edit file in which to build the SCREEN COBOL source code. This file can contain the source code for an entire SCREEN COBOL program or can contain COPY statements that insert other SCREEN COBOL program units when the source code is compiled.

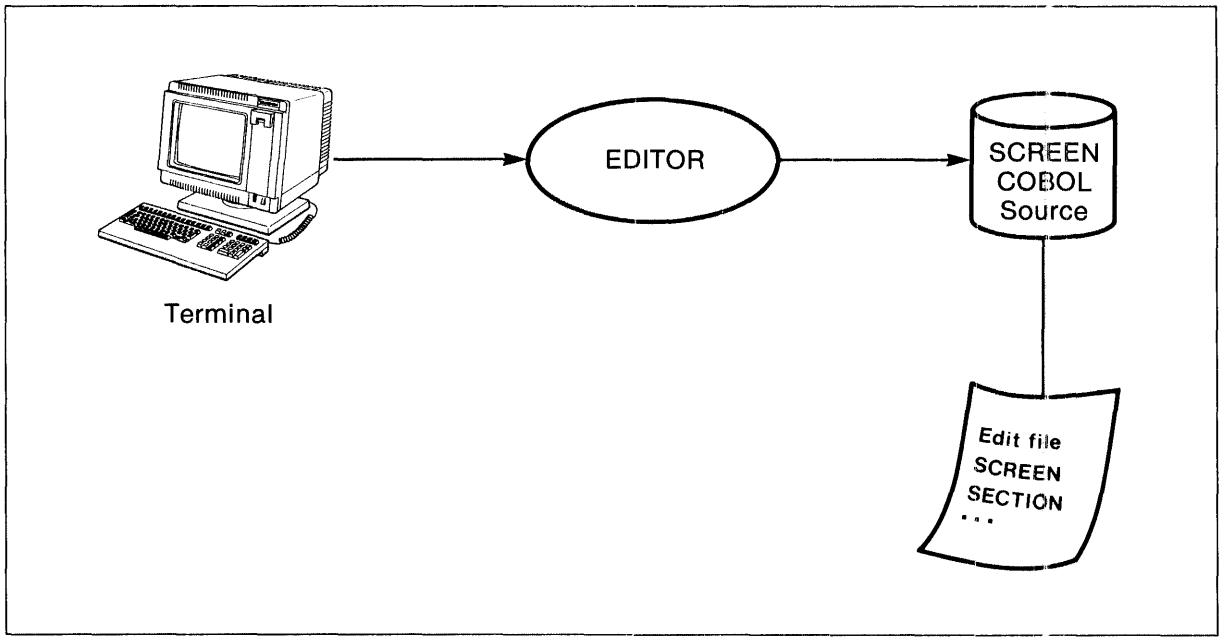


Figure 1-4. Building SCREEN COBOL Program Units with EDIT

Figure 1-5 illustrates the SCREEN COBOL compiler output. The SCOBOL run command invokes the SCREEN COBOL compiler which produces the object code according to the compiler commands specified in the run command and specified in the source code. There are three processes associated with the SCREEN COBOL compiler (SCOBOL, SCOBOL2, and SYMSERV). The SCOBOL and SCOBOL2 processes produce two object files—a director file and a code file. The TCP interprets the object code produced by the compiler when the program is executed.

If you specify the CROSSREF compiler command, the compiler produces a cross-reference listing for your program and includes the listing in the output file.

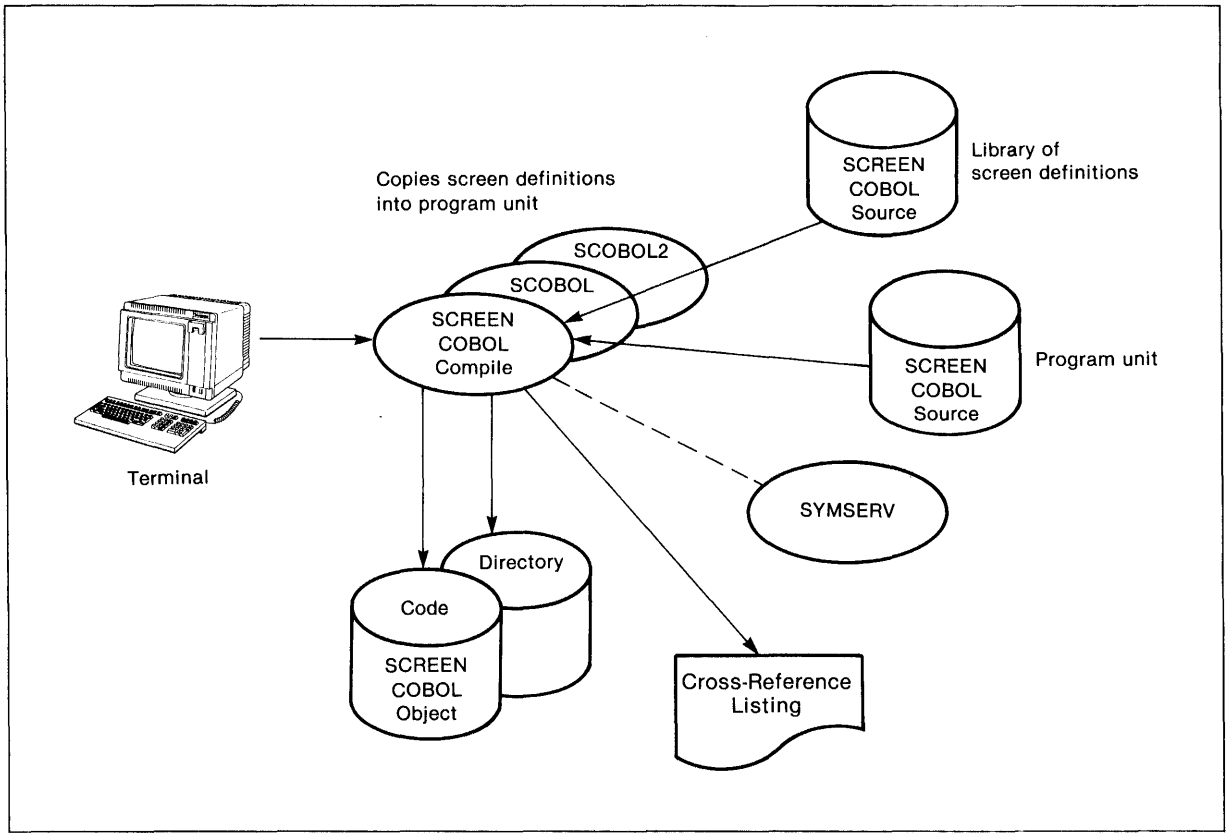


Figure 1-5. Producing SCREEN COBOL Object Files

The third process associated with the SCREEN COBOL compiler is SYMSERV. This process can be used to produce the program symbol table that is used by the INSPECT utility during program debugging. To obtain a symbol table file, you must specify the SYMBOLS compiler command. Figure 1-6 illustrates the output from the SCREEN COBOL compiler including a symbol table file.

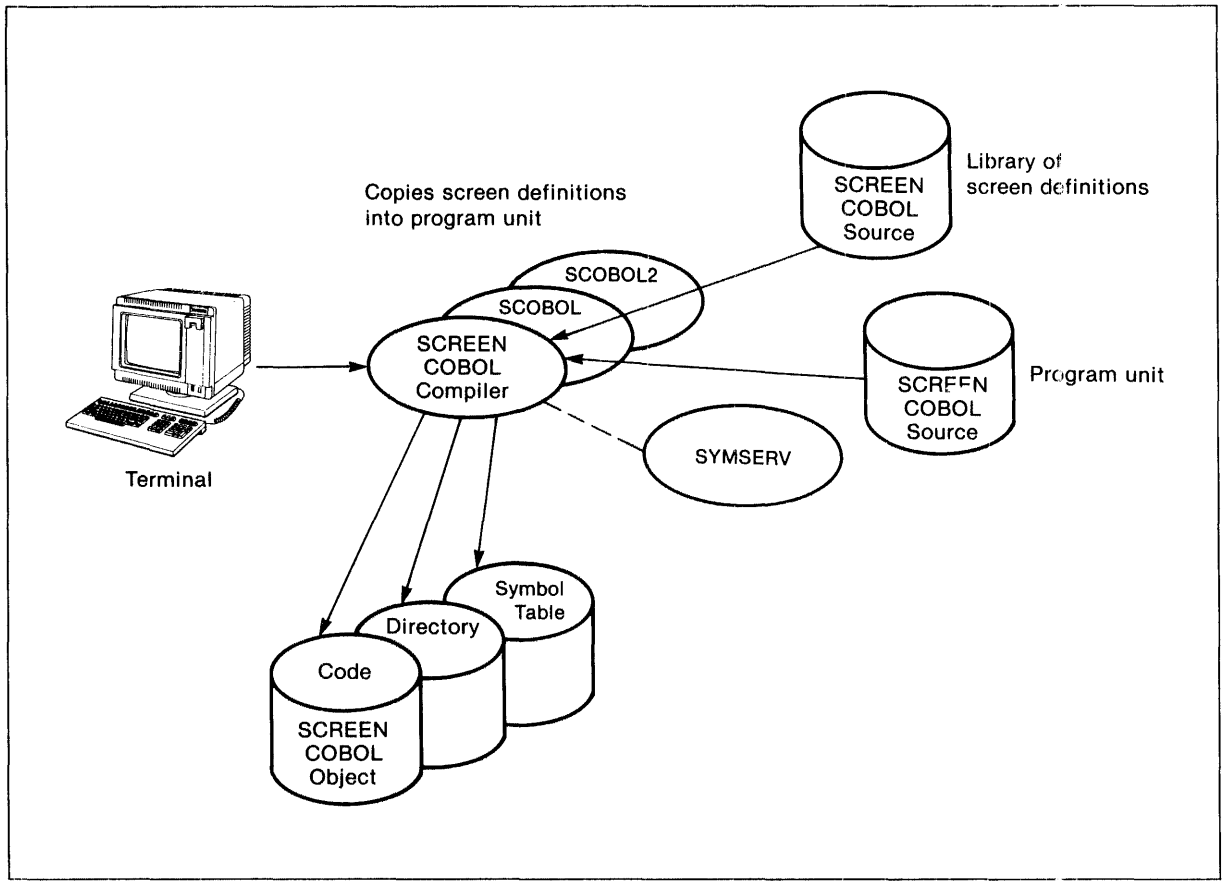


Figure 1-6. SCREEN COBOL Object Files—Including a Symbol Table

Each time a SCREEN COBOL program is successfully compiled, the new version of the object file is added to the previously compiled versions. You can use the SCREEN COBOL Utility Program (SCUP), illustrated in Figure 1-7, to manage your SCREEN COBOL object files in the following ways:

- to display information about the program units in the SCREEN COBOL object files
- to delete previously compiled program versions from the object files
- to build a new SCREEN COBOL object file by copying programs from one SCREEN COBOL object file to another
- to reclaim file space by compressing object files.

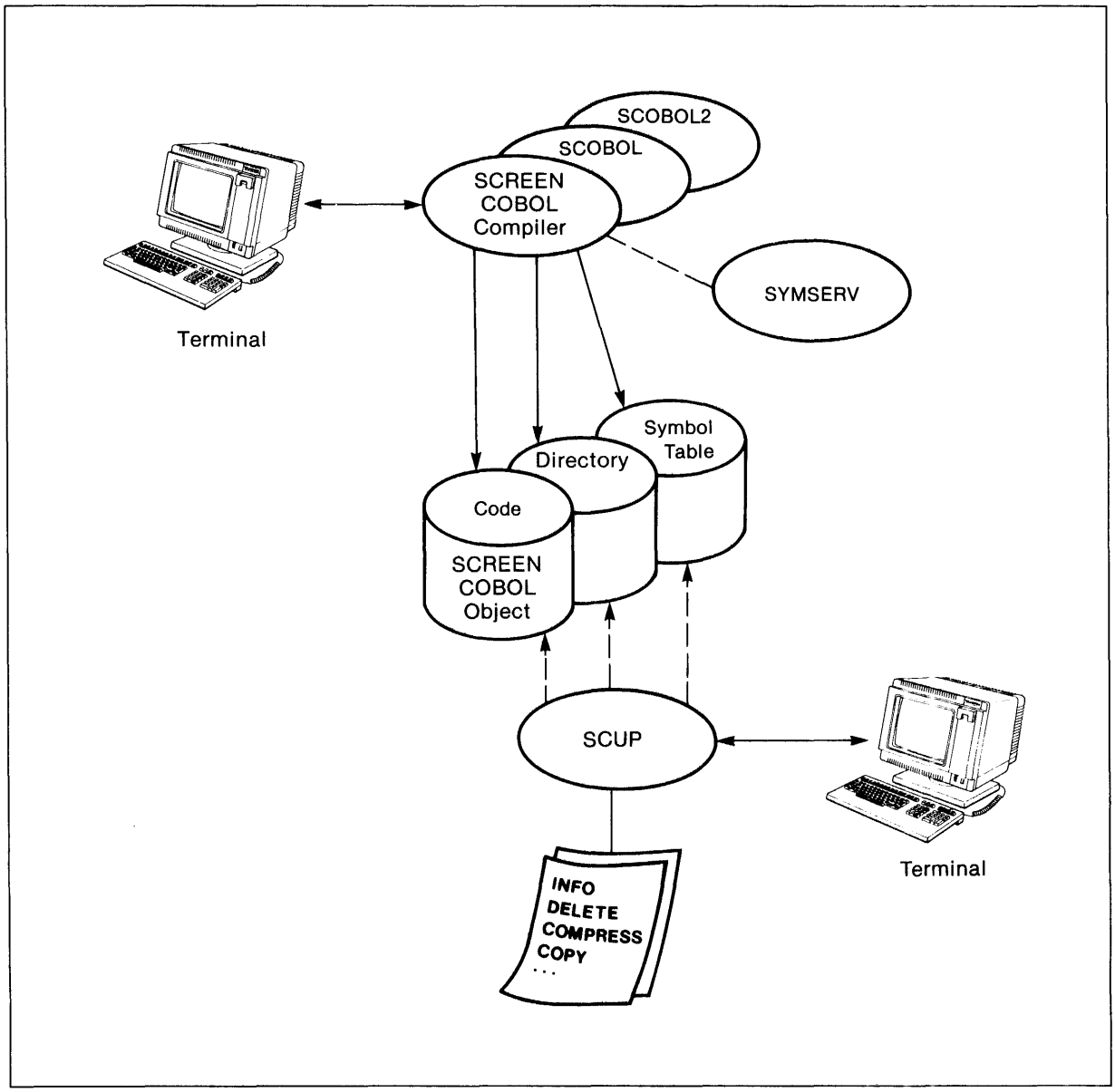


Figure 1-7. Managing SCREEN COBOL Object Files with SCUP

General rules concerning SCREEN COBOL requester program development are given in the following list:

- Design simple screens.
- Keep the operator informed of task completion, errors, the next step, and what the system is doing at all times.
- Design screens to display initial values and thus reduce keying of data. If no initial value is declared for a screen field, a default value can be established by moving a value into the data name associated with the field; this default value will be changed only if the operator enters data into the field or the program moves another value into the field.
- Protect crucial screen fields; for example, protect primary key.
- Reduce errors on crucial screen fields by using check digits. Check digit processing can be performed by the SCREEN COBOL program or by user conversion procedures as described in Appendix D.
- Keep context information in the requester and never in the server. Context is any information that is required by a process to continue operating in a previously existing environment.
- Use a modular program design for ease of maintenance.

Refer to the *PATHWAY Operating Manual* for information about configuring a PATHWAY application.

SCREEN COBOL Programming Techniques to Reduce Terminal Context

Terminal context in the PATHWAY environment is composed of data that must be maintained for each active terminal (that is, entered data, data base records, file position data, or program data) and data that is required by the TCP to execute the program units. PATHWAY provides the TCP to manage this data so that SCREEN COBOL program units can be written for a single terminal and servers can be written context free.

The following SCREEN COBOL programming techniques can be used to reduce terminal context:

- Whenever possible, limit program functions to the following:

  Accept data from the screen
  Send information to a server
  Receive data from a server
  Display data on the screen
  Call another SCREEN COBOL program

- Evaluate whether a function should be performed in a SCREEN COBOL program or in a server. For example, putting a large table of user logons in a server reduces the amount of context in the SCREEN COBOL program. The trade-off is between context in the SCREEN COBOL program and an I/O to the server.
- Accept data into the server request message and display data from the server reply message. When data is accepted from the screen into one area of working storage then moved to the request message, context data is wasted and unnecessary move instructions are used.

- Whenever possible, pass parameters instead of defining the same record definitions or data items in multiple SCREEN COBOL programs. The parameters appear in the Linkage Section of the called program. A Linkage Section in a SCREEN COBOL program unit contains pointers back to the calling program data area; the data is not duplicated.

- When appropriate, use the REDEFINES clause. Sometimes it is possible to redefine requests or replies that are not used simultaneously. In the case where one send can yield multiple replies, it might be possible to redefine the replies so they use the same storage area. Make sure the data description that matches the current reply code is the data description that is used. Use the DDL COBLEVEL command to set the correct level number in the record description.

- Use a shared request/reply buffer. In many SCREEN COBOL applications a one screen per program module structure is suggested. This sometimes means there will be at least one send per program module and possibly multiple record definitions for requests and replies. If the operator follows the program modules down the tree structure, the separate request/reply buffers consume additional context.

  By changing the design so that a reply area is passed from the first program unit in the Linkage Section to each program unit down the tree structure, the context space is reused as a global buffer. This change in design could cause some additional programming to save data that is needed when returning to certain modules. If the amount of data needed to be saved is equal to the buffer, nothing is gained by the use of a shared buffer. If the amount of data is less than the buffer, context is saved. This approach, however, more closely couples the program units, which might not be desirable.

SECTION 2

SCREEN COBOL SOURCE PROGRAM

The SCREEN COBOL procedural language is used to define and control the terminal displays. The syntax of the language enables you to:

- define the characteristics of the display screen
- indicate how the data is to be converted and how editing checks are to be applied to the data
- specify transaction messages to be sent to the server process
- control how input or output is to be accepted and displayed on the terminal screen.

SCREEN COBOL is available for use on the T16-6510, T16-6520, T16-6530, the IBM-3270, and those devices operating as conversational mode terminals as recognized by the GUARDIAN File System.

A SCREEN COBOL program is always designed as if to control a single terminal. The Terminal Control Process (TCP) that interprets the object code generated by the SCREEN COBOL compiler, however, can perform multiple executions of the same code for each terminal under its control.

PROGRAM OPERATING MODES

Generally, a SCREEN COBOL program displays formatted information, receives data entered from a terminal, and performs some action based on the data. SCREEN COBOL enables you to write programs that perform these operations in either of two modes: block mode (full screen accept and display operations) or conversational mode (line by line accept operations). To support both of these modes, some of the SCREEN COBOL statements and clauses act differently in block mode from conversational mode. These differences are summarized below and described in detail throughout the following sections.

Block Mode Program

To execute in block mode, a SCREEN COBOL program must run on a block mode terminal. The screen definitions for any SCREEN COBOL program are restricted by the characteristics of the specific type of terminal on which your program runs.

A SCREEN COBOL program running in block mode performs as follows:

- displays a full screen of information on the terminal
- accepts data entered from the terminal one screen at a time
- recognizes a specific terminal type
- recognizes function keys and associates each with a particular function (for example, pressing the F1 function key might might be associated with exiting from a screen).

Conversational Mode Program

A SCREEN COBOL program written for conversational mode operation can run on either a block mode terminal or a conversational mode terminal. Once a program is specified as conversational, that program performs according to the restrictions for a conversational terminal regardless of the type of terminal on which the program runs.

A SCREEN COBOL program running in conversational mode performs as follows:

- displays information on the terminal during an ACCEPT statement, one line at a time
- accepts data entered from the terminal one line at a time
- responds to a set of input control characters when the terminal is enabled to accept data
- recognizes only keyboard characters, carriage return, and line feed (not function keys)
- restricts the display field attributes to bell and hidden.

PROGRAM ORGANIZATION

A SCREEN COBOL program is organized into four divisions that must be written in the following order:

Identification Division
Environment Division
Data Division
Procedure Division

The Identification Division identifies the program. Comments such as the name of the programmer, the date the program was written, and a description of the program can be declared in this division.

The Environment Division specifies the program execution environment. Display error attributes, processing options, computer equipment, and terminal equipment can be described in this section.

The Data Division defines the program data structures in terms of their formats and usage. The Screen Section that appears in this division describes the structure of the data moving to and from the terminal.

The Procedure Division specifies the processing steps of the program.

LANGUAGE ELEMENTS

The SCREEN COBOL language elements fall into one of two categories: character strings and separators. Character strings are strings of contiguous characters. Separators are characters that separate one character string from another character string.

The language elements that comprise the SCREEN COBOL source program are described in the following paragraphs.

Character Set

The SCREEN COBOL character set is a subset of the ASCII character set and consists of 52 characters. These characters are listed in Table 2-1.

Table 2-1. SCREEN COBOL Character Set

0-9	Digits	,	Comma
A-Z	Letters	;	Semicolon
	Space (blank)	.	Period (decimal point)
+	Plus sign	"	Quotation mark
–	Minus sign (hyphen)	(	Left parenthesis
*	Asterisk	)	Right parenthesis
/	Stroke (slash)	>	Greater than
=	Equal sign	<	Less than
$	Currency sign	@	Commercial at

The following definitions apply to the SCREEN COBOL character set:

- Alphabetic characters include letters A through Z and space.
- Numeric characters include digits 0 through 9.
- Special characters include all characters except letters A through Z, space, and digits 0 through 9.
- Alphanumeric characters include any character in the character set.

The full ASCII character set can be used in comments and literals.

Editing Characters

Editing characters are symbols that can be used in PICTURE clauses to format screen data. Editing characters are listed in Table 2-2.

Table 2-2. Editing Characters

A	Alphabetic or space	–	Minus
B	Space insertion	CR	Credit
P	Decimal position (scaled)	DB	Debit
V	Decimal position (fixed)	*	Check protect
X	ASCII character	$	Currency symbol
Z	Zero suppress	,	Comma (decimal point)
0	Zero	.	Period (decimal point)
9	Numeric digit	/	Stroke (right slash)
+	Plus		

Punctuation Characters

Punctuation characters are used to separate words, sentences, or special clauses, and to group arithmetic relationships. Punctuation characters are listed in Table 2-3.

Table 2-3. Punctuation Characters

,	Comma
;	Semicolon
.	Period
"	Quotation mark
(	Left parenthesis
)	Right parenthesis
	Space (blank)
=	Equal sign

Separators

Separators are strings of one or more punctuation characters; they can have leading or trailing blanks. Separators are listed and defined in Table 2-4.

Table 2-4. Separators

space	A space separates language elements.
, ; .	A comma, semicolon, or period immediately followed by a space is a separator. A period can appear as a separator only when it terminates headers, entries, and sentences as defined by the syntax. A comma or semicolon is treated as a space when used as a separator.
( )	Right and left parentheses enclose certain parts of character strings. Although they must appear in balanced pairs, each is considered a separator.
"	Quotation marks are used to enclose nonnumeric literals. The characters appear in balanced pairs except when the literal is continued across a line. The first quotation mark must be preceded by a space, and the second one must be followed by a separator other than another quotation mark.

Some character strings include punctuation characters, in which case those characters do not act as separators. Any character in the ASCII character set can appear in a nonnumeric literal, provided the character does not have special meaning to a hardware device.

SCREEN COBOL Words

A SCREEN COBOL word is a character string that forms a reserved word, user-defined word, or system name. A word can have a maximum of 30 characters.

RESERVED WORDS. A reserved word has special meaning for the compiler. A reserved word cannot be used as a data item name or a system name. Reserved words are any of the following:

Keywords
Special registers
Figurative constants

Reserved words must be spelled correctly and can be used only as specified in syntax.

USER-DEFINED WORDS. A user-defined word can consist of any of the following characters:

Letters A through Z
Digits 0 through 9
The hyphen character (-)

A user-defined word must have at least one alphabetic or numeric character, must not begin or end with a hyphen, and must not contain embedded spaces. User-defined words are used for the following types of items:

Procedure name	Program name
Data name	Library name
Mnemonic name	Text name
Condition name	

SYSTEM NAMES. A system name is a SCREEN COBOL word that identifies part of the Tandem operating environment. System names are defined for equipment and operating system access. Use of each system name is restricted to a specific category, such as terminal function key or display attribute.

Literals

A literal is a character string whose value is implied either by a set of characters or by a reserved word that represents a figurative constant. A literal is numeric or nonnumeric.

NUMERIC LITERALS. A numeric literal is one or more digits (0-9), a plus or minus sign, and an optional decimal point. The value of the literal is the value of the digits. The following rules apply to numeric literals:

- A numeric literal can have a maximum of 18 digits.
- One sign character is allowed and must be the first character. The absence of a sign character indicates the literal is a positive number.
- A numeric literal can have one decimal point, which can appear anywhere within the literal except as the last character. The absence of a decimal point indicates the literal is an integer.

The following examples illustrate numeric literals:

Integer Numeric Literals	NonInteger Numeric Literals
+601	+601.1
−234116	89.6
0	0.0051
15	−.1

NONNUMERIC LITERALS. A nonnumeric literal is any ASCII character string enclosed in quotation marks. The value of the literal is the string of characters between the quotation marks. The following rules apply to nonnumeric literals:

- Nonnumeric literals can have a maximum value of 120 characters, not including the surrounding quotation marks.
- If a quotation mark is part of the literal, it must be represented in the string as two contiguous quotation marks. The additional quotation mark is not included in the character count.

The following example illustrates a nonnumeric literal:

"THIS IS A NONNUMERIC LITERAL"

"12345 THIS IS A NONNUMERIC LITERAL ALSO"

The following example illustrates a nonnumeric literal with an embedded quotation mark:

"A " " IS PART OF THIS NONNUMERIC LITERAL"

FIGURATIVE CONSTANTS. A figurative constant is a constant that has been prenamed and predefined by the SCREEN COBOL compiler so that it can be written in the source program without having to be defined in the Data Division. Figurative constants do not require quotation marks.

Figurative constants are listed and defined in Table 2-5; singular and plural forms are equivalent in meaning:

Table 2-5. Figurative Constants

ZERO ZEROS ZEROES	Depending on the context, represents the numeric value 0 or a string of one or more of the character 0.
SPACE SPACES	Represents one or more ASCII space characters (blanks).
HIGH-VALUE HIGH-VALUES	Represents one or more of the characters that have the highest position in the ASCII character set.
LOW-VALUE LOW-VALUES	Represents one or more of the characters that have the lowest position in the ASCII character set.
QUOTE QUOTES	Represents one or more quotation mark characters. Neither of these words can be used in place of quotation mark characters around a nonnumeric literal string.
ALL literal	Repeats the value of literal. Literal must be a nonnumeric literal or figurative constant other than ALL literal. When a figurative constant is used, the word ALL is redundant and is used only for readability.

The following rules apply to figurative constants:

- When a figurative constant represents multiple characters, the length of the string is determined by the compiler.
- A figurative constant can be used wherever a literal appears in a format; when the literal must be numeric, only ZERO, ZEROS, or ZEROES are permitted.
- When a figurative constant is moved or compared to another data item, the figurative constant is repeated on the right until its size is equal to the size of the data item. This happens independently of a JUSTIFIED clause for the data item.

REFERENCE FORMAT

A SCREEN COBOL source program can be written in Tandem standard or ANSI standard reference format. The Tandem standard reference format has no sequence number field (columns 1-6), has no identification field (columns 73-80), and is restricted to lines of up to 132 characters.

Although the SCREEN COBOL compiler assumes Tandem format, a SCREEN COBOL program can be written completely in either format or in a combination of both. Refer to the source text options in Section 7 for information regarding format specification.

Tandem Standard Reference Format

Lines in Tandem standard reference format are not fixed length; they can have up to 132 characters. Lines longer than 132 characters are truncated; trailing blanks are ignored. For each line, Margin R is set to follow the last nonblank character in the line, regardless of the Margin R location in any previous line.

Trailing blanks from a previous line and initial blanks on a continuation line are ignored.

The Tandem standard reference format is illustrated in Figure 2-1.

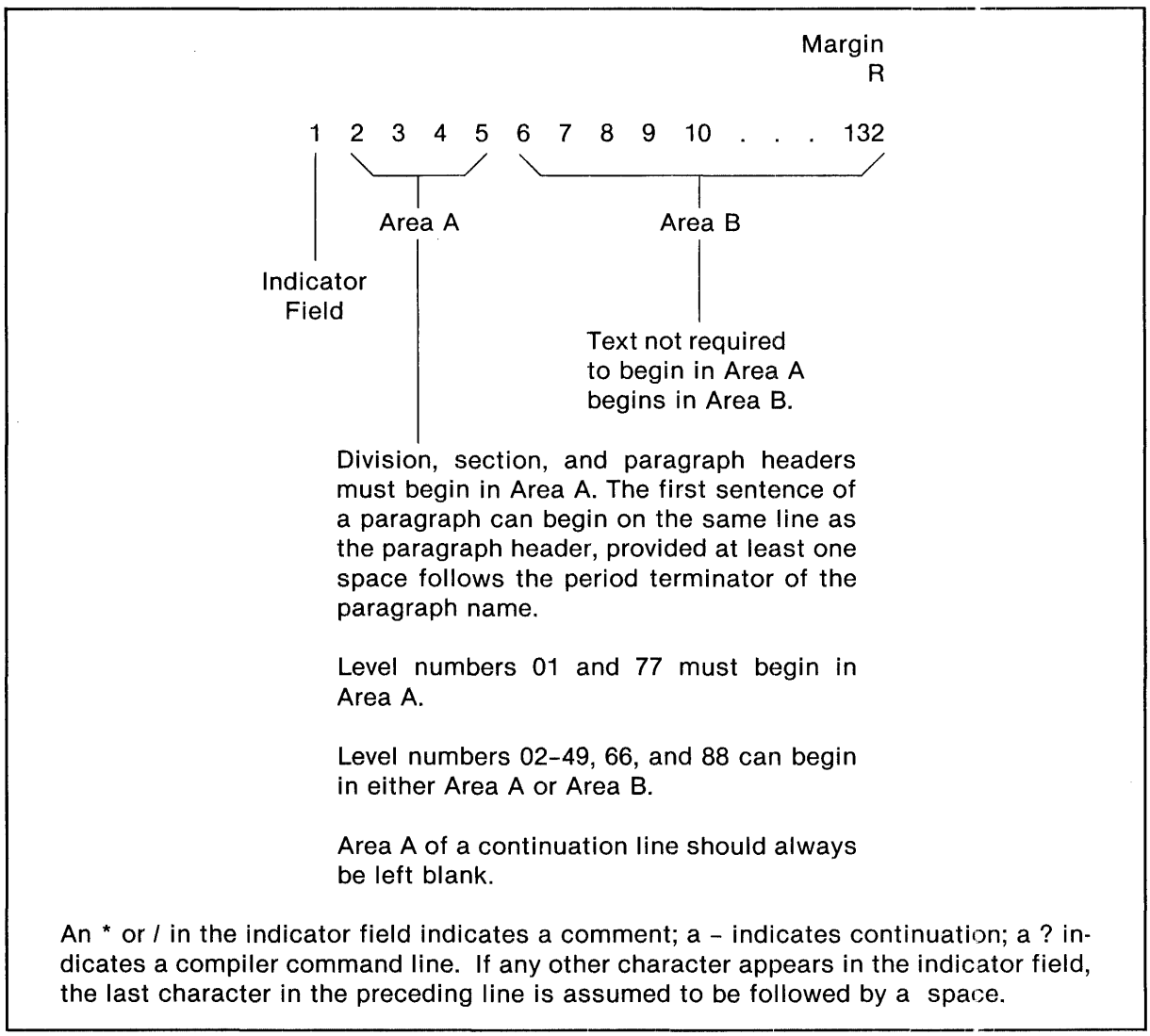


Figure 2-1. Tandem Standard Reference Format

ANSI Standard Reference Format

Each line in ANSI Standard reference format has 80 characters. The SCREEN COBOL compiler assures this by truncating lines over 80 characters, or adding blanks to fill out short lines.

A literal string that exceeds one line must fill the line on which it begins; otherwise, any trailing blanks are included as part of the literal before the continued characters.

The sequence number area (1 through 6) assigns a number to each line of code or labels a line with any combination of ASCII characters.

The positions following Margin R (73 through 80) represent the identification field. Their contents, which can include any ASCII character, are treated as a comment, and have no effect on the meaning of the program.

The ANSI standard reference format is illustrated in Figure 2-2.

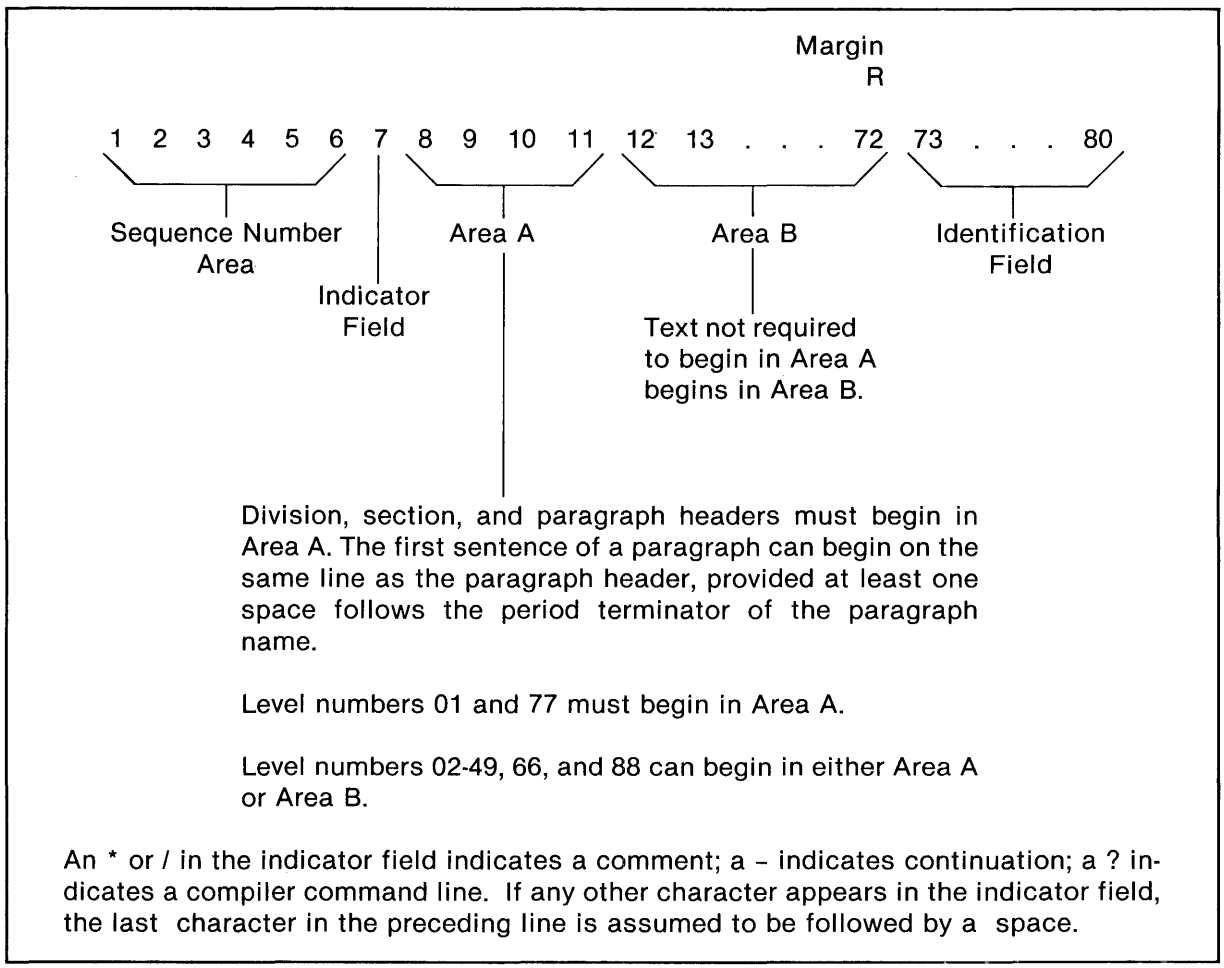


Figure 2-2. ANSI Standard Reference Format

Comment Lines

Comment lines can appear anywhere in a SCREEN COBOL program. Comment lines are indicated by a special character in the indicator field, which is column 1 in the Tandem standard reference format and column 7 in the ANSI standard reference format. The characters and their functions are as follows:

* An asterisk in the indicator field indicates the entire line is a comment.

/ A slash in the indicator field indicates the entire line is a comment. When a listing of the program is printed, a page eject is performed before printing the comment line.

Continuation Lines

Any word or literal in a SCREEN COBOL program can be continued. Continuation lines are indicated by the hyphen character in the indicator field.

If the previous line has a nonnumeric literal without a closing quotation mark, the first nonblank character in Area B of the continuation line must be a quotation mark. The continuation begins with the character immediately following that quotation mark.

Compiler Command Lines

Compiler command lines are indicated by the question mark character in the indicator field. The line is an instruction for the SCREEN COBOL compiler.

Normally, a compiler line in ANSI standard reference format is identified by a question mark in column 7; however, the SCREEN COBOL compiler interprets any line with a question mark in column 1 as a compiler command, even when the ANSI standard reference format is being used. In this special case, the line is treated as beginning with the indicator field; no sequence number area exists. Refer to Section 7 for detailed information regarding compiler commands.

ARITHMETIC OPERATIONS

Arithmetic operations are specified in the Procedure Division with the ADD, COMPUTE, DIVIDE, MULTIPLY, and SUBTRACT statements. These operations have the following common features:

- The data descriptions of the operands do not have to be the same. Any necessary conversion and decimal point alignment is supplied throughout the calculation.

- The maximum size of each operand is 18 decimal digits.

- Each arithmetic operation is evaluated using an intermediate data item. If the size of the result being developed is larger than this intermediate data item, the SCREEN COBOL program will be suspended by the TCP with an arithmetic overflow error. The contents of the intermediate data item are moved to the receiving data item according to the rules of a MOVE statement.

When a sending and receiving item in an arithmetic statement share part of their storage areas, the result is undefined.

Arithmetic Expressions

An arithmetic expression is one of the following:

- A numeric elementary item
- A numeric literal
- A numeric elementary item and a numeric literal separated by arithmetic operators
- An arithmetic expression enclosed in parentheses

Data items and literals appearing in an arithmetic expression must be either numeric elementary items or numeric literals on which arithmetic operations can be performed. Any arithmetic expression can be preceded by a plus or minus sign.

Arithmetic Operators

Four binary arithmetic operators and two unary arithmetic operators are used in arithmetic expressions. These operators are represented by specific characters, and must be preceded and followed by a space. Arithmetic operators are listed in Table 2-6.

Table 2-6. Arithmetic Operators

Binary Arithmetic Operators	Meaning
+	Addition
–	Subtraction
*	Multiplication
/	Division
**	Exponentiation
Unary Arithmetic Operators	
+	The effect of multiplying by +1
–	The effect of multiplying by –1

When a plus or minus sign immediately precedes a numeric literal (with no intervening spaces) the sign becomes a part of that literal, making it a signed numeric literal. The sign is neither a binary or unary operator. For example:

X +2 is equivalent to X, +2

X, +2 is two separate expressions.

A plus sign in any other situation is treated as a binary operator if it is preceded by an operand, and treated as a unary operator if it is not preceded by an operand. For example:

X + 2 and X + + 2 are equivalent expressions.

Evaluation of Expressions

Parentheses can be used to specify the order in which the operations of an arithmetic expression are performed. Expressions within parentheses are evaluated first. Evaluation of expressions within nested parentheses proceeds from the innermost set to the outermost set. When parentheses are not used, or expressions in parentheses are at the same level, the order of execution is as follows:

1st — Unary plus and minus
2nd — Multiplication and division
3rd — Addition and subtraction

Parentheses are used to eliminate ambiguities in logic or to modify the normal sequence of execution in expressions where it is necessary to have some deviation. When the sequence of execution is not specified by parentheses, the order for consecutive operations at the same level is from left to right. The following example illustrates the normal evaluation order in the absence of parentheses:

a + b / c + d * f − g

would be interpreted as:

(a + (b / c)) + (d * f) − g

with the sequence of operations proceeding from the innermost parentheses to the outermost.

Expressions ordinarily considered ambiguous, such as:

a / b * c, a / b / c

are permitted in SCREEN COBOL. They are interpreted as if they were written:

(a / b) * c, (a / b) / c.

Data items and literals appearing in an arithmetic expression must represent either numeric elementary data items or numeric literals.

MULTIPLE RESULTS. The ADD, COMPUTE, MULTIPLY, and SUBTRACT statements can have multiple results. Such statements behave as though they had been written in the following way:

1. One statement performs all necessary arithmetic to arrive at a result, and stores that result in a temporary storage location.

2. A sequence of statements transfers or combines the value of this temporary location with each result. These statements are considered to be written in the same left-to-right sequence that the multiple results are listed.

For example,the result of the following statement:

```
ADD a, b, c TO c, d(c), e
```

is equivalent to:

```
ADD a, b, c GIVING temp
ADD temp TO c
ADD temp TO d (c)
ADD temp TO e
```

where *temp* is the temporary storage location.

INTERMEDIATE RESULTS. Intermediate results are maintained by SCREEN COBOL during the evaluation of arithmetic expressions. The maximum number of digits held for an intermediate result is 18. If this limit is exceeded, arithmetic overflow occurs.

The following abbreviations are used to explain intermediate operations:

IR is the number of integer places carried for an intermediate result.

DR is the number of decimal places carried for an intermediate result.

OP1 is the first operand in an arithmetic expression, which has the form 9(I1)V9(D1), where I1 is the number of integer places carried and D1 is the number of decimal places carried for the first operand.

OP2 is the second operand in an arithmetic expression, which has the form 9(I2)V9(D2), where I2 is the number of integer places carried and D2 is the number of decimal places carried for the second operand.

OPR is the desired result, which has the form 9(IR)V9(DR), where IR is the number of places carried for the integer result and DR is the number of places carried for the decimal result.

Operation	Decimal Places
OP1 { + or − } OP2	DR is the greater of D1 or D2. IR is the lesser of (the greater of I1 or I2) or 18 − DR.
OP1 * OP2	DR is the greater of D1 or D2. IR is the lesser of (I1 + I2) or 18 − DR.

OP1 / OP2

DR is the greater of D1 or 1.
IR is the lesser of (I1 + D2) or 18 – DR.

If (I1 + D2 + DR) is greater than 18, the low order digits of the quotient are lost; in other words, any part of the quotient less than $10^{(I1+D2+DR-18)}$ is lost.

A normal divide computation proceeds as follows:

Example 1

```
03 A1 PIC S9(9)V9(9) VALUE 2.
03 A2 PIC S9(9)V9(8) VALUE 3.
03 AR PIC SV9(9).
       .
       .
       .
DIVIDE A1 BY A2 GIVING AR.
```

where:

```
3.00000000 | 2.000000000
```

is computed as:

```
                        00000000000000000.6
000000003.00000000 | 000000002.000000000
```

then moved to *AR* as: .600000000

Example 2

```
03 A1 PIC S9(2)V9(9) VALUE 2.
03 A2 PIC S9(2)V9(8) VALUE 3.
03 AR PIC SV9(9).
       .
       .
       .
DIVIDE A1 BY A2 GIVING AR.
```

where:

```
3.00000000 | 2.000000000
```

is computed as:

```
                 000000000.666666666
03.00000000 | 02.0000000000000000
```

then moved to *AR* as: .666666660

When a division operation in an arithmetic expression involves a COMPUTE statement or a relational expression, the intermediate results are evaluated in two steps:

1. the actual division
2. the adjustment of that result for use in further computations

Example 3

With

```
COMPUTE AX = A1/A2 + A3 * A4.
```

or

```
IF A1/A2 + A3 * A4 LESS THAN AX GO TO ...
```

the division is performed before further evaluation of either of the above statements. The intermediate result is then adjusted to fit the conceptual PICTURE derived by examining the other operands in the expression.

INCOMPATIBLE DATA. An incompatible data condition occurs when a data item is referenced in the Procedure Division and that item contains characters not permitted by its PICTURE clause. For example:

> If a position in a display numeric item contains an alphabetic character, A, and that item is used as an operand in an ADD statement, an incompatible data condition occurs. The result of this reference is undefined.

The class condition test is an exception to this rule because its purpose is to determine whether or not items contain legal data.

CONDITIONAL EXPRESSIONS

Conditional expressions identify conditions that are tested by the program to select between alternate paths of control. Conditional expressions are specified in the IF and PERFORM statements.

The two categories of conditions for conditional expressions are: simple conditions and complex conditions. Either kind of condition can be enclosed within any number of paired parentheses without changing the category of the condition.

Simple Conditions

Simple conditions are: class, condition-name, relation, and sign conditions. A simple condition has a truth value of true or false. Parentheses can enclose a simple condition without changing the truth value of the condition.

Simple conditions are described in the following paragraphs.

CLASS CONDITION. The class condition determines whether a DISPLAY item value is numeric or alphabetic.

Class condition syntax is:

```
data-name [ IS ] [ NOT ] { NUMERIC    }
                         { ALPHABETIC }
```

When NOT is included, the test condition is reversed. NOT NUMERIC tests for a field being nonnumeric; NOT ALPHABETIC tests for a field being nonalphabetic.

The NUMERIC test cannot be used with an item described as alphabetic. The NUMERIC test cannot be used with a group item composed of elementary items with data descriptions that include operational signs. If the data item being tested is signed, the item is numeric only if the contents are numeric and a valid sign is present. If the item is not signed, the item passes the test only if the contents are numeric and no sign is present. Valid signs for items with SIGN IS SEPARATE clause are + and −.

The ALPHABETIC test cannot be used with an item described as numeric.

CONDITION-NAME CONDITION. A condition-name condition determines whether or not the value of a conditional variable is equal to one of the values predefined for the condition-name.

Condition-name condition syntax is:

```
condition-name
```

The *condition-name* must be a level 88 item defined in the Data Division and given a value or a range of values.

The condition is true if the value of the conditional variable is equal to one of the *condition-name* values or falls within one of the ranges of values (including both ends of the range) given with the *condition-name.*

RELATION CONDITION. A relation condition causes a comparison of two values. Each value can be a data item, a literal, or a value resulting from an arithmetic computation; both values cannot be literals. A relation condition has a truth value of true if the relation exists between the values.

Relation condition syntax is:

```
value-1 IS ( [ NOT ] { LESS [ THAN ] }     )   value-2
           (         { <             }     )
           (                                 )
           ( [ NOT ] { EQUAL [ TO ] }        )
           (         { =           }         )
           (                                 )
           ( [ NOT ] { GREATER [ THAN ] }    )
           (         { >              }      )
```

The relational operators < = > determine the type of comparison made. A space must precede and follow each word of the relational operator. When NOT is included, the word NOT and the next keyword or relation character are one operator. NOT EQUAL is a truth test for an unequal comparison; NOT GREATER is a truth test for an equal or less comparison.

Two numeric values can be compared regardless of their usage (as defined by a USAGE clause). For all other comparisons, however, the values must have the same usage. If either of the values is a group item, nonnumeric comparison rules apply.

Comparison of Numeric Operands. Comparison of numeric operands is made with respect to the algebraic value of the operands. The length of the literal or arithmetic expression operands, in terms of the number of digits represented, is not significant. Zero is considered a unique value regardless of the sign.

Comparison of these operands is permitted regardless of the manner in which their usage is described. Unsigned numeric operands are considered positive.

Comparison of Nonnumeric Operands. Comparison of nonnumeric operands, or one numeric and one nonnumeric operand, is made with respect to the ASCII collating sequence of characters. The size of an operand is its total number of characters.

A noninteger numeric operand cannot be compared to a nonnumeric operand.

Numeric and nonnumeric operands can be compared only when their usage is the same. The following conventions apply:

- The numeric operand must be an integer data item or an integer literal.
- If the nonnumeric operand is an elementary data item or a nonnumeric literal, the numeric operand is treated as though it were moved to an elementary alphanumeric data item of the same size as the numeric data item; the content of this alphanumeric data item is then compared to the nonnumeric operand.
- If the nonnumeric operand is a group item, the numeric operand is treated as though it were moved to a group item of the same size as the numeric data item; the content of this group item is then compared to the nonnumeric operand.

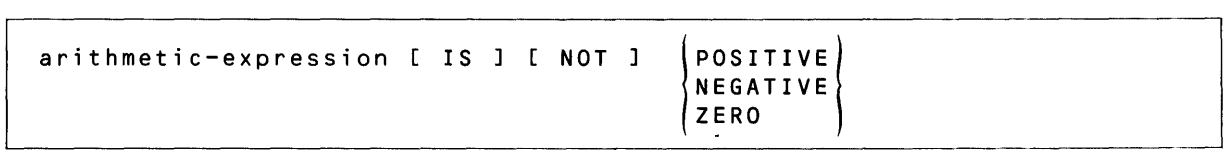
Comparison of Equal Sized Operands. If the values of operands are equal in size, characters in corresponding positions are compared starting from the high order end. This continues until either a pair of unequal characters is found or the low order end is reached. The values are equal when all pairs of characters are the same through the last pair.

The first pair of unequal characters is compared to determine their relative position in the collating sequence. The value having the character that is higher in the collating sequence is the greater value.

Comparison of Unequal Sized Operands. If the values of operands are unequal in size, comparison proceeds as though the shorter operand were extended on the right by sufficient spaces to make the operands equal in size.

SIGN CONDITION. The sign condition determines whether or not the algebraic value of an arithmetic expression is greater than, less than, or equal to zero.

Sign condition syntax is:



Arithmetic-expression must have at least one variable.

When NOT is included, the word NOT and the next keyword specify one sign condition that defines the algebraic test to be executed for truth value. NOT ZERO is a truth test for a nonzero, positive, or negative value. An item is positive if its value is greater than zero, negative if its value is less than zero, and zero if its value is equal to zero.

Complex Conditions

Complex conditions are formed by using simple conditions, combined conditions and/or complex conditions with logical connectives AND or OR, or negating these conditions with keyword NOT. The truth value of a complex condition, whether or not the value is enclosed in parentheses, is that truth value which results from the interaction of all the logical operators on the individual truth values of simple conditions, or on the intermediate truth values of conditions connected or negated.

Logical operators and their definitions are listed in Table 2-7.

Table 2-7. Logical Operators

Logical Operator	Definition
AND	Logical conjunction—the truth value is true if both conditions are true, and false if one or both are false.
OR	Logical inclusive OR—the truth value is true if one or both of the conditions is true, and false if both conditions are false.
NOT	Logical negation or reversal of truth value—the truth value is true if the condition is false and false if the condition is true.

The logical operators must be preceded by a space and followed by a space.

NEGATED SIMPLE CONDITION. A simple condition is negated through the use of the logical operator NOT. The negated simple condition effects the opposite truth value for a simple condition. Parentheses enclosing negated simple condition do not change the truth value.

Negated simple condition syntax is:

```
NOT simple-condition
```

COMBINED AND NEGATED COMBINED CONDITIONS. A combined condition results from connecting conditions with AND or OR. Each condition can be a simple condition, a negated condition, a combined condition or negated combined condition, or a combination of these.

Combined and negated combined condition syntax is:

```
condition  { { AND } condition } ...
           { { OR  }           }
```

ABBREVIATED COMBINED RELATION CONDITIONS. In a relation where one item is compared to several others, the relation can be abbreviated by leaving out the subject item name after the first reference to it. If the relational operator is the same as the previous operator, the operator can also be omitted.

Abbreviated combined relation condition syntax is:

```
condition { {AND} [ not ] [ operator ] object } ...
          { {OR  }                           }
```

If NOT appears within the abbreviated condition and is not followed by an operator, the keyword negates that portion of the condition, but does not automatically carry forward to the next relation.

The following examples illustrate abbreviated combined relation conditions and their expanded equivalents.

Abbreviated Combined Relation Condition	Expanded Equivalent
`a > b AND NOT < c OR d`	`((a > b) AND (a NOT < c)) OR (a NOT < d)`
`a NOT EQUAL b OR c`	`(a NOT EQUAL b) OR (a NOT EQUAL c)`
`NOT a = b OR c`	`(NOT (a = b)) OR (a = c)`
`NOT (a GREATER b OR < c)`	`NOT ((a GREATER b) OR (a < c))`
`NOT (a NOT > b AND c AND NOT d)`	`NOT ((((a NOT > b) AND (a NOT > c)) AND (NOT (a NOT > d))))`
`(a + b - c) > d AND NOT < e OR f`	`(a + b - c) > d AND (a + b - c) NOT < e OR (a + b - c) NOT < f`

Condition Evaluation Rules

Parentheses are used to change the order in which individual conditions are evaluated when it is necessary to depart from the standard precedence. Conditions within parentheses are evaluated first. When conditions are within nested parentheses, evaluation goes from the innermost condition to the outermost condition.

When parentheses are not used or when conditions in parentheses are at the same level, the following order of evaluation is used until the final truth value is determined:

1. Values are established for arithmetic expressions.

2. Truth values for simple conditions are established in the following order:

   relation
   class
   condition-name
   sign

3. Truth values for negated simple conditions are established.

4. Truth values for combined conditions are established:

   AND logical operators, followed by OR logical operators.

5. Truth values for negated combined conditions are established.

6. When the sequence of evaluation is not completely specified by parentheses, the order of evaluation of consecutive operations of the same hierarchical level is from left to right.

TABLES

Tables of data are common in data processing problems. For example, a data structure might have 20 total fields, described as twenty identical data items named total-one, total-two, ..., total-twenty. This would mean twenty different names, which could obscure the interrelated nature of the totals and make references awkward. A table structure simplifies this problem.

Tables are defined by using an OCCURS clause in their data description. This clause specifies that an item is repeated as many times as stated. The item is considered to be a table element, and its name and description apply to each repetition. As an example, the one-dimensional table mentioned in the preceding paragraph could be defined with this entry:

```
02  total  OCCURS 20 TIMES ...
```

In the Screen Section, a table must be an elementary item. In the Working-Storage Section and Linkage Section, the elements of a table can be groups of subordinate structures, some of which can also be tables. Thus, the previous example might appear in greater detail as:

```
02  total-g  OCCURS 20 TIMES.
    03 total-a ...
    03 total-b  OCCURS 3 TIMES ...
```

The expanded example describes *total-a* as a one-dimensional table, and describes *total-b* as a two-dimensional table because an OCCURS clause is applied to an item subordinate to the first OCCURS clause. If the description of a data item subordinate to *total-b* also had an OCCURS clause, the item would be a three-dimensional table. SCREEN COBOL allows a maximum of three dimensions in the Working-Storage Section and Linkage Section.

Frequently, tables are built in Working-Storage with constant values that a program needs in addition to the data from external sources. An example of coding for a table containing the full calendar month names is shown in Figure 2-3.

```
WORKING-STORAGE SECTION.

01  month-name-table.
    05  FILLER                          PIC X(9) VALUE "JANUARY".
    05  FILLER                          PIC X(9) VALUE "FEBRUARY".
    05  FILLER                          PIC X(9) VALUE "MARCH".
    05  FILLER                          PIC X(9) VALUE "APRIL".
    05  FILLER                          PIC X(9) VALUE "MAY".
    05  FILLER                          PIC X(9) VALUE "JUNE".
    05  FILLER                          PIC X(9) VALUE "JULY".
    05  FILLER                          PIC X(9) VALUE "AUGUST".
    05  FILLER                          PIC X(9) VALUE "SEPTEMBER".
    05  FILLER                          PIC X(9) VALUE "OCTOBER".
    05  FILLER                          PIC X(9) VALUE "NOVEMBER".
    05  FILLER                          PIC X(9) VALUE "DECEMBER".

01  month-name-table REDEFINES month-name-table.
    05  month-name  OCCURS 12 times    PIC X(9).
```

Figure 2-3. Sample Table Structure

The term FILLER is a keyword that takes the place of a data name when it is unimportant to name an item. Because occurrences of a table element do not have individual names, a reference to an occurrence must give its position number along with the data name of the table. The method of giving the position number, called subscripting, is described later in this section.

DATA REFERENCE

All items must be named so they can be referenced. Items given unique names can be referenced with no difficulty, but many programs contain items that do not have unique names. All elements of a table, for example, share a single name. Also, the same name can be used for more than one data item, and the same paragraph name can be used in different sections of the Procedure Division.

Names must be unique or made unique through qualification or subscripting.

Qualification

Every name must be unique, either because no other name has the same spelling and hyphenation, or because the name is subordinate to a unique name. In the latter case, including one or more of the higher level names qualifies the subordinate item and makes it unique. Although enough qualification must be present to make a name unique, it is not necessary to include all levels.

For data name references, group names can be used for qualification. Level 01 names are the most significant qualifiers, then level 02, and so forth.

For condition-name references, the name of the condition variable can be used as qualification, even if the variable is an elementary item.

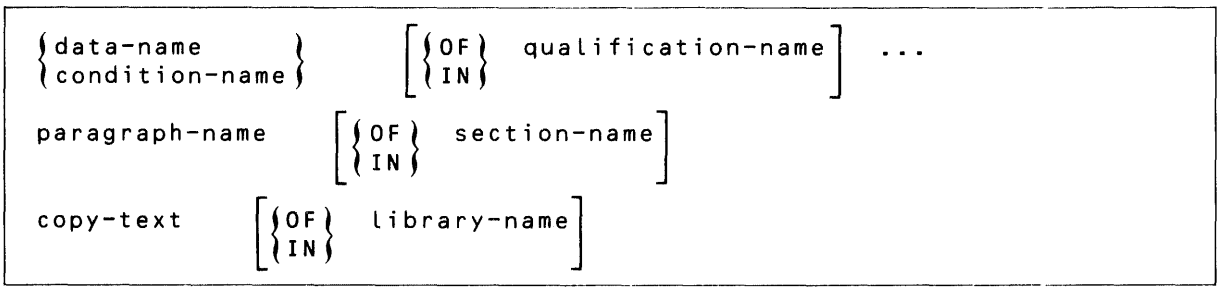
For paragraph name references, the section name is the only qualifier available. References to paragraphs within the same section never require qualification.

For copy text references in COPY statements, the copy text name must be qualified if the text library that defines it is not the default library for the compilation.

Level 01 names and section names must be unique because they cannot be further qualified. Regardless of available qualification, a name cannot be both a data name and a procedure name.

An item is qualified by following a data name, a condition-name, a paragraph name, or a copy text name by one or more phrases composed of a qualifier preceded by connective IN or OF. IN and OF are equivalent.

Qualification syntax is:



Qualification rules are as follows:

- Each qualifier must be at a higher level than the previous one, and must stay within the same structure of the name it qualifies.
- The same name cannot appear at different levels in a structure; otherwise, the name could qualify itself.
- If a data name or a condition-name is assigned to more than one data item, the data name or condition-name must be qualified each time it is referenced (except in the REDEFINES clause where, by context, qualification is unnecessary).
- A paragraph name cannot be duplicated within a section. Within its own section a paragraph name does not require qualification. When a section name is used to qualify a paragraph name, the word SECTION is not part of the name.
- A data name used as a qualifier is not subscripted, even if the data name is described with an OCCURS clause.
- A name can be qualified even when the name is unique.
- If more than one combination of qualifiers is available to make a name unique, any combination can be used.

In the following example, all data names except prefix are unique. Qualification must be used to reference either prefix item.

```
01 transaction-data ...          01 master-data ...
   03 item-no ...                   03 code-no ...
      05 prefix ...                    05 prefix ...
      05 code ...                      05 suffix ...
   03 quantity ...                  03 description ...
```

Using the same example, any of the following sentences could be used to move the contents of one prefix to the other prefix:

```
MOVE prefix OF item-no TO prefix OF code-no.
MOVE prefix OF item-no TO prefix OF master-data.
MOVE prefix OF transaction-data TO prefix IN code-no.
MOVE prefix IN transaction-data TO prefix IN master-data.
```

Subscripting

Subscripts are used to reference elements in a table. Subscripts are needed because all table elements have the same name.

The subscript can be an integer numeric literal or a data item that represents a numeric integer. When the subscript is a data item, the data item name can be qualified, but not subscripted itself. The subscript can be signed and, if signed, it must be positive.

The lowest possible subscript value is 1. This value selects the first element of a table. The other elements of the table are selected by subscripts whose values are 2, 3, 4, and so forth. If a subscript value greater than the size of the table is used, the result is undefined.

The subscript, or set of subscripts, is enclosed in parentheses and appended to the element name of the table. When more than one subscript is required, they are written in the order of most significant value to least significant value.

Subscript syntax is:

```
{data-name     } ( sub-1 [ , sub-2 [ , sub-3 ] ] )
{condition-name}
```

The following examples illustrate subscripting:

```
MOVE total(8) TO report-total-8.

MOVE day of date(3) TO print-line-date.

MOVE month-name(month-number) TO report-month.

MOVE matrix(row, column) TO output-display-line.
```

Using Identifiers

An identifier is a data name made unique by qualifiers, subscripts, or qualifiers and subscripts. A data name being used as a subscript or qualifier cannot itself be subscripted.

Identifier syntax is:

```
data-name-1  [ {OF} data-name-2 ]  ...
             [ {IN}             ]

  [ ( sub-1 [ , sub-2 [ , sub-3 ] ] ) ]
```

The following examples illustrate specification of identifiers:

```
unique-identifier

item-1 OF group-a

element OF name-table OF master-data (master-num)
```

Using Condition Names

Items are tested frequently by a program. Assigning a condition-name to an item is a convenient way to reference the item and determine its value.

Every condition-name must be unique or capable of being made unique through qualification and/or subscripting. If qualification is used to make a condition-name unique, the conditional variable can be used as the first qualifier. The containing data names of the conditional variable can also be used as qualifiers. If references to a conditional variable require subscripting, then any of its condition-names must have the same subscripting.

The following example illustrates a condition-name called restricted-use:

```
01 inventory.
   02 part-number OCCURS 100 TIMES ...
      03 prefix              PIC 99.
      03 use-code            PIC 9.
         88 restricted-use          VALUE 1.
      03 supplier-suffix     PIC 99.
```

The condition-name, *restricted-use*, might be referenced as:

```
IF restricted-use OF use-code IN part-number (30)
    NEXT SENTENCE,
ELSE...
```

DATA REPRESENTATION

In the Working-Storage Section and Linkage Section, data items are stored in a certain number of bytes; each byte is an 8-bit unit of storage. Bytes are grouped in pairs to form words.

Data items whose usage (as defined by a USAGE clause) is DISPLAY occupy one byte per character. Data items whose usage is COMPUTATIONAL occupy storage as follows:

PICTURE Size in Digits	Storage Occupied
1 to 4	2 bytes
5 to 9	4 bytes
10 to 18	8 bytes

In the Screen Section, items do not have individual storage assigned; storage of these items is of no consequence to SCREEN COBOL programming.

Standard Alignment

The standard rules for positioning data within an elementary item depend on the category of the receiving item. The rules are as follows:

- If the receiving data item is described as numeric, the sending data is aligned either by decimal point with zero fill on either end of the value or by truncation on the low end, as required. Truncation on the high end is not permitted, and if required, causes suspension of the program. When no decimal point is specified, the receiving data item is treated as if it had an assumed decimal point immediately following the rightmost character.

- If the receiving data item is described as alphanumeric or alphabetic, the sending data is aligned at the leftmost character position in the data item with space fill or truncation to the right as required.

Optional Alignment

Standard data representation and alignment rules are not always appropriate, so provisions exist to override them. The JUSTIFIED clause can be used in the data description to right justify data within a data item.

Sometimes a server requires that data items in messages be aligned on word boundaries. Data items aligned on word boundaries are said to be synchronized. Synchronization typically is achieved by organizing and describing data so that item boundaries coincide with word boundaries. This task can be eliminated by using the SYNCHRONIZED clause to force alignment of data items to their natural boundaries.

SECTION 3

IDENTIFICATION DIVISION

The Identification Division identifies the SCREEN COBOL program. The division has one required paragraph and five optional paragraphs. If other paragraphs are present, they are treated as comments.

The format of the Identification Division is shown in Figure 3-1.

```
   IDENTIFICATION DIVISION.

   PROGRAM-ID. program-unit-name.

[ AUTHOR. [ comment-entry ] ]

[ INSTALLATION. [ comment-entry ] ]

[ DATE-WRITTEN. [ comment-entry ] ]

[ DATE-COMPILED. [ comment-entry ] ]

[ SECURITY. [ comment-entry ] ]
```

Figure 3-1. Identification Division Format

The division header is

```
IDENTIFICATION DIVISION.
```

The header must begin in Area A and must be terminated with a period separator.

Optional paragraphs AUTHOR, INSTALLATION, DATE-WRITTEN, and SECURITY are included for documentation purposes only. The comment-entry parameter for these paragraphs can be any combination of characters from the SCREEN COBOL character set and represents text describing each paragraph heading.

PROGRAM-ID PARAGRAPH

The required PROGRAM-ID paragraph names the SCREEN COBOL program unit.

The syntax of the PROGRAM-ID paragraph is:

```
PROGRAM-ID. program-unit-name
```

where

program-unit-name

is the name of the SCREEN COBOL program unit; the name can have from 1 through 30 alphanumeric characters. The name can differ from the file name of the source code or the object file. This name is used in a CALL statement when the program is referenced in another SCREEN COBOL program unit. This name is also used by the PATHCOM SET TERM INITIAL command.

DATE-COMPILED PARAGRAPH

The optional DATE-COMPILED paragraph causes the compiler to generate the current date and time and insert it in this line of the source listing.

The syntax of the DATE-COMPILED paragraph is:

```
DATE-COMPILED. [ comment-entry ]
```

where

comment-entry

is any combination of characters from the SCREEN COBOL character set.

When this paragraph is included, the compiler generates the current date and time, replacing the DATE-COMPILED line and any comment-entry with this line:

DATE COMPILED. yy/mm/dd - hh:mm:ss

yy	is the year	range 00-99
mm	is the month	range 01-12
dd	is the day	range 01-31
hh	is the hour	range 00-23
mm	is the minute	range 00-59
ss	is the second	range 00-59

SECTION 4

ENVIRONMENT DIVISION

The Environment Division declares the operating environment of the program unit and provides optional error reporting for screen input operations. The division has two sections: a required Configuration Section and an optional Input-Output Section.

The format of the Environment Division is shown in Figure 4-1.

```
   ENVIRONMENT DIVISION.

   CONFIGURATION SECTION.

   SOURCE-COMPUTER. comment-entry

   OBJECT-COMPUTER. object-computer-entry

[ SPECIAL-NAMES. special-names-entry ]

[ INPUT-OUTPUT SECTION. input-output-entry ]
```

Figure 4-1. Environment Division Format

The division header is:

```
ENVIRONMENT DIVISION.
```

The header must begin in Area A and must be terminated with a period separator.

CONFIGURATION SECTION

The required Configuration Section declares the operating environment of the program unit. These declarations can include terminal type characteristics and screen display attributes.

The section header is:

```
CONFIGURATION SECTION.
```

The header must begin in Area A and must be terminated with a period separator.

The Configuration Section contains three paragraphs:

SOURCE-COMPUTER paragraph (required)

OBJECT-COMPUTER paragraph (required)

SPECIAL-NAMES paragraph (optional)

SOURCE-COMPUTER Paragraph

The required SOURCE-COMPUTER paragraph names the computer system by which the program unit is compiled. The SCREEN COBOL compiler assumes the system is a Tandem system and treats any name given as a comment.

The SOURCE-COMPUTER paragraph syntax is:

```
SOURCE-COMPUTER. comment-entry.
```

where

comment-entry

is one or more words. *comment-entry* cannot be blank or null characters.

OBJECT-COMPUTER Paragraph

The required OBJECT-COMPUTER paragraph names the computer system on which the object program runs. The SCREEN COBOL compiler assumes the system is a Tandem system and treats the name given as a comment.

The OBJECT-COMPUTER paragraph syntax is:

```
OBJECT-COMPUTER. comment-word,
 [ TERMINAL IS terminal-type ]
 [ CHARACTER-SET IS character-set-type ] .
```

where

comment-word

is a single word only.

TERMINAL IS terminal-type

specifies the type of terminal for which the program is intended. The terminal type entry is one of the following keywords:

T16-6510 T16-6520 T16-6530	These keywords denote a Tandem product number
IBM-3270	This keyword denotes a terminal that is one of the IBM-3270 family, or a terminal that is program-compatible with such a terminal.

Program units compiled for a T16-6520 terminal can be run on a T16-6530 terminal. If T16-6520 is specified as terminal-type and the program unit runs on a T16-6530 terminal, features unique to the T16-6530 are prohibited.

CONVERSATIONAL	This keyword denotes that a terminal operates in conversational mode regardless of the terminal type.

Program units compiled for conversational mode can be run on T16-6510, T16-6520, and T16-6530 terminals or on any device operating as a conversational mode terminal as recognized by the GUARDIAN File System. Features unique to the block mode terminal types are not recognized by the conversational mode SCREEN COBOL program.

If the TERMINAL IS specification is omitted, the program can run on all of the terminal types listed above. Features unique to a particular terminal cannot be used.

CHARACTER-SET IS character-set-type

provides limited support of national use characters, that is, foreign character sets that are not USASCII. This parameter can be used only with the T16-6530 terminal and if specified, must follow the TERMINAL IS specification.

character-set-type is one of the following keywords:

USASCII	US ASCII
FRANCAIS-AZ	French AZERTY
FRANCAIS-QW	French QWERTY
DEUTSCH	German/Austrian
ESPANOL	Spanish
UK	United Kingdom
SVENSK-SUOMI	Swedish/Finnish
DANSK-NORSK	Danish/Norwegian

If the CHARACTER-SET specification is omitted, *USASCII* is used.

If the CHARACTER-SET specification is included and the character set type differs from the current setting in the terminal or the terminal setting is unknown, the terminal is signalled the character set type at the first DISPLAY BASE statement in a program unit. After the program unit completes execution, the terminal is reset to its original character set.

Programmatic support of national use characters is in the following areas:

- field characteristic clause UPSHIFT—Lowercase national use characters are upshifted to their uppercase equivalents.
- class condition—The condition ALPHABETIC checks for characters in the national use characters.
- symbol A in PICTURE clauses—A check is made for characters in the national use character set.

Programmatic support of national use characters does not affect the following areas:

- field characteristic clause MUST BE—Range tests are not supported for national use characters.
- tests that involve collating sequence matters—Any comparison tests, such as less-than or greater-than relations, are not supported for national use characters.

SPECIAL-NAMES Paragraph

The optional SPECIAL-NAMES paragraph allows you to select names and have those names assigned to certain system names. The paragraph also matches features of a specific terminal with the words used in the program to refer to those features. With careful use of the correspondences established in this paragraph, you can remove much of the dependence on terminal type from the body of the program unit.

The SPECIAL-NAMES paragraph syntax is:

```
SPECIAL-NAMES.

    [ {mnemonic-name IS  {system-name                 }}     ,...]
    [ {                  {( { system-name } ,... )    }}         ]

  [ , CURRENCY [ SIGN ] IS literal-1 ]

  [ , DECIMAL-POINT IS COMMA ] .
```

where

mnemonic-name

is an identifier you select to be associated with a *system-name*. The *mnemonic-name* can be used later in the Screen Section or the Procedure Division of the program to refer to a function key or display attribute indicated by *system-name*.

A list of system-names can be equated to a single *mnemonic-name* only if the system-names refer to display attributes; this causes the *mnemonic-name* to represent the combination of the display attributes. →

```
system-name
```

specifies a function key or display attribute available on the terminal. Table 4-1 lists the function keys and display attributes that can appear as a system-name.

```
CURRENCY [ SIGN ] IS literal-1
```

specifies a literal to be used instead of the dollar currency sign ($). *Literal-1* must be a single character and cannot be any of the following:

Digits	0 through 9
Characters	A B C D L P R S V X Z space
Special	* + – , . ; ( ) " / =

```
DECIMAL-POINT IS COMMA
```

exchanges the function of comma and period in PICTURE character strings and numeric literals in the remainder of the program.

Table 4-1. Function Key and Display Attribute System Names

System-Name (1)	Allowed for 6510	Allowed for 6520	Allowed for 6530	Allowed for 3270	Allowed for Conv. (4)
Function Key					
CLEAR (2)				X	
ENTER				X	
F1 - F16 (unshifted)	X	X	X		
SF1 - SF16 (shifted)	X	X	X		
NEXT-PAGE		X	X		
PA1 - PA3 (2)				X	
PA4 - PA10				X	
PF1 - PF24				X	
PREV-PAGE		X	X		
RETURN-KEY (3)			X		
ROLL-DOWN		X	X		
ROLL-UP		X	X		
Display Attribute					
BELL					X
BLINK	X	X	X		
BRIGHT				X	
DIM		X	X		
HIDDEN	X	X	X	X	X
MDTOFF		X	X	X	
MDTON		X	X	X	
NOBELL					X
NOBLINK	X	X	X		
NOREVERSE		X	X		
NORMAL	X	X	X	X	
NOTHIDDEN	X	X	X	X	X
NOUNDERLINE		X	X		
NUMERIC-SHIFT				X	
PROTECTED	X	X	X	X	
REVERSE		X	X		
UNDERLINE		X	X		
UNPROTECTED	X	X	X	X	

NOTES:

(1) System-name words are not reserved words.

(2) Used in ESCAPE clause of ACCEPT statement only.

(3) If the SCREEN COBOL program is to run on a Tandem 6530 terminal, a return key function can be defined in the SPECIAL-NAMES paragraph of the program. When the RETURN key is pressed a function code will be transmitted. If a function is not defined, no return key function code exists—pressing the return key will cause a forward tab action.

A return key function is local to a program unit. The first DISPLAY BASE statement of the program unit causes the terminal to adjust the RETURN key operation to the setting indicated by the executing program unit.

(4) Applies for any terminal specified CONVERSATIONAL in the OBJECT-COMPUTER paragraph.

The following example illustrates the SPECIAL-NAMES paragraph:

```
SPECIAL-NAMES.
    ENTER-KEY    IS F1,
    EXIT-KEY     IS F16,
    INPUT-ATTR   IS UNDERLINE,
    SIGNAL-ATTR IS (REVERSE, NOUNDERLINE).
```

INPUT-OUTPUT SECTION

The optional Input-Output Section provides error reporting for screen input operations. If this section is omitted, the error display attribute is dependent on the terminal type specified in the Configuration Section.

The section header is:

```
INPUT-OUTPUT SECTION.
```

The header must begin in Area A and must be terminated with a period separator.

The Input-Output Section syntax is:

```
SCREEN-CONTROL.

 ERROR-ENHANCEMENT [ IS ] mnemonic-name  [IN  {FIRST}]
                                        [    {ALL  }]

  [ WITH [ NO ] AUDIBLE ALARM ] .
```

where

ERROR-ENHANCEMENT [IS] mnemonic-name

specifies the display attribute with which fields found to be in error are to be enhanced. The BLINK attribute is used for the T16-6510, T16-6520, and T16-6530 terminals; the BRIGHT attribute is used for the IBM-3270 terminal.

IN FIRST

enhances the first field that is found to be in error.

IN ALL

enhances all fields that are found to be in error.

NOTE

For terminals operating in conversational mode, IN FIRST is the only recognized enhancement option. If IN ALL is specified, the phrase is ignored and the first field containing an error is enhanced.

WITH [NO] AUDIBLE ALARM

enables or disables the audible indicator when an error is detected.

Procedure Division ACCEPT statement processing checks the contents of input fields against the requirements of a PICTURE clause and any constraints, such as those imposed by a MUST BE field characteristic clause. ACCEPT processing attempts to indicate which field is in error. The ERROR-ENHANCEMENT option allows you to control some aspects of the error processing.

SECTION 5

DATA DIVISION

The Data Division describes the data that the program creates, accepts as input, manipulates, or produces as output. The division has three sections: a Working-Storage Section, a Linkage Section, and a Screen Section. Each section is optional and is included only when the type of data the section defines is used in the program.

Data described in the Data Division falls into two categories:

- Data formatted for display on a terminal or received as input from a terminal.
- Data developed internally by the program and placed in temporary areas described in the Working-Storage Section or Linkage Section.

The format of the Data Division is shown in Figure 5-1.

```
  DATA DIVISION.
[ WORKING-STORAGE SECTION.
    data-description-entries ]
[ LINKAGE SECTION.
    data-description-entries ]
[ SCREEN SECTION.
    input-control-entries     <---- For conversational mode only.
    screen-description-entries ]
```

Figure 5-1. Data Division Format

The division begins with a division header. The format of the header is:

```
DATA DIVISION.
```

The header must begin in Area A and must be terminated with a period separator.

DATA DIVISION SECTIONS

Three sections comprise the Data Division: Working-Storage Section, Linkage Section, and Screen Section. When all three sections are included in a program, they must be written in the order shown in Figure 5-1. Items within each section can be written in any order.

Each section describes a different type of data. Sections are defined as follows:

- Working-Storage Section—This section describes the structure of local data developed within the program. Data entries in this section are initialized each time the program unit is called; therefore, values are not retained between calls.
- Linkage Section—This section describes the structure of parameter data passed to a subprogram by a CALL statement. Items described in the calling program are referenced in the USING clause of the Procedure Division of a called program.
- Screen Section—This section describes the types and locations of fields in screens that can be displayed on the terminal. Screens described in the Screen Section are those that are referenced in the Procedure Division of the program.

Working-Storage Section

The Working-Storage Section defines records and miscellaneous data items that are used for internal purposes. Data entries in this section can be set to initial values. When local data items or intermediate storage is not necessary, this section can be omitted.

The section begins with a section header. The format of the header is:

```
WORKING-STORAGE SECTION.
```

The header must begin in Area A and must be terminated with a period separator.

Data description entries for individual items follow the header. All item names must be unique. Subordinate data names can be duplicated as long as they can be qualified.

Linkage Section

The Linkage Section describes data that is passed from one program to another and is available to both programs. This section is required when a program is called from another program. No local data is used in the called program for these items; the calling program item is used during execution of the called program.

The section begins with a section header. The format of the header is:

```
LINKAGE SECTION.
```

The header must begin in Area A and must be terminated with a period separator.

A Procedure Division reference in the called program accesses the location in the calling program. Statements within the Procedure Division of the called program can only reference Linkage Section items given in the Procedure Division header USING clause of the calling program. Subordinate data items and condition names can be used. The called program is invoked by a CALL statement with a USING clause corresponding to the USING clause of the calling program.

The structure of the Linkage Section is the same as that of the Working-Storage Section except the VALUE clause is prohibited for items other than level 88 items.

Screen Section

The Screen Section describes the screens that are referenced in the Procedure Division. The structure of the Screen Section is similar to that of the Working-Storage Section. The section makes provision for two types of screens: base and overlay.

The section begins with a section header. The format of the header is:

```
SCREEN SECTION.
```

The header must begin in Area A and be terminated with a period separator.

DATA STRUCTURE

Data is described through a set of entries that name the components of a structure, describe the attributes of those components, and describe the structure into which the components are organized. Each entry has a level number followed by a data name, and possibly a series of independent clauses. The level numbers depict the structure, dividing the data further and further down to its smallest parts.

The lowest subdivisions of a structure, that is, those not further subdivided, are called elementary items. A structure can be a single elementary item or a series of elementary items.

Sets of elementary items can be referenced by combining them into groups. Groups, in turn, can be combined into groups; an elementary item, therefore, can belong to more than one group.

Level Numbers 01-49

Level numbers 01 through 49 describe the hierarchy of data items. The structure itself is assigned level number 01.

The system of level numbers shows the relationship of elementary items to group items. Data items within a group are assigned level numbers higher than that of the group item. Level numbers within the group need not be consecutive, but they must be ordered so that the higher the level number the lower the entry in the hierarchy.

A group includes all group and elementary items following it until a level number less than or equal to the level number of that group is encountered. All items or groups immediately subordinate to a given group item must be described using identical level numbers greater than the level number of that group item.

An example of level numbering is shown in Figure 5-2.

```
01  address-data.
    05  office-number.
        10  district          PIC 99.
        10  region            PIC 999.
    05  office-address.
        10  street            PIC X(25).
        10  city              PIC X(15).
        10  state             PIC X(5).
        10  zip-code          PIC 9(5).

01  personnel-data.
    05  office-manager        PIC X(35).
    05  no-of-employees       PIC 9(4).
    05  tax-groups.
        10  hourly            PIC 9(3).
            15  part-time     PIC 99.
            15  full-time     PIC 99.
        10  exempt            PIC 9(4).
```

Figure 5-2. Level Numbering Within a Structure

Level Numbers 66, 77, and 88

Three additional types of data entries can exist in the Working-Storage Section and Linkage Section: level 66, level 77, and level 88 data entries. Entries that begin with these level numbers do not define the hierarchy of the item described. Entries are defined as follows:

- Level 66—A level 66 data entry specifies elementary items or groups introduced by a RENAMES clause. These entries are used to regroup contiguous elementary data items.
- Level 77—A level 77 data entry is an independent data item that is not a subdivision of another data item. The data item is not itself subdivided.
- Level 88—A level 88 entry defines a condition name, including a value or range of values that define the condition to be tested.

DATA DESCRIPTION ENTRY

A data description entry defines the characteristics of a data item. The entry can be used in the Working-Storage Section or Linkage Section of the SCREEN COBOL program.

Several forms are available to describe items for various purposes. Some entries cause the creation of items (memory space is allocated), while others supply alternative descriptions or reference points for already existing data. Others supply specification of value ranges for later testing.

A skeleton of the data description entry is shown in Figure 5-3.

```
WORKING-STORAGE SECTION.

      or

LINKAGE SECTION.

Format 1

level-number  {data-name-1}
              {FILLER     }

  [ JUSTIFIED clause ]

  [ OCCURS clause ]

  [ PICTURE clause ]

  [ REDEFINES clause ]

  [ SIGN clause ]

  [ SYNCHRONIZED clause ]

  [ USAGE clause ]

  [ VALUE clause ]  <----- For WORKING-STORAGE SECTION only.

Format 2

  [ 66 new-name [ RENAMES clause ] ]

Format 3

  [ 88 condition-name , [ VALUE clause ] ]
```

Figure 5-3. Data Description Entry Skeleton

Format 1 of Figure 5-3 describes data of levels 01 through 49 and level 77. The data-name-1 entry is the name of the storage area defined by the subordinate items. In the following example, *store-address* references everything from *street* through *zip-code*.

```
01  sample-record.
    05  store-id.
        10  store-number        PIC 999.
        10  store-region        PIC X.
    05  store-manager           PIC X(35).
    05  store-address.
        10  street              PIC X(25).
        10  city                PIC X(15).
        10  state               PIC X(2).
        10  zip-code            PIC 9(5).
    05  FILLER                  PIC X(14).
```

The FILLER keyword takes the place of a data name when it is unimportant to name an item. FILLER is commonly used when building Working-Storage records, such as error messages, where most of the text is groups of constants. The text groups can be separated by the filler. In the following example, FILLER defines an area in storage that cannot be referenced in the program except as part of the enclosing item, *first-record:*

```
01  first-record.
    05  record-code      PIC 99.
    05  record-type      PIC XX.
    05  FILLER           PIC X(30).
    05  division-code    PIC 999.
```

A level 77 entry cannot itself be subdivided. Level 77 entries, like level entries 01 through 49, must be immediately followed by a data name or keyword FILLER. For example:

```
01  first-record.
    05 record-code      PIC 99.
    05 record-type      PIC XX.
77  temp-1             PIC X(4).
77  temp-2             PIC X(3).
```

Various examples of level 77 items appear in Section 6.

Format 2 of Figure 5-3 describes a level 66 entry, which renames one or more contiguous elementary items. In the following example, the group *card-codes* is renamed *code:*

```
05  card-codes.
    10 store-code       PIC 9.
    10 state-code       PIC 9(4).
66  code RENAMES card-codes.
```

Format 3 of Figure 5-3 describes a level 88 entry, which assigns condition-name values. In the following example, item *tax-code* is defined with a range of values:

```
05  tax-code           PIC 99.
    88  tax-range       VALUES ARE 01 THRU 20.
```

JUSTIFIED Clause

The JUSTIFIED clause causes nonstandard positioning of data within a receiving item. The clause can only appear in the data description of an elementary item; the clause cannot be used for a data item that is described as numeric.

The syntax of the JUSTIFIED clause is:

```
{JUST     }  RIGHT
{JUSTIFIED}
```

When a receiving data item is described with the JUSTIFIED clause, the standard alignment rules do not apply. If a sending item is too big for the receiving item, the sending item is truncated on the left. If the sending item is smaller than the receiving item, the rightmost character of the sending item is aligned with the rightmost character of the receiving field and the value is extended to the left with space characters.

When the JUSTIFIED clause is omitted, standard alignment rules dictate that alignment is left justified and truncation or padding, when necessary, occurs on the right.

NOTE

The JUSTIFIED clause is ignored when initializing an item with the literal given in a VALUE clause.

OCCURS Clause

The OCCURS clause defines tables and other sets of repeating items, thus eliminating the need for separate item entries. These tables can be a fixed number of elements or can vary within given limits. An OCCURS clause is illegal in an 01 level.

The syntax of the OCCURS clause is:

Format 1 (fixed length table)

```
OCCURS max [ TIMES ]
```

where

max

is an integer that represents the number of elements in the table.

Format 2 (variable length table)

```
OCCURS min TO max [ TIMES ] DEPENDING [ ON ] depend
```

where

min

is an integer that represents the smallest number of elements in the table at any time. The integer must be greater than or equal to zero, and less than or equal to *max*.

max

is an integer that represents the greatest number of elements the table can have at any time.

depend

is an integer data item that controls the size of the table. As the value of the depend item increases or decreases, the number of elements in the table increases or decreases. When the table size decreases, those elements beyond the new depend limit are lost even if the next statement increases the table to include them. When the table size increases, you must assign values to the new elements before using them.

The following example illustrates the OCCURS clause:

```
01 table-group.
   02  activity-count                    PIC 99.
   02  activity-table OCCURS 10 TO 20 TIMES
              DEPENDING ON activity-count.
       05  activity-entry                PIC 999.
```

When using the data name that represents a table item, you must use subscripts to access the item. You can use the data name without subscripts only when you want the entire table (for example, in a MOVE statement). If the data name is a group item, you must use subscripts for all items belonging to the group whenever they are used as operands. Subordinate data names used as objects of a REDEFINES clause are not considered operands and, therefore, cannot be subscripted.

A data description entry with an OCCURS DEPENDING ON clause can only be followed, within its data description, by descriptions of subordinate items. In other words, only one table with a variable number of occurrences can appear in a single data description, and the data items contained by the table must be the last data items in the data description.

Data items subordinate to an entry described with an OCCURS clause can themselves contain an OCCURS clause. Tables can consist of such multiple occurrences of subordinate tables for a maximum of three levels. A data description entry containing either format of the OCCURS clause can be followed by subordinate entries containing a fixed length table OCCURS clause; however, a data description entry with an OCCURS DEPENDING ON cannot be subordinate to a group entry described with either format of the OCCURS clause.

PICTURE Clause

The PICTURE clause defines the characteristics of an elementary item.

The syntax of the PICTURE clause is:

```
{PIC    }  [ IS ] character-string
{PICTURE}
```

where

character-string

is one or more symbols that determine the category of an elementary item and place restrictions on values assignable to the item.

A maximum of 30 characters is allowed in *character-string.* When the same PICTURE character repeats, you can write it once followed by an unsigned integer enclosed in parentheses. The integer indicates how many times that character is repeated. For example:

```
PIC 9(5) is equivalent to PIC 99999.
```

Although only 30 characters can make up a character string, you can use the repetition technique to define items longer than 30 characters.

The character-string symbols that are defined in the following paragraphs are used to describe a data item.

PICTURE CHARACTER-STRING SYMBOLS. Each symbol that is used to describe a data item has a specific function. The symbols are as follows:

A represents a character position for a letter of the alphabet or a space character. The symbol is counted in the size of the data item.

P indicates scaling when the decimal point is not among or adjacent to the digits of the data item stored. The symbol is counted in determining the maximum number of digit positions in numeric items (the maximum is 18). One or more P symbols can appear only as a contiguous string to the left or right of all other digit positions in the PICTURE string. Since P implies an assumed decimal point, the P symbol is redundant when used with the V symbol.

If an operation involves conversion of data from one form of internal representation to another and the data item being converted is described with the P symbol, each digit position described by a P is considered to have the value zero; the size of the data item includes those digit positions.

S represents a signed numeric value. The symbol is counted in the size of the item only if a SIGN IS SEPARATE clause is used.

V represents the decimal point location in noninteger numeric items. The symbol is not counted in the size of the item.

X represents a character position that can have any character from the ASCII character set. The symbol is counted in the size of the item.

9 represents a character position for a digit. The symbol is counted in the size of the item.

ITEM SIZE. The size of a data item is determined by the symbols in its PICTURE string. Each A, X, and 9 is counted as one character position. An S is counted as one character only if the item is subject to a SIGN IS SEPARATE clause.

If a data item is described as DISPLAY in a USAGE clause, the size of the item includes the PICTURE string symbols. If the item is described as COMPUTATIONAL, the size of the item is computed differently, as described under the USAGE clause.

CATEGORIES OF DATA. The PICTURE clause can describe three categories of data: alphabetic, numeric, and alphanumeric. The results of most statements in the Procedure Division depend on the categories of the data items. Some statements require certain categories for some or all of their operands. In some cases, a statement can take different actions depending on the category of the data items.

In the discussion that follows, 9 and A symbols within the PICTURE string are described as representing character positions that have only numbers or letters and spaces. For reasons of efficiency, the SCREEN COBOL compiler does not always enforce this restriction. Characters other than those permitted can be moved into these positions if they appear in the corresponding group positions of a sending data item. SCREEN COBOL considers every group item to be alphanumeric. Manipulations on group items ignore all PICTURE strings. For example, a move operation into a group item can cause any position of an item to contain any ASCII character.

Alphabetic Data. An alphabetic data item can have only A symbols in the PICTURE string. The contents of this type of item are represented externally as some combination of the 26 letters of the alphabet and the space character.

The following examples illustrate alphabetic data:

```
05  package-code     PIC AAA.

05  dept-id          PIC AA(6)AA.

05  dept-code        PIC AA(2)AA.
```

Numeric Data. A numeric data item can have 9, P, S, and V symbols in the PICTURE string. The number of digits described must be greater than zero and not more than 18. The contents of this type of item are represented externally as a combination of digits 0 through 9.

If the item is signed, a plus or minus is included when the data is moved to a screen item, or when a SIGN IS SEPARATE clause is specified. In all other instances, the sign is encoded within one of the digits.

The following examples illustrate numeric data:

```
05  division-total       PIC S9(10)V99.

05  fraction-amount      PIC PP99.
```

Alphanumeric Data. An alphanumeric data item can have combinations of A, X, and 9 symbols in the PICTURE string, but the item is treated as if the string contained all X symbols. The contents of the item can be any combination of ASCII characters. A PICTURE string of all A symbols or all 9 symbols is not an alphanumeric item.

The following examples illustrate alphanumeric data:

```
10  stock-item-name       PIC X(25).

10  zone-id               PIC A(4)99.
```

REDEFINES Clause

The REDEFINES clause allows the same computer storage area to be described in more than one way. This capability is valuable for such tasks as input data validation when tests require different descriptions of the data. This capability is convenient when some portions of a record are constant, while other portions vary.

The syntax of the REDEFINES clause is:

```
REDEFINES data-name-2
```

where

data-name-2

is the data item being redefined.

The REDEFINES entry must immediately follow the entry for the data item being redefined or must immediately follow the last item subordinate to that data item. The level number of the REDEFINES entry must be the same as the item being redefined by the clause.

The following rules apply to the REDEFINES clause:

- Level numbers 66 and 88 cannot be redefined.
- The redefined data item cannot have an OCCURS clause or REDEFINES clause.
- The data name of the redefined item cannot be subscripted or qualified.
- Neither the original definition nor the redefinition can include an item whose size is variable due to an OCCURS clause of a subordinate entry.
- A VALUE clause cannot be included.
- When the level number is not 01, the redefinition cannot be greater than the number of character positions (bytes) in the data item you are redefining.
- The redefined item can be subordinate to an item with an OCCURS clause or a REDEFINES clause.
- The REDEFINES entry can be followed by subordinate data entries. Redefinition continues until the appearance of a level number less than or equal to that of the data name being redefined or until the ending of the current section of the Data Division.

The REDEFINES clause redefines a storage area, not the data items occupying the area. Multiple redefinition of the same area is permitted, but all definitions must begin with a REDEFINES clause containing the data name of the entry that originally defined the area.

The following example illustrates the REDEFINES clause:

```
WORKING-STORAGE SECTION.

01  record-in.
    05  record-code          PIC 9.
    05  record-detail        PIC X(30).
    05  record-subtotal      PIC 9(3)V99.
01  record-total REDEFINES record-in.
    05  total-1              PIC 9(5)V99.
    05  total-2              PIC 9(5)V99.
    05  total-3              PIC 9(5)V99.
    05  total-4              PIC 9(5)V99.
    05  total-5              PIC 9(6)V99.
    05  total-5-sub REDEFINES total-5 PIC X(8).
```

RENAMES Clause

The RENAMES clause assigns a new data name to one or more contiguous elementary items within a data description. RENAMES does not cause any allocation of storage. The clause can only be used with a level 66 entry.

The syntax of the RENAMES clause is:

```
66 new-name RENAMES old-name   [ {THROUGH} end-name ] .
                               [ {THRU   }          ]
```

where

new-name

is the new name for a group item or elementary item.

old-name

is a group item, an elementary item, or the first of several items to be given a new name.

end-name

is the last group item or elementary item to be included in the new name.

The RENAMES clause merely renames a group of existing data items and does not redescribe any of their characteristics; therefore, no other clauses can be used. One or more RENAMES entries can be written for a structure; these entries can occur in any order, but must immediately follow the last data description entry of the structure.

When the THROUGH option is not specified, *new-name* merely renames *old-name*. *New-name* is a group item only if *old-name* is a group item.

When the THROUGH option is specified, the following rules apply:

- *Old-name* and *end-name* must be data areas within the same structure.
- *Old-name* and *end-name* cannot have the same names, but the names can be qualified.
- *Old-name* and *end-name* cannot be the names of data entries with level number 01, 77, 66, or 88.
- *Old-name* and *end-name* cannot be described by an OCCURS clause in their definitions, and they cannot be subordinate to an item described by an OCCURS clause.
- *End-name* cannot name an item that occupies character positions preceding the beginning of the area described by *old-name*.
- *End-name* cannot name an item that is subordinate to *old-name*.
- Items within the renamed area cannot be described by an OCCURS clause.

When the THROUGH option is specified, *new-name* is a group item that includes all elementary items within the bounds established by *old-name* and *end-name*. The following defines the beginning and end of the group:

- If *old-name* is an elementary item, the new group item begins with *old-name*.
- If *old-name* is a group item, the new group item begins with the first elementary item of *old-name*.
- If *end-name* is an elementary item, the new group item ends with *end-name*.
- If *end-name* is a group item, the new group item ends with the last elementary item of *end-name*.

The following example illustrates the RENAMES clause:

```
05  card-codes.
    10  store-code          PIC 9.
    10  state-code          PIC 9(4).
05  account-number          PIC 9(6).
05  check-digit             PIC 9.
66  card-number RENAMES card-codes THRU check-digit.
```

SIGN Clause

The SIGN clause specifies the position and mode of an operational sign for a numeric data item. The clause can only be used for items that are described as DISPLAY in a USAGE clause and have an S symbol in the PICTURE string.

The syntax of the SIGN clause is:

```
SIGN [ IS ]  {LEADING }  [ SEPARATE [ CHARACTER ] ]
             {TRAILING}
```

where

`LEADING`

indicates the sign is at the beginning of the item.

`TRAILING`

indicates the sign is at the end of the item.

`SEPARATE [ CHARACTER ]`

specifies the sign becomes a separate character and is counted in the size of the item. A + for positive and a − for negative is placed at the beginning or end of the item value.

If this phrase is omitted, the sign is at the end of the item and is not counted in the size of the item.

The following example illustrates the SIGN clause:

```
05  WS-subtotal-value       PIC S9(02) SIGN IS TRAILING SEPARATE.
```

SYNCHRONIZED Clause

The SYNCHRONIZED clause forces alignment of an elementary item on the most natural computer storage boundary.

The syntax of the SYNCHRONIZED clause is:

```
{ SYNC         }   [ RIGHT ]
{ SYNCHRONIZED }   [ LEFT  ]
```

where

RIGHT and LEFT

have no effect in SCREEN COBOL.

A VALUE clause must not appear for any group item that has a subordinate item described with the SYNCHRONIZED clause.

In most cases the alignment supplied automatically by the compiler is also the most natural; however, use of the SYNCHRONIZED clause has an effect in a few special cases. Alignment considerations are as follows:

- Alignment requirements can cause SCREEN COBOL to generate implicit FILLER data. The existence of this generated data must be accounted for in certain situations.

- DISPLAY items are composed of one or more character positions and are stored as an equal number of 8-bit bytes. The byte boundary is also their natural storage boundary; therefore, the SYNCHRONIZED clause has no effect on DISPLAY item alignment.

- COMPUTATIONAL items are stored as an even multiple of bytes. Their most natural storage unit is some multiple of the 16-bit computer word, each of which contains two bytes. The SCREEN COBOL compiler automatically aligns COMPUTATIONAL items to word boundaries. This is also the natural boundary for small COMPUTATIONAL items (those items with PICTURES containing up to four 9s).

- Larger COMPUTATIONAL items (those items with PICTURES containing five or more 9s) are naturally stored as one or two 32-bit double-words. The SYNCHRONIZED clause is effective for these items because it forces their alignment on a double-word boundary.

- All level 01 and level 77 items in the Working-Storage Section and Linkage Section are automatically allocated by the SCREEN COBOL compiler so they begin on a word boundary. The compiler logic treats these items as simultaneously beginning on a byte, word, and double-word boundary. Thus, each level 01 and level 77 Working-Storage Section or Linkage Section item is inherently aligned to its most natural storage boundary.

- Words begin on two-byte boundaries and double-words begin on four-byte boundaries. Alignment, either automatic or as requested by use of the SYNCHRONIZED clause, generates implicit FILLER data in some cases.

  — If an odd number of character positions precedes a word-aligned item within a record, the compiler inserts one character position (byte) of FILLER before the item to complete allocation of the preceding word.

  — If the number of character positions preceding a double-word aligned item within a record is not a multiple of four, the compiler inserts the amount of FILLER (1, 2, or 3 bytes) needed to complete allocation of the preceding double-word. These extra bytes are not part of the data item.

  — If a group item contains two items separated by implicit FILLER bytes, these bytes are a part of that group item. However, a group item always begins with the first character position of its first elementary item, ignoring any implicit FILLER bytes that were generated to properly align that item. Thus, the initial character positions of a group item are never implicit FILLER.

- Special considerations apply when aligning an elementary data item that is described with an OCCURS clause, is subordinate to a group item described with an OCCURS clause, or both. In these cases all occurrences of the data item must be aligned uniformly.

  — The first occurrence of the item is aligned to the required storage boundary (if the elementary item also begins a containing table's first occurrence, that table's first occurrence is defined to begin at the first character position of the item). When the aligned item is itself a table, the first occurrence will end on the appropriate storage boundary (byte, word, double-word) and the remaining occurrences follow without additional FILLER bytes.

  — When the aligned item (or table of aligned items) belongs to a higher level table, further adjustment might be necessary. If the elementary item is word-aligned and the containing group occurrence consists of an odd number of character positions, the compiler inserts one byte of FILLER after each group occurrence. If the item is double-word aligned and the size of the containing group occurrence is not some multiple of four, the compiler inserts the appropriate amount of FILLER (1, 2, or 3 bytes) after each group occurrence. In all cases, inserted bytes are not part of the containing occurrences themselves, but are included in group items that contain the complete table. The preceding sequence is repeated for each higher level table.

The following example illustrates alignment as it applies to multiple OCCURS clauses:

```
01  master.
    02  table-1 OCCURS 5 TIMES.
        03  table-2 OCCURS 5 TIMES.
            04  table-3  OCCURS 5 TIMES.
                05  item-a        PIC 999 COMPUTATIONAL.
                05  item-b     PIC X.
            04  item-3         PIC X.
        03  item-2             PIC X.
```

Master appears to occupy this many bytes:

(((2+1) * 5+1) * 5+1) * 5 = 405 bytes

but it actually occupies

(((2+1+1) * 5+1+1) * 5+1+1) * 5 = 560 bytes

due to the alignment requirement for the COMPUTATIONAL item.

Implicit FILLER bytes must be accounted for in several situations. These bytes are counted when determining the size of group items that contain them. Thus, when a data item contains implicit FILLER bytes, the character positions of the bytes are included in the allocation requirements of the item. Also, implicit FILLER bytes must be included among the character positions redefined if a containing group item appears as the object of a REDEFINES clause.

Automatic alignment or requested alignment of data items described by redefinition of character positions (through use of the REDEFINES clause) follows the rules described in the preceding paragraphs. However, when the first data item allocated by a redefinition requires word or double-word alignment, the data item being redefined must begin on the appropriate boundary. In other words, SCREEN COBOL does not permit redefinitions that require insertion of implicit FILLER bytes before the first data item of the redefinition. Any bytes inserted at other places within the redefinition are counted when determining the redefinition size.

USAGE Clause

The USAGE clause defines how a data item is stored within the Tandem system, and normally affects the number of character positions used. The USAGE clause does not restrict how the item is used; however, some statements in the Procedure Division require certain usages for their operands.

The syntax of the USAGE clause is:

```
[ USAGE [ IS ] ]  { COMP         }
                  { COMPUTATIONAL }
                  { DISPLAY       }
```

where

`COMP or COMPUTATIONAL`

indicates a numeric data item that is suitable for computations.

`DISPLAY`

indicates a data item value that is stored in the standard data format as a sequence of ASCII characters.

If the clause is omitted, the default is *DISPLAY*.

A USAGE clause can be written at any level. A USAGE clause written at the group level applies to each elementary item in the group. The usage of an elementary item cannot contradict the USAGE clause of a group to which the item belongs. Note, however, that a group item is always considered to be alphanumeric by SCREEN COBOL; thus, the USAGE clause of a group item might not always apply to the manipulation of the item.

A COMPUTATIONAL item has a value suitable for computations and, therefore, must be numeric. The PICTURE string of the item can have only the symbols 9, S, V, and P. Two to eight bytes are selected for a COMPUTATIONAL item, depending on the number of 9 symbols in the PICTURE as follows:

Number of 9 symbols	Size of Data Item
1 to 4	2 bytes
5 to 9	4 bytes
10 to 18	8 bytes

Declaration of a group item as COMPUTATIONAL implies that all subordinate items in the group are COMPUTATIONAL. The group item itself cannot be used in computations.

A DISPLAY item has a value that is stored in the standard data format as a sequence of ASCII characters. The characteristics of the item are given in the PICTURE string.

If the PICTURE string of a numeric item contains an S symbol, the item has an operational sign. If a SIGN IS SEPARATE clause is not specified, the operational sign is maintained as part of either the leading or trailing digit; the affected character position will contain a non-digit ASCII character.

VALUE Clause

A VALUE clause specifies an initial value of a Working-Storage item or the value of a level 88 condition-name.

The syntax of the VALUE clause is:

Format 1 (data initialization)

```
VALUE [ IS ] literal
```

where

literal

is the initial value to be assigned to a data item. The value can be a figurative constant.

→

Format 2 (condition name entries)

```
88 condition-name     { VALUE [ IS ]  }
                      { VALUES [ ARE ] }

  { value-1 [ { THROUGH } value-2 ] }  ,...
  {           { THRU    }           }
```

where

condition-name

is the name of the condition value.

value-1

is either a single literal value or the first of a range of literal values tested by the condition.

value-2

is the final literal value in a range of literal values tested by the condition. The value must be greater than *value-1.*

VALUE CLAUSE FOR DATA INITIALIZATION. Format 1 of the VALUE clause is used to assign an initial value to a Working-Storage item at the time the program is entered. The VALUE clause must not conflict with other clauses in the data description of an item, or in the data descriptions of other items within the hierarchy. The following rules apply:

- If an item is numeric, all literals of the VALUE clause must be numeric and must be in the range of values set by the PICTURE string. Truncation of nonzero digits is not allowed. A signed numeric literal only applies to a signed numeric PICTURE. Initialization follows standard alignment rules.

- If an item is nonnumeric, all literals of the VALUE clause must be nonnumeric and must not exceed the size of the PICTURE. JUSTIFIED clauses are ignored.

- The VALUE clause is not permitted in a data description entry that meets the following criteria:

  — The entry contains an OCCURS or REDEFINES clause.

  — The entry is subordinate to an entry containing an OCCURS or REDEFINES clause.

  — The entry has a variable size due to an OCCURS clause in a subordinate entry.

- If the VALUE clause is used for initialization at the group level, the literal must be a figurative constant or a nonnumeric literal. The group area is initialized without consideration for the individual elementary or other group items within this group. Thus, the group should not have items with descriptions that include JUSTIFIED or USAGE IS COMPUTATIONAL clauses. A VALUE clause cannot appear at the subordinate levels within this group.

The following example illustrates the VALUE clause used for data initialization:

```
WORKING-STORAGE SECTION.

01  main-heading.
    05  FILLER        PIC XX      VALUE SPACES.
    05  FILLER        PIC X(8)    VALUE "DIVISION".
    05  FILLER        PIC XX      VALUE SPACES.
    05  FILLER        PIC X(6)    VALUE "REGION".
        .
        .
        .
01  counters.
    05  no-of-reads   PIC 9(5)    VALUE ZEROS.
    05  no-of-writes  PIC 9(5)    VALUE ZEROS.
```

VALUE CLAUSE FOR CONDITION-NAME ENTRIES. Format 2 of the VALUE clause is used with condition-name entries. A data item assigned in the Data Division using the special level number 88 is a condition-name; the item under which the 88 appears is the condition variable. A value or a range of values can be defined within this variable for testing. Each entry under a condition variable includes a condition-name with a VALUE clause specifying a value or a range of values for that condition-name.

All condition-name entries for a particular condition variable must immediately follow the entry describing that variable. A condition-name can be associated with any data description entry, even if specified as FILLER, with the following exceptions:

- A condition-name cannot be associated with a level 66 or 77 item.
- A condition-name cannot be associated with a group item with a JUSTIFIED or USAGE IS COMPUTATIONAL clause.

A single value, several values, or a range of values can be given for a condition-name entry.

The following example illustrates single values for condition-names:

```
05  return-code      PIC 99.                    condition variable
    88  end-of-file           VALUE 01.
    88  error-on-read         VALUE 02.        condition-names
    88  permanent-error       VALUE 03.
    88  error-on-write        VALUE 04.
```

A statement using one of these condition-names might look like this:

```
IF end-of-file,
    PERFORM end-up-routine.
```

The following example illustrates a range of values for a condition-name:

```
05  tax-code               PIC 99.
    88  tax-range      VALUES ARE 00, 03, 07 THROUGH 11.
```

A statement testing *tax-code* for being in the range of 00, 03, 07, 08, 09, 10, 11 might look like this:

```
IF NOT tax-range
     PERFORM tax-error-routine.
```

SCREEN DESCRIPTION ENTRY

A screen description entry declares the characteristics of a screen format. The entry is used in the Screen Section of the SCREEN COBOL program.

A screen can be composed of any combination of literal fields, input fields, output fields, input-output fields, and overlay areas. Each of these items can be combined into logically related groups. A group declaration provides easy reference to related fields, but it is not required.

The two types of screens are: base and overlay.

- Base screen—A base screen can be displayed independently. This type of screen can contain overlay areas upon which overlay screens can be displayed.
- Overlay—An overlay screen is displayed in an overlay area of a base screen. This allows a base screen (with, for example, a constant header section) to be used with various overlay screens.

The structure of the screen description entry is similar to a data description entry. The screen description entry is a series of declarative sentences, each beginning with a level number to indicate the hierarchy. A higher number indicates that the entry is subordinate to the previous entry. The 01 level is the highest statement in the paragraph. Subordinate entry levels can be any number from 02 through 49.

A skeleton of the screen description entry is shown in Figure 5-4.

```
SCREEN SECTION.

01 base-screen-name [ BASE ] [ SIZE clause ]
   [ input-control-character clauses ]     <--- For conversational
                                                mode only.
   [ field-characteristic clauses ] ... .

   { screen group } ...
   { screen field }
   [ screen overlay area ]

[ 01 overlay-screen-name  OVERLAY  SIZE clause [ field-
       characteristic clauses ] ... .

    { screen group } ...
    { screen field }
```

Figure 5-4. Screen Description Entry Skeleton

Level 01 introduces a screen description entry. This level defines the name of the screen, a name by which the screen is known throughout the program; defines the size of the screen; and indicates whether the screen is a base or overlay screen. The intermediate levels define groups of items. The highest numbered levels define the characteristics of the screen fields.

The screen description can have the following parts: screen name, screen overlay area, screen group, and screen field. Each of these parts defines a specific attribute of the screen. The following example illustrates a screen description entry.

```
SCREEN SECTION.
01   ENTER-AMT   BASE   SIZE 12, 80.
     05 FILLER           AT 1,  12   VALUE "ORDER DETAIL ENTRY".
     05 FILLER           AT 2,  1    VALUE "CUSTOMER".
     05 FILLER           AT 4,  1    VALUE "ITEM".
     05 FILLER           AT 4,  10   VALUE "QUANTITY".
     05 LINE1-HEADER     AT 5,  1    VALUE "MENU LIST".
     05 OVER1 AREA       AT 6,  1    SIZE 10,80.
01   OVER1-SCREEN OVERLAY SIZE 10,80.
     05 LINE1-OVERLAY    AT 2,  10   VALUE "1 DISPLAY PREVIOUS ORDER".
```

The input control character clauses are available for terminals in conversational mode. These clauses define the specific input control characters to be used during execution of an ACCEPT statement. The input control character clauses, which are referenced in text and described in detail later in this section, are summarized in Figure 5-5.

```
01 screen-name  {[ BASE ] [ SIZE clause ]}
                {OVERLAY  SIZE clause    }

              [ ABORT-INPUT [ IS ] {"nonnumeric-literal"          } ]
                                   {numeric-literal               }
                                   {  [,numeric-literal ]         }
                                   {OFF                           }

              [ END-OF-INPUT [ IS ] {"nonnumeric-literal"         } ]
                                    {numeric-literal              }
                                    {  [,numeric-literal ]        }
                                    {OFF                          }

              [ FIELD-SEPARATOR [ IS ] {"nonnumeric-literal"} ]
                                       {numeric-literal     }
                                       {OFF                 }

              [ GROUP-SEPARATOR [ IS ] {"nonnumeric-literal"} ]
                                       {numeric-literal     }
                                       {OFF                 }

              [ RESTART-INPUT [ IS ] {"nonnumeric-literal"        } ]
                                     {numeric-literal             }
                                     {  [,numeric-literal ]       }
                                     {OFF                         }
```

Figure 5-5. Input Control Character Clauses

A number of field characteristic clauses are available to define the characteristics of screen fields. These clauses, which are referenced in text and described in detail later in this section, are summarized in Figure 5-6.

```
level-num  {field-name}     {[ AT ] line-spec, column-spec}
           {FILLER    }     {REDEFINES field-name-2        }

  [ ADVISORY ]

  [ FILL nonnumeric-literal ]

  [ LENGTH [ MUST BE ]  {literal-1     [{THROUGH} literal-2]  ,... }
                        {              [{THRU   }          ]       }

  [ mnemonic-name ] ...

    MUST [ BE ]   literal-1       THROUGH   literal-2      ,...
                                  THRU

  [ OCCURS  {lines-phrase [ columns-phrase ]}
            {columns-phrase [ lines-phrase ]}

    [ DEPENDING [ ON ] data-name-1 ]                          ]

  [{PIC    }  [ IS ] character-string                         ]
  [{PICTURE}                                                  ]

  [ PROMPT screen-field ]

  [ RECEIVE [ FROM ] {ALTERNATE                }              ]
                     {ALTERNATE OR TERMINAL    }              ]
                     {TERMINAL                 }              ]
                     {TERMINAL OR ALTERNATE    }              ]

  [ REDEFINES field-name-2 ]

  [ SHADOWED [ BY ] data-name-1 ]

  [{TO    }  data-name-1                                      ]
  [{FROM  }                                                   ]
  [{USING }                                                   ]

  [ UPSHIFT  [{INPUT       }]                                 ]
             [{OUTPUT      }]                                 ]
             [{I-O         }]                                 ]
             [{INPUT-OUTPUT}]                                 ]

  [ USER [ CONVERSION ] numeric-literal ]

  [ VALUE nonnumeric-literal ]

  [ WHEN  {ABSENT}    {CLEAR}  ]
          {BLANK }    {SKIP }  ]

  [ [ WHEN ] FULL  {TAB }      ]
                   {LOCK}      ]
```

Figure 5-6. Screen Field Characteristic Clauses

Base Screen

A base screen is a screen that is initially displayed on the terminal and is used to establish the current screen for each program unit. In contrast to an overlay screen that is displayed in the overlay area of a base screen, the base screen can be displayed independently.

The base screen syntax is:

```
01 screen-name [ BASE ] [ SIZE lines, cols ]

 [ field-characteristic-clause ] ...
```

where

`screen-name`

is the name given to the base screen.

`SIZE lines, cols`

indicates the size of the screen. The number of lines and columns can each range from 1 through 255. The size can be no larger than the physical limits of the terminal screen for base screens.

If this option is omitted, the default is 24 lines, 80 columns.

`field-characteristic-clause`

is one or more clauses that define default characteristics for all fields subordinate to the screen unless these characteristics are explicitly overridden for a particular group or field. The clauses that can appear here are:

FILL
mnemonic-name
UPSHIFT
USER CONVERSION
WHEN ABSENT
WHEN BLANK
WHEN FULL

Screen Overlay Area

A screen overlay area defines an area of a base screen within which an overlay screen can be displayed. When overlay screens are used in a program, a screen overlay area must be defined in the base screen description entry.

The screen overlay area syntax is:

```
level-no area-name  AREA AT line, col  SIZE lines, cols
```

where

`level-num`

is a numeric literal that indicates the hierarchy. The value must be within the range of 2 through 49. Subordinate entries are not allowed.

`area-name`

is the name given to the screen overlay area.

`AREA AT line, col`

specifies the position of the upper left-hand corner of the area relative to the boundaries of the screen.

`SIZE lines, cols`

determines the number of lines and columns included in the area. The entire area must lie within the boundaries of the base screen, and no fields can overlap the area.

For T16-6510 terminals, the *cols* value must be the same as the number of columns declared for the base screen.

Overlay Screen

An overlay screen is a screen that is displayed in an overlay area of a base screen.

The overlay screen syntax is:

```
01 screen-name  OVERLAY  SIZE lines, cols

 [ field-characteristic-clause ] ...
```

where

`screen-name`

is the name given to the overlay screen.

`SIZE lines, cols`

indicates the size of the overlay screen. The size can be no larger than the size of the overlay area into which it is to be placed. For the T16-6510, the width must be exactly the same as the base screen.

`field-characteristic-clause`

is a clause that defines default characteristics for all fields subordinate to the screen unless explicitly overridden for a particular group or field. The clauses that can appear here are:

FILL
mnemonic-name
UPSHIFT
USER CONVERSION
WHEN ABSENT
WHEN BLANK
WHEN FULL

Screen Group

A screen group is a combination of fields that are grouped together to provide collective references to the subordinate fields and to define the common characteristics of the fields. A screen group can contain subordinate groups.

The screen group syntax is:

```
level-num  {group-name}    [[ AT ] line, column]
           {FILLER    }    [                    ]

 [ field-characteristic-clause ] ...

   {screen-field}  ...
   {screen-group}
```

where

`level-num`

is a numeric literal that indicates the hierarchy. The value must be within the range of 2 through 48.

`group-name`

is the name given to the group.

`FILLER`

is a keyword that takes the place of *group-name.*

`AT line, column`

specifies the home position of the group relative to the boundaries of the screen. The line number and column number must be within the size specified for the screen. The positions of subordinate fields can be given relative to the home position; this allows you to move groups easily.

If this clause is omitted, group relative addressing is not allowed in the group.

`field-characteristic clause`

is one or more clauses that define default characteristics for all fields subordinate to the group unless these characteristics are explicitly overridden for a particular field. The clauses that can appear here are:

FILL
mnemonic-name
UPSHIFT
USER CONVERSION
WHEN ABSENT
WHEN BLANK
WHEN FULL

Screen Field

A screen field is a single elementary item.

The screen field syntax is:

```
level-num  { field-name }  [ field-characteristic-clause ]... .
           { FILLER     }
```

where

`level-num`

is a numeric literal within the range of 2 through 49 that indicates the hierarchy.

`field-name`

is the name given to the field.

`FILLER`

is a keyword that takes the place of *field-name*. FILLER must be used for a literal field.

`field-characteristic-clause`

is one or more clauses that define a characteristic of the field. The clauses that can appear here depend on the field type.

The four types of screen fields are determined by the data association clauses TO, FROM, and USING. Screen field types and the clauses that can be used with each are listed in Table 5-1.

Table 5-1. Screen Types and Allowable Field Characteristic Clauses

Screen Type	Determined By	Required Clauses	Optional Clauses
Literal	No TO, FROM, or USING clause	AT VALUE	mnemonic-name
Input	TO clause only	AT or REDEFINES PICTURE	FILL LENGTH mnemonic-name MUST BE OCCURS RECEIVE SHADOWED UPSHIFT USER CONVERSION VALUE WHEN ABSENT WHEN BLANK WHEN FULL
Output	FROM clause only	AT or REDEFINES PICTURE	ADVISORY FILL mnemonic-name OCCURS SHADOWED UPSHIFT USER CONVERSION VALUE
Input-Output	USING clause or TO and FROM clause	AT or REDEFINES PICTURE	ADVISORY FILL LENGTH mnemonic-name MUST BE OCCURS RECEIVE SHADOWED UPSHIFT USER CONVERSION VALUE WHEN ABSENT WHEN BLANK WHEN FULL

Input Control Character Clauses

Input control character clauses are for terminals operating in conversational mode. These clauses define the characters used during the execution of an ACCEPT statement to perform the following:

- delimit a screen field or a group of screen fields described with an OCCURS clause
- terminate or abort the processing of an ACCEPT statement
- restart the processing of an ACCEPT statement

These clauses, which are recognized only by terminals in conversational mode, are described in the following paragraphs.

ABORT-INPUT CLAUSE. The ABORT-INPUT clause defines the characters used to terminate the processing of the current ACCEPT statement with an abort termination status. The ABORT-INPUT clause is recognized only by terminals operating in conversational mode.

The syntax of the ABORT-INPUT clause is:

```
ABORT-INPUT [ IS ]  { "nonnumeric-literal"                          }
                    { numeric-literal [, numeric-literal ]          }
                    { OFF                                           }
```

where

"nonnumeric-literal"

is one or two alphanumeric characters enclosed in quotation marks.

numeric-literal

is one or two integers. Each integer must be within the range of 0 through 255. *numeric-literal* is the decimal value of an 8-bit binary number.

If a process is responding in place of a terminal, SCREEN COBOL interprets the 8-bit pattern (two numeric literals convert to a 16-bit pattern) as a non-keyboard character.

OFF

specifies that ABORT-INPUT is not available for the current screen.

If this clause is omitted, the abort input characters are @@.

If used, the ABORT-INPUT clause must be specified at the 01 screen level. A character defined for ABORT-INPUT cannot be specified for another input control character.

If the abort input character is entered during an ACCEPT statement no values in the Working-Storage Section are changed by that ACCEPT statement.

END-OF-INPUT CLAUSE. The END-OF-INPUT clause defines the characters used to indicate the end of the last input field for the current ACCEPT statement. The END-OF-INPUT clause is recognized only by terminals operating in conversational mode.

The syntax of the END-OF-INPUT clause is:

```
END-OF-INPUT [ IS ]  { "nonnumeric-literal"                          }
                     { numeric-literal [, numeric-literal ]          }
                     { OFF                                           }
```

where

"nonnumeric-literal"

is one or two alphanumeric characters enclosed in quotation marks.

numeric-literal

is one or two integers. Each integer must be within the range of 0 through 255. *numeric-literal* is the decimal value of an 8-bit binary number.

If a process is responding in place of a terminal, SCREEN COBOL interprets the 8-bit pattern (two numeric literals convert to a 16-bit pattern) as a non-keyboard character.

OFF

specifies END-OF-INPUT is not available for the current screen.

If this clause is omitted, the end of input characters are //.

If used, the END-OF-INPUT clause must be specified at the 01 screen level. A character defined for END-OF-INPUT cannot be specified for another input control character.

FIELD-SEPARATOR CLAUSE. The FIELD-SEPARATOR clause defines the character used to separate one screen field from another during an ACCEPT statement. If a screen field description includes an OCCURS clause, each occurence is treated as one field. The FIELD-SEPARATOR clause is recognized only by terminals operating in conversational mode.

The syntax of the FIELD-SEPARATOR clause is:

```
FIELD-SEPARATOR [ IS ]  {"nonnumeric-literal"}
                        { numeric-literal    }
                        { OFF                }
```

where

"nonnumeric-literal"

is one alphanumeric character enclosed in quotation marks.

numeric-literal

is one integer that must be within the range of 0 through 255. *numeric-literal* is the decimal value of an 8-bit binary number.

If a process is responding in place of a terminal, SCREEN COBOL interprets the 8-bit pattern as a non-keyboard character.

OFF

specifies that FIELD-SEPARATOR is not available for the current screen.

If this clause is omitted, the field separator character is a comma (,).

If used, the FIELD-SEPARATOR clause must be specified at the 01 screen level. The character defined for FIELD-SEPARATOR cannot be specified for another input control character.

In the following example, the FIELD-SEPARATOR clause defines S as the keyboard character to be used.

```
SCREEN SECTION.
01  EMP-RECORD-SCREEN    BASE  SIZE 24, 80
                                FIELD-SEPARATOR IS "S" .
```

GROUP-SEPARATOR CLAUSE. The GROUP-SEPARATOR clause defines the character used during the processing of an ACCEPT statement to indicate the following:

- the last item in an OCCURS clause
- the end of a field, if the field preceding the group separator has no multiple occurences.

The GROUP-SEPARATOR clause is recognized only by terminals operating in conversational mode.

The syntax of the GROUP-SEPARATOR clause is:

```
GROUP-SEPARATOR [ IS ]  { "nonnumeric-literal" }
                        { numeric-literal      }
                        { OFF                  }
```

where

"nonnumeric-literal"

is one alphanumeric character enclosed in quotation marks.

numeric-literal

is one integer that must be within the range of of 0 through 255. *numeric-literal* is the decimal value of an 8-bit binary number.

If a process is responding in place of a terminal, SCREEN COBOL interprets the 8-bit pattern as a non-keyboard character.

OFF

specifies that GROUP-SEPARATOR is not available for the current screen.

If this clause is omitted, the group separator character is a semicolon (;).

If used, the GROUP-SEPARATOR clause must be specified at the 01 screen level. The character defined for GROUP-SEPARATOR cannot be specified for another input control character.

RESTART-INPUT CLAUSE. The RESTART-INPUT clause defines the characters used to restart input processing during the current ACCEPT statement. The RESTART-INPUT clause is recognized only by terminals operating in conversational mode.

The syntax of the RESTART-INPUT clause is:

```
RESTART-INPUT [ IS ]  { "nonnumeric-literal"                          }
                      { numeric-literal [, numeric-literal ]          }
                      { OFF                                           }
```

where

"nonnumeric-literal"

is one or two alphanumeric characters enclosed in quotation marks.

numeric-literal

is one or two integers. Each integer must be within the range of 0 through 255. *numeric-literal* is the decimal value of an 8-bit binary number.

If a process is responding in place of a terminal, SCREEN COBOL interprets the-8 bit pattern (two numeric literals convert to a 16-bit pattern) as a non-keyboard character.

OFF

specifies that RESTART-INPUT is not available for the current screen.

If this clause is omitted, the restart input characters are !!.

If used, the RESTART-INPUT clause must be specified at the 01 screen level. A character defined for RESTART-INPUT cannot be specified for another input control character.

If the current ACCEPT statement is restarted, the data entered before the restart input characters does not change the values of the associated data items in working-storage. If data is entered on the same line following the restart input characters, the data is ignored.

The following example illustrates the input control character clauses:

```
SCREEN SECTION.

01 CUSTOMER-REC-SCREEN      BASE          SIZE 24, 80
                                         FIELD-SEPARATOR ","
* Documents the default field separator character.

                                         GROUP-SEPARATOR OFF

                                         ABORT-INPUT     "AI"
* Defines the keyboard abort input characters as AI.

                                         END-OF-INPUT    64, 64
* Defines the keyboard end of input characters as @@.

                                         RESTART-INPUT   "2" .
* Defines the keyboard restart input character as 2.
```

Field Characteristic Clauses

Field characteristic clauses specify various characteristics of screen fields. These clauses are described in the following paragraphs.

ADVISORY CLAUSE. The ADVISORY clause identifies a single output or input-output field as the one to be used for informational and error messages generated by the TCP.

The syntax of the ADVISORY clause is:

```
ADVISORY
```

Every base screen should have an advisory field. The field should be alphanumeric with a size of at least 35 characters. Error messages that appear in this field are described in Appendix A.

An overlay screen must not have an advisory field.

For terminals in conversational mode, an advisory field must be defined for the screen or the standard advisory messages will not appear on the terminal.

AT CLAUSE. The AT clause specifies the location of the field.

The syntax of the AT clause is:

```
AT line-spec, column-spec
```

where

`line-spec`

specifies the line in which the field begins.

`column-spec`

specifies the column in which the field begins.

Both *line-spec* and *column-spec* can appear in the following forms:

`numeric-literal`	This form represents the line or column relative to the beginning of the screen.
`* [+ numeric-literal]` `[- numeric-literal]`	This form represents a location relative to the current position. The current position begins at line 1, column 1 and is advanced to the first available position following a field after that field is declared.
`@ [+ numeric-literal]` `[- numeric-literal]`	This form represents a location relative to the home position of the group containing the field declaration. The home position is the first data character of the field and is specified for the group with the AT clause.

Either the AT or the REDEFINES clause must be included in every screen field declaration. If both clauses appear in the screen field declaration, they must both refer to exactly the same position.

FILL CLAUSE. The FILL clause declares a padding character for the field. When output to the field does not fill the full width specified, the padding character fills in to the right of the field.

The syntax of the FILL clause is:

```
FILL nonnumeric-literal
```

where

nonnumeric-literal

is one character long.

If this clause is omitted, the fill character is *SPACES*.

On input, the trailing FILL characters are removed from the input string before the input is analyzed for errors and converted. If a TO clause contains a numeric field, the leading and trailing FILL characters are removed before the input is processed. FILL characters embedded within a field are not removed.

LENGTH CLAUSE. The LENGTH clause specifies the acceptable number of characters that can be entered into a screen input field. The number of characters input is determined before conversion, but after the fill characters are removed.

The syntax of the LENGTH clause is:

```
LENGTH [ MUST BE ]  { literal-1  [ { THROUGH } literal-2 ] }  ,...
                                   { THRU    }
```

where

literal-1 and literal-2

are numeric values from 0 through the field size. If *literal-2* is included, its value must be greater than *literal-1*.

The maximum value allowed by the compiler is 255.

If this clause is omitted, any number of characters are allowed within the constraints of the picture.

The following example specifies that *FLD1* is optional (length can be 0), but must be five characters long if it is entered; *FLD2* is required, but 1 through 5 characters can be entered.

```
04  FLD1  AT 1, 1  TO X  PIC A9999  LENGTH  0, 5.
04  FLD2  AT 2, 1  TO Y  PIC ZZZZ9  LENGTH  1 THRU 5.
```

When a field is optional and no characters are input, the value of the associated data item is changed by the ACCEPT statement according to the WHEN ABSENT/BLANK field characteristic clause.

mnemonic-name CLAUSE. The mnemonic-name clause allows display attributes to be specified for a screen field. The mnemonic-name is associated with the attributes by a declaration in the SPECIAL-NAMES paragraph of the Environment Division.

The syntax of the mnemonic-name clause is:

```
mnemonic-name
```

The display attributes combined with the default values for unspecified attributes determine the display attributes for the field when the field is displayed initially; display attributes can be restored by a RESET statement, as described in Section 6.

The default value for the protection attribute depends on the screen field type. If the field is an input or input-output field, the default is UNPROTECTED. If the field is an output field, the default is PROTECTED.

MUST BE CLAUSE. The MUST BE clause specifies the acceptable values for an input screen field.

The syntax of the MUST BE clause is:

```
MUST [ BE ]  { literal-1  [ { THROUGH } literal-2 ] }  ,...
                           { THRU    }
```

where

literal-1 and literal-2

are numeric literals for numeric items and nonnumeric literals for alphanumeric items.

Any figurative constant except ALL can be specified.

The literals used in this clause must match for the screen field and the associated data item or an error is generated. For example, if a screen field receives alphanumeric character data, that data must go into a data item that is defined with a nonnumeric PICTURE clause.

Numeric items are compared numerically; alphanumeric items are compared left to right according to the ASCII character set. For example:

An input string 9 is less than 10 if the screen PICTURE clause is numeric.

An input string "9" is greater than "10" if the screen PICTURE clause is nonnumeric.

When the MUST BE clause is processed a numeric literal is scaled to match the PICTURE clause defined for the associated data item. For example, if a data item is defined with a PICTURE 999.99 and the value 100 is received from the terminal, the input value is scaled two places and stored into the data item as 100.00.

OCCURS CLAUSE. The OCCURS clause specifies multiple occurrences of screen fields. This clause can define a column, a row, or a rectangular array of fields. Each occurrence of the field is identical except for location, and each is associated with a particular occurrence of a Working-Storage data item having an OCCURS clause.

The syntax of the OCCURS clause is:

```
OCCURS  { lines-phrase [ columns-phrase ]   }
        { columns-phrase [ lines-phrase   ] }

 [ DEPENDING [ ON ] data-name-1 ]
```

where *columns-phrase* is

```
  IN literal-1 COLUMNS { OFFSET   } { literal-k } ,...
                       { SKIPPING }
```

where *lines-phrase* is

```
  ON literal-2 LINES [ SKIPPING literal-3 ]
```

where

`IN COLUMNS, ON LINES`

determines the number of field occurrences, the location of each field occurrence, and the ordering of the field occurrences.

`literal-1`

is a numeric literal that specifies the number of field occurrences on a line.

`literal-k`

is a numeric literal that specifies the horizontal spacing of the field columns.

When OFFSET is specified, *literal-k* is the number of spaces between the first column of a field occurrence *(literal-1)* and the first column of the next field occurrence *(literal-1* +1) on the same line.

When SKIPPING is specified, *literal-k* is the number of spaces between the last column of a field occurrence (column k) and the first column of the next field occurrence (column k+1) on the same line. There can be at most (literal-1) *−1* separations. If there are fewer separations, the last *literal-k* is used repeatedly. No separation is required after the last literal.

`literal-2`

is a numeric literal that specifies how many lines contain occurrences.

`literal-3`

is a numeric literal that specifies how many lines are skipped between each line that contains occurrences of the field. →

DEPENDING

indicates that the number of occurrences is variable.

data-name-1

is the unsubscripted name of an elementary numeric item where the current number of occurrences is defined. This item must be defined in the Working-Storage Section or Linkage Section. On input (execution of an ACCEPT statement), this item is set. On output (execution of a DISPLAY statement), this item is used to define the number of values output.

The following conventions apply to the OCCURS clause:

- When the IN phrase is omitted, a single occurrence on each line is indicated.
- The order of the phrases determines the order in which the occurrence numbers are assigned to the occurrences.
  - If the ON phrase is specified first, the occurrences are numbered sequentially from line to line down a column.
  - If the IN phrase is specified first, the occurrences across a line are numbered sequentially.
- A screen field described with an OCCURS clause and associated with a data item by a TO, FROM, or USING clause, must define the same maximum number of occurrences in the OCCURS clause as is specified in the associated data item OCCURS clause. The following example is a working storage data item associated with the screen field.

```
WORKING-STORAGE SECTION.
01 GAME-SCHE-REC.
   .
   05 TABLE-A              PIC X(8)  OCCURS 4 TIMES.
   .
SCREEN SECTION.
   .
   05 FIELD-A  AT 6, 10    PIC X(8)  USING TABLE-A
                                     OCCURS IN 4 COLUMNS SKIPPING 1.
```

  If the data item named in the TO, FROM, or USING clause has subordinate items and contains multiple OCCURS clauses, the maximum number of occurrences for each OCCURS clause must match the maximum number of occurrences specified in the corresponding screen field descriptions.

- A single screen description can have any number of variable length tables. The restriction of one per structure that applies to the Working-Storage Section and Linkage Section does not apply to screens.
- A screen field that is described with an OCCURS clause must be referenced without a subscript when the field is used as one of the screen identifiers in an ACCEPT statement. In other statements where screen identifiers can be used, a screen field that is described with an OCCURS clause can appear with or without a subscript. A reference without a subscript refers to all occurrences of the table. A reference that includes a subscript refers only to the occurrence selected by the value of the subscript.

- When a screen field described with a DEPENDING phrase is referenced in an ACCEPT statement, part of the input processing is the determination of the size of the table—the value to be stored into *data-name-1.* All occurrences of the field are examined and *data-name-1* is set to the occurrence number of the last occurrence that was entered. If the field is also a required field, all preceding occurrences of the field must also be entered. Failure to do this causes a PREVIOUS FIELD MISSING error message to be displayed for the terminal operator.

- Several tables on the same screen might have the same *data-name-1* in their DEPENDING phrase. If the tables are referenced in the same ACCEPT statement, the value of *data-name-1* is set to the maximum of the values that would be computed when considering each table separately. If this causes the value of *data-name-1* to be set greater than the highest supplied occurrence of a table whose fields are required, the input is in error and a REQUIRED FIELD MISSING or EARLIER FIELD MISSING (depending on the order of the fields) message is displayed for the terminal operator.

- When a field described with a DEPENDING phrase is referenced without a subscript in any statement other than an ACCEPT statement, the reference is to all occurrences within the current size of the table, as specified by the value in *data-name-1.*

The following example illustrates the OCCURS clause:

```
05  FLD-A    AT 6, 10  PIC X(8) FROM TBL-A
             OCCURS IN 4 COLUMNS OFFSET 10.
```

An equivalent OCCURS clause would be:

```
             OCCURS IN 4 COLUMNS SKIPPING 2.
```

PICTURE CLAUSE. The PICTURE clause defines the format in which the data appears on the terminal screen.

The syntax of the PICTURE clause is:

```
{PIC    }  [ IS ] character-string
{PICTURE}
```

where

character-string

can take the same form as described in the data description entry with the following exceptions:

The symbol S cannot appear in the picture.

Numeric edited and alphanumeric edited forms are allowed.

Generally, input from a screen field is performed in a manner that is inverse to normal editing functions implied by the picture. The input editing always correctly reconverts a value using the same picture for input and output.

The input editing process is different for the two classes of the input item:

- Alphanumeric input—Only the left-hand portion of the picture corresponding to the actual number of input characters must be matched. The remaining portion of the picture is ignored.
- Numeric input—Leading and trailing spaces and fill characters are first removed from the input data string. Then an attempt is made to match each character in the picture with a character in the input data, proceeding from right to left. If a match cannot be made, the data is considered to be in error.

Some picture symbols are special in that the positions they represent might be omitted from the input data string. Symbols that can be included in this category are Z, comma, multiple plus and minus signs, CR, DB, and multiple currency signs. If a mismatch occurs with an input character of this type, and if a space would be acceptable at that point in the input string, the data is not considered in error; the picture symbol is replaced by a space and the editing attempts to match the input character with the next picture symbol.

PICTURE Character-String Symbols. Each symbol that is used to describe a screen data item has a specific function. The symbols are as follows:

A — represents a character position for a letter of the alphabet or a space character. If the character is not a letter or a space, it is flagged as an error.

B — represents a character position where a space must occur in the input. The space is deleted during conversion into its associated data item. This character should not be used as the rightmost character of a numeric picture because trailing spaces are removed before conversion.

P — indicates an implicit decimal position (with value zero) to be used in aligning the decimal point in the numeric result. Refer to the description of the V symbol for cautions.

V — indicates the decimal point location in a numeric item in which the terminal operator will not enter an explicit decimal point. The alignment takes place from the last character entered in the field by the terminal operator. This symbol should be used with care because the variable length nature of terminal operator input could cause unintended alignments to occur. It is recommended that the LENGTH clause be used to require full length entry whenever a picture with implicit decimal places and potentially absent positions (for example, positions defined with the Z symbol) is used.

X — represents a character position that can have any character from the ASCII character set.

Z — represents a position that must be a digit or must be a space if no digits appear to the left of the symbol. The symbol is replaced by a space during editing only when it is one of a set of multiple Z symbols. A space is equivalent to a zero for purposes of conversion.

9 — represents a character position that must be a digit.

0 — represents a character position where a zero must appear. The zero is deleted during conversion into the associated data item.

/ — represents a character position where a right slant must appear. The / is deleted during conversion into the associated data item.

, — represents a character position where a comma must appear if any digits appear to the left of it. If no digits appear to the left of the symbol, the character must be a space (or other floating insertion character). The comma is deleted during conversion into the associated data item.

. represents a character position where a period must appear and indicates decimal point alignment. The period is deleted during conversion into the associated data item.

+ represents a position where either a plus or a minus sign must appear. Multiple plus signs represent positions that must contain some number of digits preceded by a single plus sign or a single minus sign, preceded by spaces. The symbol is replaced by a space during editing only when it is one of a set of multiple plus signs.

– represents a position where either a space or a minus sign must appear. Multiple minus signs represent positions that must contain some number of digits preceded by an optional minus sign, preceded by spaces. The symbol is replaced by a space during editing only when it is one of a set of multiple minus signs.

CR represents two positions that must contain the characters CR, or spaces. These symbols are replaced by spaces during editing if the value is nonnegative.

DB represents two positions that must contain the characters DB, or spaces. These symbols are replaced by spaces during editing if the value is nonnegative.

* represents a position that must be a digit or an asterisk. If the position is a digit, the digit must be to the left of all asterisks.

$ represents a position where a currency symbol must appear. Multiple currency symbols represent positions that must contain some number of digits preceded by a currency symbol, preceded by spaces. The symbol is replaced by a space during editing only when it is one of a set of multiple currency symbols.

Item Size. The size of a data item is determined by the symbols of its PICTURE string. The character-string symbols DB and CR are each counted as two character positions. Symbols V and P are not counted. All others are counted as one character position.

PROMPT CLAUSE. The PROMPT clause associates a named screen item for output with a screen field for input. During the processing of an ACCEPT statement the contents of a named screen item can be displayed (to assist the terminal operator) before the screen input is read.

The syntax of the PROMPT clause is:

```
PROMPT screen-field
```

where

screen-field

is the name of a previously defined screen field. The contents of *screen-field* can be described in the Screen Section with a VALUE clause or in a working storage data item and output with a FROM clause. The contents of *screen-field* are used as a prompt for the screen field described with the PROMPT clause.

PROMPT Clause for Block Mode. For terminals operating in block mode, a *screen-field* described in the Screen Section is displayed during the ACCEPT statement.

If a PROMPT clause is specified, the value of *screen-field* is displayed during the ACCEPT statement.

PROMPT Clause for Conversational Mode. For terminals operating in conversational mode, *screen-field* is used as a signal for input. In the Screen Section, a screen field description must precede the associated PROMPT clause in the same screen description.

During execution of the ACCEPT statement, the value specified in the prompt screen field is displayed before the terminal is able to receive input. The prompt value is always displayed in the first column of a screen line.

The following example illustrates a PROMPT clause with *screen-field* described in the Screen Section. When the associated ACCEPT statement executes, *LAST NAME* appears on the screen followed by a set of parentheses (delimiting the field size) and the cursor.

```
SCREEN SECTION .
01 ADDCUST-SCREEN  BASE  SIZE 24, 80 .
   05 NAME1-PROMPT      AT 3, 2   VALUE "LAST NAME: " .
   05 LAST-NAME-FIELD   AT 3, 13  PIC X(10) USING CUST-LAST-NAME
                                  LENGTH MUST BE 1 THRU 10
                                  PROMPT NAME1-PROMPT .
```

The next example illustrates a PROMPT clause with *screen-field* described in the Working-Storage Section and output with a FROM clause.

```
WORKING-STORAGE SECTION.
01 NEWCUST-REC.
   05 NEW-LAST-NAME             PIC X(10) VALUE SPACES.
   .
01 WS-PROMPT-VALUE              PIC X(11) VALUE "LAST NAME: ".
   .
   .
SCREEN SECTION.
01 NEWCUST-SCREEN.
   05 LAST-NAME-PROMPT  AT 3, 2   PIC X(11) FROM WS-PROMPT-VALUE.
   05 LAST-NAME-FIELD   AT 3, 13  PIC X(10) USING NEW-LAST-NAME
                                             LENGTH MUST BE 1 THRU 10
                                             PROMPT LAST-NAME-PR OMPT.
```

If the PROMPT clause is defined with a FROM or USING phrase, the value currently stored in the associated working-storage data item is displayed in parentheses following the prompt. For example, if LAST NAME (Brown) appears, Brown was the value entered during the last ACCEPT statement for this field.

If the PROMPT clause is defined with a TO phrase, the parentheses are not displayed.

RECEIVE CLAUSE. The RECEIVE clause specifies whether screen field data can be accepted from a terminal, another kind of device, or both. This option is supported only for applications running on Tandem 6530 terminals with version C00 (or later) microcode and Tandem 6AI (revision A00) firmware.

The syntax of the RECEIVE clause is:

```
RECEIVE [ FROM ] {ALTERNATE                }
                 {ALTERNATE OR TERMINAL    }
                 {TERMINAL                 }
                 {TERMINAL OR ALTERNATE    }
```

where

`ALTERNATE`

causes data to be accepted from a device other than the terminal. The other devices that PATHWAY supports are:

- optical character recognition reader
- optical bar code reader
- magnetic string reader for badges or cards

`ALTERNATE OR TERMINAL`

causes data to be accepted from one of the alternate devices listed above and from the terminal keyboard.

`TERMINAL`

causes data to be accepted only from the terminal keyboard.

`TERMINAL OR ALTERNATE`

causes data to be accepted from one of the alternate devices listed above and from the terminal keyboard.

If this clause is omitted, data can be accepted only from the terminal keyboard.

The RECEIVE clause restricts input from the terminal keyboard for screen fields defined with the ALTERNATE option. These fields can accept data only from an alternate device that is plugged into a Tandem 6530 terminal.

You can use the SCREEN COBOL TURN statement to change this attribute to a previously defined option.

An example of the RECEIVE clause is:

```
SCREEN SECTION.
01 INVENTORY-REC-SCREEN   BASE SIZE 24, 80.
   .
   .
   05 PROD-FIELD       AT 5, 28 PIC X(10)  RECEIVE FROM ALTERNATE
                                           USING WS-PROD-ID.
   05 COUNT-FIELD      AT 7, 28 PIC X(10)  RECEIVE FROM
                                           ALTERNATE OR TERMINAL
                                           TO WS-PROD-COUNT.
   .
   .
```

REDEFINES CLAUSE. The REDEFINES clause specifies that the screen field being defined is an alternate interpretation of a previously defined field.

The syntax of the REDEFINES clause is:

```
REDEFINES field-name-2
```

where

field-name-2

is the previously defined field.

The two fields must be identical in size and display attributes.

The REDEFINES clause allows an ACCEPT statement to be issued for a given physical field using different rules. An example would be postal codes in the U.S. and in the U.K.

```
05 ZIP-US AT 10, 10          PIC 999999  LENGTH 0, 5  TO ZIP-US-WS.
05 ZIP-UK REDEFINES ZIP-US   PIC XXXXXX  LENGTH 0, 6  TO ZIP-UK-WS.
```

Either the REDEFINES or the AT clause must be included in every screen field declaration. If both clauses appear in the screen field declaration, they must refer to exactly the same position.

SHADOWED CLAUSE. The SHADOWED clause associates a secondary Working-Storage data item with a nonliteral screen field. This additional field can be used to determine whether input was supplied for the screen field or to control selection of the field for output statements.

The syntax of the SHADOWED clause is:

```
SHADOWED [ BY ] data-name-1
```

where

data-name-1

is the data item to be associated with a nonliteral screen field. The size of the data item is one byte with a description of PIC X or PIC 9 COMP.

The rightmost bit of *data-name-1* is the SELECT bit for the screen field. This bit is examined by the DISPLAY, TURN, RESET, and SET NEW-CURSOR statements that include the SHADOWED modifier; when this modifier is used in the statement, a field listed in the statement will not be affected unless the SELECT bit in its SHADOWED item is set to 1.

| ... | RETURN | ENTER | SELECT |

The bit to the left of the SELECT bit is the ENTER bit. When a screen field is specified in an ACCEPT statement, a 1 or a 0 is stored into this bit. If data is present in the field, a 1 is stored in the ENTER bit. If spaces, fill characters, or nothing is entered in the field, a 0 is stored in this bit.

The bit to the left of the ENTER bit is the RETURN bit. If a shadowed field is specified in an ACCEPT statement, a 1 or a 0 is stored into this bit. If data, fill characters, or spaces are present in the field, a 1 is stored in the RETURN bit; otherwise, a 0 is stored in this bit. If operating on a T16-6510 terminal, the RETURN bit always contains a 1 for a shadowed field.

The values stored in the RETURN and ENTER bits depend on the following information received from the terminal:

- If the screen field is tabbed across (contains nothing), the values stored are:

  RETURN bit = 0 ENTER bit = 0

- If the screen field contains fill characters or spaces, the values stored are:

  RETURN bit = 1 ENTER bit = 0

- If the screen field contains normal data, the values stored are:

  RETURN bit = 1 ENTER bit = 1

If the ESCAPE clause is executed during the ACCEPT statement (for example, an abort input is specified for a terminal operating in conversational mode), the settings for the RETURN bit and the ENTER bit are undefined.

If the screen field to which the SHADOWED clause applies has an OCCURS clause, *data-name-1* given in the SHADOWED clause should be the data item having an OCCURS clause with the same maximum number of occurrences to match the occurences in the OCCURS clause of this corresponding field in the Screen Section.

An example of the SHADOWED clause is:

```
SCREEN SECTION.
01 LOCATION-REC-SCREEN    BASE SIZE 24, 80.
   .
05 STATE-FIELD          AT 5, 28  PIC X(2) USING WS-STATE
                                     SHADOWED BY WS-DATA-ITEM.
   .
```

TO, FROM, USING CLAUSES. The TO, FROM, USING clauses are collectively referred to as data association clauses. These clauses specify a Working-Storage Section or Linkage Section data item that is associated with the screen field for moving data to and from the screen field. The clauses determine the general type of a field.

The syntax of the TO, FROM, USING clauses is:

```
{TO   }
{FROM } data-name-1
{USING}
```

where

TO

specifies that data is to be moved from the screen field into the *data-name-1* area; this is an input association.

FROM

specifies that data is to be moved from the *data-name-1* area into the screen field; this is an output association.

USING

is equivalent to specifying both TO and FROM with the same data name.

data-name-1

is a working-storage data item associated with an elementary screen field; the field cannot be a subscripted item.

The following rules apply:

- A TO, FROM, and USING clause can be specified only with an elementary screen field.
- The TO and FROM clauses can both be specified for a screen field. If both clauses are specified, the data names can differ.
- If a data association clause is specified for any field, a PICTURE clause must also be specified for that field.
- The category of the screen field must be compatible with the associated data item in the Working-Storage Section or Linkage Section. A numeric edited field must be associated with a numeric data item, and an alphabetic edited field must be associated with an alphabetic or alphanumeric data item.

The data movement occurs in connection with the execution of a DISPLAY or ACCEPT statement. The statements explicitly or implicitly name the screen field containing the data association clause.

UPSHIFT CLAUSE. The UPSHIFT clause specifies that lowercase alphabetic characters are to be translated to uppercase characters for input and output.

The syntax of the UPSHIFT clause is:

```
UPSHIFT    [ INPUT          ]
           [ OUTPUT         ]
           [ INPUT-OUTPUT   ]
           [ I-O            ]
```

If UPSHIFT appears by itself, *INPUT-OUTPUT* is assumed.

If the clause is omitted, lower case alphabetic characters for the field remain in lower case.

USER CONVERSION CLAUSE. The USER CONVERSION clause gives a user-defined number to be passed with the field to a conversion procedure.

The syntax of the USER CONVERSION clause is:

```
USER [ CONVERSION ] numeric-literal
```

The USER CONVERSION clause is used only if the application makes use of a user conversion procedure. Refer to Appendix D for details regarding user conversion procedures.

VALUE CLAUSE. The VALUE clause specifies the initial value of a screen field. The initial value is displayed during a DISPLAY BASE or OVERLAY statement, during a RESET DATA statement, and during screen recovery. The VALUE clause is required for literal screen fields.

The syntax of the VALUE clause is:

```
VALUE nonnumeric-literal
```

where

nonnumeric-literal

is the character form of the specified value. The *nonnumeric-literal* must not be longer than the size specified for the field in the PICTURE clause; if it is shorter, the *nonnumeric-literal* is left justified and padded with the fill character.

The value does not have to be valid according to conversion and checking restraints for input fields. However, if the value is not valid and the value comes in from the terminal during ACCEPT statement processing, the field is in error.

The VALUE clause cannot be used for a field using the OCCURS clause.

The following example illustrates the VALUE clause:

```
SCREEN SECTION.
 01 ORD-DETAIL-SCRN  SIZE 12, 40.
    05 FILLER AT 1, 12  VALUE "ORDER DETAIL ENTRY".
    05 FILLER AT 2, 1   VALUE "CUSTOMER".
    05 ENTRY-GROUP  AT 5, 4.
       10  FILLER AT     @,     @  VALUE "ITEM".
       10  FILLER AT     @, @ + 9  VALUE "QUANT".
```

WHEN ABSENT/BLANK CLAUSE. The WHEN ABSENT/BLANK clause controls the disposition of working storage associated by TO or USING clauses with absent or blank fields.

The syntax of the WHEN ABSENT/BLANK clause is:

```
WHEN  {ABSENT}   {CLEAR}
      {BLANK }   {SKIP }
```

where

ABSENT

indicates that the disposition is for absent fields, that is, fields for which no data is returned from the terminal. An absent field is possible only if the terminal has a Modified Data Tag (MDT). (Refer to the paragraph Terminal Considerations in this section for information regarding the MDT.)

BLANK

indicates that the disposition is for fields containing blanks, null characters, or fill characters.

CLEAR

sets the working storage to zero for numeric items and to spaces for alphabetic or alphanumeric items.

SKIP

leaves the working storage unaltered.

If this clause is omitted, absent fields are skipped and blank fields are cleared; that is, WHEN ABSENT SKIP and WHEN BLANK CLEAR.

WHEN FULL CLAUSE. The WHEN FULL clause specifies the action to be taken when the last position of an input screen field is filled and additional characters are keyed into the terminal.

The syntax of the WHEN FULL clause is:

```
[ WHEN ] FULL  { TAB  }
               { LOCK }
```

where

TAB

causes the cursor to advance to the next input field.

LOCK

causes the terminal to lock the keyboard.

If the clause is omitted, the default is *LOCK*.

The WHEN FULL clause is only effective for terminals that support more than one alternative action. Currently those terminals are the T16-6520, T16-6530, and the IBM-3270.

TERMINAL CONSIDERATIONS

For dial-in terminals, the TCP issues a wait for modem connect immediately after the terminal file is opened. At terminal start-up time no program unit or data area is attached to the terminal; therefore, the terminal is using a minimum of TCP resources while waiting for modem connect. When the terminal is stopped, the terminal file is closed. The close causes the modem to disconnect if no other process has the terminal file open.

Tandem currently supports the IBM-3270, the T16-6510, the T16-6520, the T16-6530, and any device operating as a conversational mode terminal as recognized by the GUARDIAN File System. Each terminal set has unique requirements. These requirements are described in the following paragraphs.

IBM-3270 Considerations

The supported IBM-3270 terminals have a number of different physical screen sizes. In general, a screen can be used on any model that has a physical size at least as large as the logical size of the screen definition. However, a field that wraps from one line to the next in the logical screen definition does not wrap (or wraps differently) if the physical screen width exceeds the logical screen width; this is because the field goes to the next line only at the end of the physical line. This is an important consideration if a screen is intended to run on both 40- and 80-character width displays.

All nonliteral fields must reserve a character position immediately before the field. For example:

If a field is at line 2, column 2, and is one character long, then 2,1 and 2,2 are reserved for the field. A second field cannot be at 2,3 because both fields would attempt to use location 2,2.

A field cannot be at 1,1 because the character before 1,1 does not exist and thus cannot be reserved.

The IBM-3270 terminal has a Modified Data Tag (MDT) associated with this character that immediately precedes the field. If the MDT is on when a read modified operation is performed, the data in the field is sent to the computer; if the MDT is not on, data is not sent. The MDT can be set or reset from the computer or from the keyboard. In normal operation, the MDT is reset by the computer and set by the terminal after data is entered into the field.

In SCREEN COBOL, the MDT is treated syntactically as a display attribute, even though the MDT affects data transmission rather than the display. Normally, MDTs are not referenced by a SCREEN COBOL program, but they can be manipulated by the program. One possible method is to specify MDTON as an initial display attribute of a field that has a VALUE clause; this causes the initial value to be returned as if entered by the terminal operator even though the terminal operator does not change anything in the field.

The TCP controls the MDTs in the same way it controls display attributes with two important exceptions:

- When a TURN TEMP statement selects an input field for changing of display attributes, the MDT bit is always set.

- When a RESET TEMP statement selects an input field for resetting of display attributes, the MDT bit is set, regardless of the initial MDT attribute of the field.

These two exceptions apply only to the TURN and RESET statements that have the TEMP modifier.

These MDT rules allow fields to be handled correctly when they contain errors. When an error is detected in a field, a TURN TEMP of a display attribute is normally performed on that field, whether explicitly by the program or implicitly by the action of the ACCEPT statement. As indicated by the preceding rules, the MDT will be set also, thus guaranteeing that the field will again come in from the terminal on the next read operation. After that next read operation, a RESET TEMP is performed, which removes the flagging display attribute while again forcing the MDT bit on. The latter setting of the MDT is necessary because a further read of the same data might be performed if another field is found to be in error, and the data in the field that was RESET must come in once again and be properly accepted.

SCREEN COBOL supports the program attention (PA) keys 4 through 10. A user-replaceable procedure that lets the PATHWAY Terminal Control Process (PATHTCP) support these keys is described in Appendix D.

The PROTECTED display attribute is allowed for IBM-3270 terminals (refer to Table 4-1 of Section 4). This attribute is a control that determines whether or not a field is protected against terminal operator entry. Normally, all input fields are unprotected, and all others are protected. The program can use the PROTECTED attribute to dynamically control the protection of 3270 fields; care must be taken to ensure the field has the appropriate protection during sensitive operations, for example, during ACCEPT statement processing.

The minimum separation between screen elements for the IBM-3270 is indicated in Table 5-2.

Table 5-2. Minimum Separation (in Characters) Between Screen Elements for the IBM-3270

First Element \ Second Element	Field	Literal	Overlay Area	End of Screen
Start of base screen	1	1	0	0
Start of overlay screen occupying an overlay area that does not have the same width as its base screen (a)	1	1	0	0
Field	1	1	0	0
Literal	1	1	0	0
Overlay Area	1	1	0	0

NOTE (a):

When an overlay screen occupies an overlay area that does not have the same width as its base screen, an overlay field cannot wrap from one line to the next.

T16-6510 Considerations

When defining screen declarations for display on a T16-6510, the following restrictions should be noted:

- The following fields must have one reserved character immediately before the field and one immediately following the field:
  - All input fields
  - Output fields and literals with BLINK or HIDDEN attributes
  - Output fields referred to in a TURN or RESET statement

  Note that none of the above named fields can begin in line 1, column 1 because the character before 1,1 does not exist.

- The last position of the screen (line 24, column 80) cannot be used, including use as a reserved character as specified in the previous restriction.

- The PROTECTED attribute of a field cannot be changed. The PROTECTED attribute determines the intensity; therefore, any other intensity specifications are ignored.

- Overlay areas must have the exact same width as the base screen.
- Screens occupying an overlay area that is scrolled cannot contain any input fields.
- Fields cannot wrap from the bottom to the top line of the screen.
- The RETURNED bit is always set for a SHADOWED field.

The T16-6510 terminal does not support Modified Data Tags (MDT).

The minimum separation between screen elements for the T16-6510 is indicated in Table 5-3.

Table 5-3. Minimum Separation (in Characters) Between Screen Elements for the T16-6510

First Element \ Second Element	In/Attr (a)	Out/NoAttr (a)	Overlay Area	End of Screen
Start of Screen	1	0	0	1
In/Attr (a)	2	1	1	2
Out/NoAttr (a)	1	0	0	1
Overlay Area	1	0	0	1

NOTE (a):

The space requirements of fields and literals depend upon certain characteristics. Two groupings can be made, identified above as In/Attr and Out/NoAttr. A field or literal can be classified as follows:

Out/NoAttr: Output-only fields or literals that do not have BLINK or HIDDEN attributes and that are not used in a TURN or RESET statement.

In/Attr: All other fields and literals.

T16-6520 Considerations

All nonliteral fields must reserve a character immediately before the field. For example:

If a field is at line 2, column 2, and is one character long, then 2,1 and 2,2 are reserved for the field. A second field cannot be at 2,3 because both fields would attempt to use location 2,2.

A field cannot be at 1,1 because the character before 1,1 does not exist and thus cannot be reserved.

The T16-6520 terminal has an MDT associated with this character that immediately precedes the field. The TCP uses the read modified data operation when reading the screen. If the MDT is on when a read modified operation is performed, the data in the field is sent to the computer; if the MDT is not on, data is not sent. The MDT can be set or reset from the computer or from the keyboard. In normal operation, the MDT is reset by the computer and set by the terminal after data is entered into the field.

In SCREEN COBOL, the MDT is treated syntactically as a display attribute, even though the MDT affects data transmission rather than the display. Normally, MDTs are not referenced by a SCREEN COBOL program, but they can be manipulated by the program. One possible method is to specify MDTON as an initial (default) display attribute of a field that has a VALUE clause; this causes the initial value to be returned as if entered by the operator even though the operator does not change anything in the field.

During execution of a SCREEN COBOL program, the TCP controls the MDTs in the same way it controls display attributes with two important exceptions:

- When a TURN TEMP statement selects an input field for changing of display attributes, the MDT bit is always set.
- When a RESET TEMP statement selects an input field for resetting of attributes, the MDT bit is set, regardless of the initial MDT attribute of the field.

These two exceptions apply only to the TURN and RESET statements that have the TEMP modifier. When the TURN and RESET statements do not have the TEMP modifier, these statements treat the MDT attributes like normal display attributes.

These MDT rules allow fields to be handled correctly when they contain errors. When an error is detected in a field, a TURN TEMP of a display attribute is normally performed on that field, whether explicitly by the program or implicitly by the action of the ACCEPT statement. As indicated by the preceding rules, the MDT will be set also, thus guaranteeing that the field will again come in from the terminal on the next read operation. After that next read operation, a RESET TEMP is performed (normally only implicitly as part of the ACCEPT action), thus removing the display attribute set by the TURN TEMP statement while again forcing the MDT bit on. The latter setting of the MDT is necessary because a further read of the same data might be performed if another field is found to be in error, and the data in the field that was RESET must come in once again and be properly accepted.

The PROTECTED display attribute is allowed for T16-6520 terminals (refer to Table 4-1 of Section 4). This attribute is a control that determines whether or not a field is protected against terminal operator entry. Normally, all input fields are unprotected, and all others are protected. The program can use the PROTECTED attribute to dynamically control the protection of T16-6520 fields; care must be taken to ensure the field has the appropriate protection during sensitive operations, for example, during ACCEPT statement processing.

A limit is placed on the number of fields that can exist on a screen. This is due to a space limitation with the internal Data Attribute Table within the terminal. Each field requires at least one entry in the table; fields that have more than one character position separating them and fields that are followed by literal values require two entries in the table. The table has a maximum of 332 entries. Thus, a screen can have at most 166 fields if the fields meet the requirement for two entries in the internal Data Attribute Table.

Fields cannot wrap from the bottom to the top line of the screen.

The minimum separation between screen elements for the T16-6520 is indicated in Table 5-4.

Table 5-4. Minimum Separation (in Characters) Between Screen Elements for the T16-6520

First Element \ Second Element	Field	Literal	Overlay Area	End of Screen
Start of base screen	1	1	0	0
Start of overlay screen occupying an overlay area that does not have the same width as its base screen (a)	1	1	0	0
Field	1	1	0	0
Literal	1	0 or 1 (b)	0	0
Overlay Area	1	1	0	0

NOTES:

(a) When an overlay screen occupies an overlay area that does not have the same width as its base screen, an overlay field cannot wrap from one line to the next.

(b) If two successive literals have the same attributes, then no separation is necessary; otherwise at least one position must separate them.

T16-6530 Considerations

The T16-6530 terminal is upward compatible with the T16-6520 terminal. Considerations listed for the T16-6520 also apply to the T16-6530.

Program units compiled for a Tandem 6520 terminal can be run on a 6530 terminal. *T16-6520* must be specified as the terminal-type in the OBJECT-COMPUTER paragraph of the Environment Division and the program unit must be run on a 6530 terminal. Those features unique to the Tandem 6530 terminal will not function.

The additional screen memory of the Tandem 6530 (relative to the Tandem 6520) is used to retain screen format information in the terminal. Redisplaying a screen from the terminal memory reduces time utilization. If the terminal screen memory is full and a screen is to be displayed for the first time, PATHWAY uses a least-recently-used algorithm to select the screen for replacement.

PATHWAY can enable a return key function when a SCREEN COBOL program takes control of a Tandem 6530 terminal. For a return key function to become effective, the program's SPECIAL-NAMES paragraph must contain a RETURN-KEY phrase as the *system-name* parameter. The return key function is local to a SCREEN COBOL program and must be defined in the program or no return key function will exist. To use this function in a program that was previously compiled, you must recompile the program and include the RETURN-KEY phrase. If a program is defined for a Tandem 6520 terminal and run on a Tandem 6530 terminal, you cannot use a return key function.

The T16-6530 terminal enables the use of other devices to input data into screen fields. Refer to the RECEIVE clause earlier in this section.

Conversational Mode Considerations

A conversational terminal is any device that operates as a conversational mode terminal as recognized by the GUARDIAN File System. The TCP assumes that this device can process carriage return, line feed, and bell operations. If the data entered during accept processing exceeds the size of the I/O buffer, a field prompt is simply redisplayed without an advisory error message.

SPECIAL REGISTERS

Special registers are data items defined automatically by the SCREEN COBOL compiler, not by the program. Each special register has a particular purpose, and should be used only in the manner outlined in its description.

DIAGNOSTIC-ALLOWED Special Register

The DIAGNOSTIC-ALLOWED special register indicates whether or not diagnostic screens are to be displayed to inform the terminal operator if an error or termination condition occurs. A copy of this register is local to each program unit.

The register is initialized to the value specified by the DIAGNOSTIC parameter of the PATHCOM SET TERM command each time the program unit is called. If the DIAGNOSTIC parameter has not been specified on the SET TERM command, the default value is YES, which enables display of diagnostic screens. The program can move the value NO into the register to disable display of diagnostic screens.

The register has an implicit declaration of

```
01 DIAGNOSTIC-ALLOWED PIC AAA.
```

For additional information regarding the use of diagnostic screens, refer to Section 6 and Appendix A.

LOGICAL-TERMINAL-NAME Special Register

The LOGICAL-TERMINAL-NAME special register contains the name of the terminal executing the program unit. The name of the terminal is defined in PATHCOM through the FILE parameter of the SET TERM command. A single copy of this register is global to the program units. The register is initialized when the terminal is first started.

The register has an implicit declaration of

```
01 LOGICAL-TERMINAL-NAME PIC X(16).
```

NEW-CURSOR Special Register

The NEW-CURSOR special register controls placement of the cursor in the next accept operation. A single copy of this register is global to the program units.

If the register value is not a valid screen position when an accept operation begins, the cursor is positioned to the first field of the ACCEPT statement. At the end of any accept operation, the register is set to zero; this causes the default position for the next accept operation to be the first field of that ACCEPT statement.

The register has an implicit declaration of

```
01 NEW-CURSOR.
   02 NEW-CURSOR-ROW  PIC 9999 COMP.
   02 NEW-CURSOR-COL  PIC 9999 COMP.
```

OLD-CURSOR Special Register

The OLD-CURSOR special register indicates the row and column occupied by the cursor at the last accept operation. A single copy of this register is global to the program units. The register is set by each ACCEPT statement executed by a program unit; the program unit can subsequently access the register.

The register has an implicit declaration of

```
01 OLD-CURSOR.
   02 OLD-CURSOR-ROW  PIC 9999 COMP.
   02 OLD-CURSOR-COL  PIC 9999 COMP.
```

REDISPLAY Special Register

The REDISPLAY special register can prevent unnecessary moving of an entire screen into terminal memory. This register indicates to the TCP whether or not a screen must be sent to the terminal during processing of a DISPLAY statement. The REDISPLAY register affects only the DISPLAY statement.

This register supports T16-6520 and T16-6530 terminals that store multiple screens in terminal memory and can display these screens upon command. This register does not support terminals operating in conversational mode.

A single copy of the REDISPLAY register is global to the SCREEN COBOL program units. The register is set to NO when the terminal is first started. The SCREEN COBOL program can move YES into the register for specific DISPLAY statements. When entering or returning from a called program, the value of the REDISPLAY register is undefined.

When the REDISPLAY register is set to NO (the normal setting) and a DISPLAY statement is executed, the following occurs:

1. The TCP checks whether the screen is in the terminal memory and whether any data fields have been entered since the previous display operation. If the screen is present and no changes have been made to the data items, the TCP displays the screen.

2. If a data item has been changed since the previous display operation, the variable data items associated with the screen are moved from the Working-Storage Section to the terminal memory. This operation takes place only when redisplaying a screen, not when displaying a base screen or menu screen.

When the register is set to YES and a DISPLAY statement is executed, the TCP checks whether the screen is in the terminal memory and the following occurs:

- If the screen is present, the TCP displays the screen from the terminal memory. No data items are moved from the Working-Storage Section.
- If the screen is not present, the TCP moves the variable data items associated with the screen from the Working-Storage Section to the terminal memory, as if the register had been set to NO. Then, the screen is displayed.

The register has an implicit declaration of

```
01 REDISPLAY PIC AAA.
```

An example of the REDISPLAY special register is:

```
PROCEDURE DIVISION.
 .
 .
 MOVE "YES" TO REDISPLAY.
 DISPLAY EMPLOYEE-REC-SCREEN.
 .
```

RESTART-COUNTER Special Register

The RESTART-COUNTER special register contains the number of times a transaction has been restarted during transaction mode. The first time the BEGIN-TRANSACTION verb executes, the register is set to zero. This number is incremented immediately following each execution of the BEGIN-TRANSACTION verb.

The register has an implicit declaration of

```
01 RESTART-COUNTER  PIC 9999 COMP.
```

STOP-MODE Special Register

The STOP-MODE special register can prevent interruption of multiple step transactions. A single copy of this register is global to the program units.

The register is set to zero when the terminal is first started, and the value is subsequently under program control. Most programs will continue with a value of zero.

When the value is nonzero, several PATHCOM commands are affected. The effect of the STOP TERM, SUSPEND TERM, and FREEZE SERVER commands is delayed until the register value returns to zero. The SUSPEND and FREEZE commands can be issued in a form that causes the STOP-MODE value to be disregarded.

The register has an implicit declaration of

```
01 STOP-MODE  PIC 9999 COMP.
```

TELL-ALLOWED Special Register

The TELL-ALLOWED special register can be set by the program to control the issuing of tell messages during ACCEPT statement processing.

A copy of this register is available to each program unit. The register is initialized to YES each time the program unit is called. The program can move NO into the register to prevent tell messages from being displayed during succeeding accept operations.

When this register is set to YES and a tell message is waiting, the following occurs:

- When the TCP is about to complete an accept operation, it displays the tell message (prefixed by the word MESSAGE:) in the ADVISORY field.
- The TCP waits for any function key from the terminal operator, then resets the field and completes the accept operation.

When this register is set to NO, display of the tell message is postponed.

The register has an implicit declaration of

```
01 TELL-ALLOWED  PIC AAA.
```

TERMINAL-FILENAME Special Register

The TERMINAL-FILENAME special register contains the internal form of the file name for the terminal executing the program unit. A single copy of this register is global to the program units. The register is initialized when the terminal is first started.

The register has an implicit declaration of

```
01 TERMINAL-FILENAME  PIC X(24).
```

TERMINAL-PRINTER Special Register

The TERMINAL-PRINTER special register contains the external form of the file name for the printer that is associated with the terminal executing the program unit. If no associated printer is defined in PATHCOM, this register contains blanks. A single copy of this register is global to the program units. The register is initialized when the terminal is first started.

The register has an implicit declaration of

```
01 TERMINAL-PRINTER  PIC X(36)
```

TERMINATION-STATUS Special Register

The TERMINATION-STATUS special register communicates completion status of an ACCEPT, SEND, and BEGIN-TRANSACTION statement. A copy of this register is available to each program unit. The register is initialized to zero each time the program unit is called.

This special register also communicates an error number when the ON ERROR branch of a CALL statement is taken.

The register has an implicit declaration of

```
01 TERMINATION-STATUS  PIC 9999 COMP.
```

TERMINATION-SUBSTATUS Special Register

The TERMINATION-SUBSTATUS special register communicates an error number further describing the error communicated in the special register TERMINATION-STATUS when the ON ERROR branch of a CALL statement is taken. A copy of this register is local to each program unit.

The register has an implicit declaration of

```
01 TERMINATION-SUBSTATUS  PIC 9999 COMP.
```

TRANSACTION-ID Special Register

The TRANSACTION-ID special register contains the value of the transaction identifier that the Transaction Monitoring Facility (TMF) assigns when the BEGIN-TRANSACTION statement executes. TMF assigns a unique identifier to this register for each new or restarted transaction. The register is set to SPACES after either the END-TRANSACTION or the ABORT-TRANSACTION statement executes.

Generally, the contents of this special register should not be displayed on a terminal screen because the associated data item contains binary data. Use this register to locate uniquely identified transactions.

The register has an implicit declaration of

```
01 TRANSACTION-ID  PIC X(8).
```

SECTION 6

PROCEDURE DIVISION

The Procedure Division includes all of the processing steps for the program. The steps are organized into SCREEN COBOL statements and sentences, and grouped into paragraphs, procedures, and sections.

The format of the Procedure Division is shown in Figure 6-1.

```
PROCEDURE DIVISION [ USING data-name-1 [ , data-name-2 ] ... ] .
[ DECLARATIVES.
{ [ section-name SECTION . ]
    [ paragraph-name . [ sentence ] ... ] ... } ...
{ paragraph-name . [ sentence ] ... } ...
END DECLARATIVES. ]
{ [ section-name SECTION . ]
    [ paragraph-name . [ sentence ] ... ] ... } ...
{ paragraph-name . [ sentence ] ... } ...
```

Figure 6-1. Procedure Division Format

DIVISION STRUCTURE

The division begins with a division header. The format of the header is:

```
PROCEDURE DIVISION [ USING data-name-1 [ , data-name-2 ] ... ] .
```

The header must begin in Area A and must be terminated with a period separator.

The USING phrase is applicable only in a subprogram that is to execute under control of a CALL statement that also contains a USING phrase. The identifiers in the USING phrase must correspond exactly in number and structure to the identifiers specified in the USING phrase of the CALL statement. A maximum of 29 names can be specified.

Execution begins with the first executable statement after the Procedure Division header, excluding any declarative procedures, and continues on in the logical order. The Procedure Division header must be immediately followed by the DECLARATIVES keyword and declarative procedures, or immediately followed by a paragraph or section name.

During execution, control is transferred to a paragraph only at the beginning of the paragraph. Control is passed to a sentence within a paragraph only from the immediately preceding sentence, unless the immediately preceding sentence is a GO TO statement.

When control reaches the end of a paragraph, control passes to the first section of the following paragraph. The only exception is when control reaches the end of a paragraph and that paragraph is the last in the range of a currently active PERFORM operation.

An example of Procedure Division structure is shown in Figure 6-2.

```
PROCEDURE DIVISION.
initialization SECTION.
get-started.
   sentence
      statement
      statement.
   sentence
      statement
          .
          .
          .
finish-up-init.
   sentence
          .
          .
          .
main-processing SECTION.
begin-it.
   sentence
          .
          .
          .
process-input-data.
          .
          .
          .
end-of-job.
   EXIT PROGRAM.
```

Figure 6-2. Procedure Division Structure

Declarative Procedures

A special portion of the Procedure Division is reserved for declarative procedures. These procedures are screen recovery routines specified by USE statements. When used, this portion must be coded immediately after the Procedure Division header. The portion begins with keyword DECLARATIVES and ends with keywords END DECLARATIVES. The following example illustrates a declarative procedure:

```
PROCEDURE DIVISION.
DECLARATIVES.
RECOV-SECT-1 SECTION.
USE FOR RECOVERY...
        .
        .
        .
END DECLARATIVES.
MAIN SECTION.
begin-my-program.
        .
        .
        .
```

Sections

A section, which is optional, is used to group related paragraphs for processing steps. Reference to a section name in a PERFORM statement, for example, would include all paragraphs in that section for the PERFORM execution range.

A section begins with a section header in Area A. The format of the header is:

```
section-name SECTION.
```

A section ends at the next section header, at keywords END DECLARATIVES, or at the physical end of the Procedure Division.

Paragraphs

A paragraph is used to group related sentences and statements. A paragraph usually has at least one sentence, but sentences are not required. For example:

```
get-all-input.

get-the-first-record.
 ACCEPT my-screen...
```

Reference to a paragraph name permits branching from one area of code to another.

A paragraph begins with a paragraph name in Area A. A paragraph ends immediately before the next paragraph name or section name, or at the physical end of the Procedure Division.

Sentences and Statements

A sentence is a string of one or more statements, ending with a period. A statement is a combination of words and symbols beginning with a SCREEN COBOL verb. For example:

```
chk-report-yy.
   IF current-yy IS LESS THAN 0 OR GREATER THAN 99
      DISPLAY "REPORT YEAR IS NOT BETWEEN 00 AND 99, RE-ENTER "
         "YEAR" IN msg-1
      ACCEPT current-yy UNTIL my-file1
      GO TO chk-report-yy.
```

Sentences can be grouped into three functional categories:

- imperative — takes an action unconditionally
- conditional — takes an action based on a condition
- compiler directing — uses compiler-directing verbs COPY or USE

An imperative sentence is constructed from one or more imperative statements terminated by a period. An imperative sentence can have a GO TO statement or an EXIT PROGRAM statement. If an EXIT PROGRAM statement is present, it must be the last statement in the sentence.

The following examples illustrate imperative sentences:

```
ADD a1 TO b1 GIVING c1, d1, e1.

ADD 25 TO x2,
   GO TO next-image.
```

A conditional sentence tests a conditional item or some relationship between values to determine an action to take.

The following example illustrates a conditional sentence:

```
IF last-tax IS LESS THAN current-tax
    PERFORM higher-tax
ELSE  PERFORM  lower-tax.
```

Procedures

A procedure consists of a paragraph, a group of successive paragraphs, a section, or a group of successive sections. A procedure name is a paragraph or section name; the name can be qualified.

PROCEDURE DIVISION STATEMENTS

Procedure Division statements can be grouped into eight categories. Table 6-1 lists each statement and its category.

Table 6-1. Classification of Statements

Statement Category	Statement Keywords
Arithmetic	ADD COMPUTE DIVIDE MULTIPLY SUBTRACT
Conditional	BEGIN-TRANSACTION ON ERROR CALL...ON ERROR IF SEND...ON ERROR
Data Movement	ACCEPT DATE/DAY/TIME MOVE SET
Terminal Input/Output	ACCEPT CLEAR DELAY DISPLAY PRINT SCREEN RECONNECT MODEM RESET SCROLL SEND TURN
Interprogram Communicating	CALL CHECKPOINT EXIT PROGRAM
Program Control	EXIT GO TO PERFORM STOP RUN
Compiler Directing	COPY USE
Transaction Monitoring	ABORT-TRANSACTION BEGIN-TRANSACTION END-TRANSACTION RESTART-TRANSACTION

Statements are described in alphabetic order in the following paragraphs.

ABORT-TRANSACTION Statement

The ABORT-TRANSACTION statement aborts the transaction of a terminal operating in transaction mode. Transaction mode is an operating mode in which PATHWAY servers that are configured to run under the Transaction Monitoring Facility (TMF) can lock and update audited files. When this statement executes, all data base updates that were made to audited files during the transaction are backed out, and no attempt is made to restart the transaction.

The syntax of the ABORT-TRANSACTION statement is:

```
ABORT-TRANSACTION
```

Execution of this statement causes the terminal to leave transaction mode and the special register TRANSACTION-ID to be set to SPACES. If the terminal is not in transaction mode when this statement is executed or if a fatal error occurs while aborting the transaction, the terminal is suspended for a pending abort.

Refer to the *Introduction to Transaction Monitoring Facility (TMF)* and the *Transaction Monitoring Facility (TMF) Users Guide* for additional information about programming with TMF.

ACCEPT Statement

The ACCEPT statement operations for terminals operating in block mode are slightly different from operations for terminals operating in conversational mode.

If the terminal associated with the SCREEN COBOL program is operating in block mode, ACCEPT performs the following:

- Displays all the prompt values defined for the screen fields described with PROMPT clauses.
- Waits for response from the terminal.
- Receives data for input to the program data area from the terminal.
- Returns only valid data to the program; checking the definitions in the Screen Section of the Data Division determines the validity of the data.

If invalid data is entered and an ADVISORY field is defined for the base screen, an error message is displayed on the screen, the field in error is enhanced, and the data can be corrected or reentered.

If the terminal associated with the SCREEN COBOL program is operating in conversational mode, ACCEPT performs the following:

- Displays the prompt value defined for the first screen field described with a PROMPT clause. The prompt value is always displayed in the first column of the screen line.
- Waits for response from the terminal. If the TIMEOUT phrase is used, ACCEPT waits the time limit specified in this phrase.
- Receives input from the terminal and stores the data into the associated working-storage items of the program data area. Input can be accepted from the terminal one screen field at a time; one field per line. However, the capability referred to as typeahead enables entering data for more than one field on the same line.

- Returns only valid data to the program; checking the definitions in the Screen Section of the Data Division determines the validity of the data.

If invalid data is entered and an ADVISORY field is defined, an error message is displayed, the prompt is redisplayed for the field in error, and the data can be reentered. If an ADVISORY field is not defined for the base screen, only the prompt is redisplayed for the field in error and the data can be reentered.

The syntax of the ACCEPT statement is:

```
ACCEPT [ screen-identifier ] ...

{ UNTIL { [ ( ] { comp-condition-1 } ... [ ) ] ESCAPE [ ON ]  } }
        {   [ ( ] { comp-condition-2 } ... [ ) ]              }
        {                                                     }
        { [ ( ]   comp-condition-1    ... [ ) ]               }
        {                                                     }
        { ESCAPE [ ON ] { [ ( ] { comp-condition-2 } ... [ ) ] } }
```

where

screen-identifier

specifies the screen fields from which data is accepted. Each *screen-identifier* can name an entire screen, a screen group, or an elementary input item of any base or overlay screen that is currently displayed. If *screen-identifier* is a group, all subordinate elementary items that have a TO or USING clause in their definition are referenced. The *screen-identifier* cannot be a subscripted item.

The order in which fields appear in the *screen-identifier* list is the order in which they are checked and converted.

If this parameter is omitted, the completion condition *comp-condition* in either the UNTIL or ESCAPE clause determines when the statement is to terminate. No data is accepted from the screen, and no working storage item is altered. A typical reason for omitting the identifier would be during the display of a help screen.

In block mode, if a screen contains only filler items and a DISPLAY statement is followed by an ACCEPT statement without a screen identifier, the screen remains until a function key signals the termination of the ACCEPT statement. A typical instance for omitting the identifier would be during the display of a help screen.

UNTIL and ESCAPE

specify the conditions under which the statement is to complete. These conditions are typically the names of the terminal function keys that the terminal operator can use. At least one of these two clauses must be present with a completion condition. If both clauses are present, any one completion condition can appear in only one of the two clauses.

comp-condition-1

specifies the completion conditions under which the statement is to terminate with input of data.

→

comp-condition-2

specifies the completion conditions under which the statement is to terminate without input of data.

Language elements that can appear as *comp-conditions* are:

ABORT

indicates that the abort input control characters were entered to terminate the ACCEPT statement. This phrase is effective only for terminals in conversational mode. Refer to the Input Control Characters paragraph described in Section 5.

ABORT is allowed only in the ESCAPE clause. If this phrase is executed, the data items in working storage are not changed.

If the terminal is in block mode, ABORT is treated as a comment.

INPUT

indicates that the ACCEPT statement terminates with valid screen input. This phrase is effective only for terminals in conversational mode.

If the terminal is in block mode, INPUT is treated as a comment.

TIMEOUT numeric-literal

indicates that the terminal operator is to be given a limited time to complete the data entry. This language element can appear only in the ESCAPE clause. The time allowed is the number of seconds specified by *numeric-literal*. If the terminal does not respond within the specified time, the ACCEPT operation is completed without input data.

If this phrase is not specified, there is no time limit.

NOTE

In conversational mode, all *comp-condition* phrases except ABORT, INPUT, and TIMEOUT are ignored.

mnemonic-names

indicates the ACCEPT operation completes when the terminal operator presses the associated function key. This assumes a *mnemonic-name* has been associated with a terminal function key; the association is specified by an IS phrase in the SPECIAL-NAMES paragraph of the Environment Division. Because of terminal characteristics, certain keys can be used only in the ESCAPE clause. The function keys are described in Section 4.

mnemonic-name-1 THROUGH mnemonic-name-2

indicates a set of function keys as a single condition. The *mnemonic-names* must be associated with the keys from the same range of function keys; for example, F2 THROUGH F15.

TIMEOUT ACCEPT OPERATION. The numeric literal specified in the TIMEOUT phrase is processed for the ACCEPT statement in the following manner. The SCREEN COBOL compiler converts the number of seconds indicated in *numeric-literal* into 2 to the *t* power (2t). The value 2t is the actual time that elapses before the ACCEPT statement times out. If the numeric literal is not an exact power of 2, the number of seconds is converted to the next higher power of 2. Table 6-2 illustrates the conversion from *numeric-literal* to the actual time that elapses.

Table 6-2. TIMEOUT Conversions for ACCEPT Statement

POWER OF 2 t	1	2	2	3	4	4	5	5	6	6	7	7	8
TIMEOUT *numeric-literal* (in seconds)	2	3	4	8	9	16	17	32	33	64	65	127	129
Actual time elapse 2t (in seconds)	2	4	4	8	16	16	32	32	64		128	128	256

BLOCK MODE ACCEPT OPERATION. The ACCEPT statement enables the terminal keyboard and waits for input from the terminal. When a valid control key code is received from the terminal, the keyboard is disabled and a RESET TEMP is executed automatically; this causes the removal of any temporary field attributes and/or data from the display regardless of whether they were originally displayed explicitly by the program or implicitly through the ACCEPT statement. If termination is caused by the *comp-condition* specified in the ESCAPE clause, the ACCEPT statement terminates at this point.

If a prompt is used and the field named in the PROMPT clause is an output field, the ACCEPT statement causes the current value for the output field to be displayed before reading the data input from the terminal. An output field named in the PROMPT clause must be defined as FILLER or defined with a FROM or USING clause.

The data entered from the terminal is checked against the requirements given for the field by its definition in the Screen Section of the Data Division. The TCP checks only those fields referenced by the *screen-identifier* list in the ACCEPT statement.

If errors are discovered and the terminal is in block mode during the data checking, the following occurs:

- If a field with the ADVISORY clause is defined for the current screen, a DISPLAY TEMPORARY of the advisory field automatically occurs using the standard error message for the first error detected.
- If the terminal is equipped with an audible alarm, the alarm sounds provided its use was not suppressed in the SCREEN-CONTROL paragraph of the Environment Division.
- The first field in error has a temporary modification of its display attribute with the standard error enhancement as declared in the SCREEN-CONTROL paragraph. The program can specify that all fields in error are enhanced (refer to the INPUT-OUTPUT Section in Section 4).
- The statement is restarted following these display operations.

If no data errors are found during the checking, the following occurs:

- The validated data from all referenced screen fields present, including all required fields, is converted and moved into the TO or USING data items in working storage associated with the screen fields.

- Absent screen input fields do not change the associated working storage data items unless specifically requested with the WHEN ABSENT field characteristic clause.

- All SHADOWED fields associated with the input fields of the ACCEPT statement have their ENTERED and RETURNED bits set appropriately. If these bits are checked by a comparison statement, the ENTERED and RETURNED bits should be checked together.

If the completion is through an ESCAPE clause, none of the TO or USING data items are affected. Data variables retain their values and SHADOWED ENTERED bits are not valid.

At the end of any accept operation, the NEW-CURSOR special register is set to zero (row 0, column 0). This controls the placement of the cursor for the next accept operation and causes the default position to be the first field of the current ACCEPT statement.

The ACCEPT statement indicates the condition that caused completion by storing the condition code value into the TERMINATION-STATUS special register. Each *comp-condition* is assigned a code value according to its position in the UNTIL or ESCAPE clauses. The codes are assigned by considering the conditions of the UNTIL and ESCAPE clauses to be a single list and assigning each condition the code value that corresponds to its position in the list. When several conditions are grouped together with parentheses, they are considered to all occupy the same position; that is, all the conditions within the parentheses receive the same code value, and the next condition following the group receives the code value that is one greater than that assigned to the conditions in the group.

In the following example, the value of TERMINATION-STATUS will be 1 if ENTER is pressed, 2 for CLEAR, 2 for PA1, and 3 for PF1.

```
ACCEPT CUSTOMER-SCREEN UNTIL ENTER
    ESCAPE ON (CLEAR, PA1), PF1
```

CONVERSATIONAL MODE ACCEPT OPERATION. The ACCEPT statement displays the prompt value for the first screen field described with a PROMPT clause, enables the keyboard, and waits for data to be entered from the terminal. (If no screen field description contains a PROMPT clause, the ACCEPT statement begins at the first column of the screen.) If termination is caused by a *comp-condition* specified in the ESCAPE clause, the ACCEPT statement terminates at this point with no changes to the working storage data items.

The ACCEPT statement always displays the prompt value in the first column of the screen line and positions the cursor at the end of the prompt field regardless of the positions specified for the field in the screen description.

When the terminal is enabled for input, data can be accepted for each input field a line at a time or accepted for more than one field on the same line. If the typeahead capability is used, field or group separators delimit the screen fields such that multiple fields of data are accepted in a single buffer. When using typeahead, only the prompt value for the first field is displayed. Then, no other prompts appear until the end of the input is indicated by either a carriage return or an input control character.

The ACCEPT statement processes input data in the order the data is received from the terminal. The input data is associated with the screen fields in the sequence the fields are defined in the Screen Section. The data is accepted until there is no more input, the abort input character is entered, or an error is detected. The sequence in which the screen identifiers are processed is from top to bottom and from left to right as follows:

1. The screen field with a lower row (line) number is processed before a screen field with a higher row number.

2. Within the same row, the screen field with a lower column number is processed before a screen field with a higher column number.

The input data is checked against the requirements given for a field by the field definition in the Screen Section. Only those fields referenced by the *screen-identifier* list are checked. During ACCEPT statement processing, the input data is scanned for input control characters that identify the input fields and indicate an abort, end-of-input, or restart operation. Mnemonic names (except BELL and HIDDEN) are not recognized in conversational mode. Therefore, function keys have no effect.

A field error affects only the data in the field that contains the error; fields containing data entered before the error was detected remain valid. Fields containing data entered after the error was detected are ignored.

If an error is discovered during the data checking, the following occurs:

- Only the first field having an error is detected and enhanced. The BELL attribute is the only recognized error enhancement in conversational mode.
- If a field with the ADVISORY clause is defined for the current screen, the advisory field is displayed on the next line following the line with the error.
- ACCEPT processing restarts after the error display operation. The prompt for the field containing the error is redisplayed, and the cursor is positioned to accept the correct input.

Not all errors are detected immediately. If an error is detected after subsequent screen fields have been entered and processed, an error message is displayed and the ACCEPT statement is restarted at the beginning. This is the same action that occurs when a restart input character is processed.

If no data errors are found during the checking, the following occurs:

- The validated data from each referenced screen field is converted and moved as the field is received from the terminal. The converted data is placed in either the TO or USING data item in working storage associated with the screen field. The characteristics defined for a screen field such as PICTURE, UPSHIFT, and so forth, apply to the converted value.
- Absent screen input fields do not change the associated working storage data items unless specifically requested with the WHEN ABSENT field characteristic clause.
- All SHADOWED fields associated with the input fields of the ACCEPT statement have their ENTERED and RETURNED bits set appropriately. If these bits are checked by a comparison statement, the ENTERED and RETURNED bits should be checked together.

ACCEPT statement processing stores a condition code into the TERMINATION-STATUS special register. A code value is assigned to each *comp-condition* in the same way as described previously for block mode.

The following example illustrates an ACCEPT statement for conversational mode. The value of TERMINATION-STATUS will be 1 if valid input is entered, 2 for ABORT, and 2 for TIMEOUT.

```
ACCEPT EMPLOYEE-SCREEN UNTIL INPUT
    ESCAPE ON (ABORT, TIMEOUT 180).
PERFORM ONE OF
    300-CHECK-NULL-NAME
    200-EXIT-ROUTINE
    DEPENDING ON TERMINATION-STATUS.
```

ACCEPT DATE/DAY/TIME Statement

The ACCEPT DATE/DAY/TIME statement causes the TCP to obtain the current GUARDIAN system settings for date, day, and time and return them to your program data area.

The syntax of the ACCEPT DATE/DAY/TIME statement is:

```
ACCEPT accept-name FROM  { DATE }
                         { DAY  }
                         { TIME }
```

where

accept-name

is the identifier of the data item where DATE, DAY, or TIME is stored. DATE, DAY, and TIME are typically defined as:

```
PICTURE 9(6)   for  DATE
PICTURE 9(5)   for  DAY
PICTURE 9(8)   for  TIME
```

DATE

is the current date expressed as a 6-digit number yymmdd where yy is the year, mm is the month, and dd is the day. For example, February 25, 1982 would be returned as 820225.

DAY

is the current Julian date expressed as a 5-digit number yyddd where yy is the year and ddd is the day of the year. For example, February 25, 1982 would be returned as 82056.

TIME

is the current time based on a 24-hour clock, expressed as an 8-digit number hhmmsscc where hh is the hour, mm the minutes, ss the seconds, and cc the hundredths of seconds. For example, the time 2:41 P.M. would be returned as 14410000. The range of values allowed is 00000000 through 23595999.

The following sentence stores the current date (yymmdd) in *todays-date*, the Julian date (yyddd) in *julian-date*, and the current time (hhmmsscc) in *time-right-now:*

```
WORKING-STORAGE SECTION.

01  date-and-time-fields.
    05 todays-date       PIC 9(6)      VALUE ZERO.
    05 julian-date       PIC 9(5)      VALUE ZERO.
    05 time-right-now    PIC 9(8)      VALUE ZERO.
    .
    .
    .
PROCEDURE DIVISION.
    .
    .
    .
ACCEPT todays-date FROM DATE
ACCEPT julian-date FROM DAY
ACCEPT time-right-now FROM TIME
```

ADD Statements

The ADD statements sum numeric values and store the results in one or more data items. When defining a field to hold a total, the size of the field should be considered. The receiving field must be large enough to hold the result and thus avoid truncation of nonzero digits. The forms of the ADD statements are:

```
ADD TO
ADD GIVING
ADD CORRESPONDING
```

Each form is described in the following paragraphs.

ADD TO. The ADD TO statement adds together all values specified and then adds that sum to the current value in each data item specified.

The syntax of the ADD TO statement is:

```
ADD { value } ,... TO { result } ,...
```

where

value

is either a numeric literal or the identifier of an elementary numeric data item.

result

is the identifier of a numeric data item to which value, or the sum of the values, is added.

ADD GIVING. The ADD GIVING statement adds together all values specified and then replaces the current value of each data item specified with the sum.

The syntax of the ADD GIVING statement is:

```
ADD { value } ,... GIVING { result } ,...
```

where

value

is either a numeric literal or the identifier of an elementary numeric data item.

result

is the identifier of a numeric data item into which the sum of the values is stored.

ADD CORRESPONDING. The ADD CORRESPONDING statement adds together elementary items in one group to any corresponding items in another group and then stores the totals in the second group used for the addition. Items correspond when they have the same names and qualifiers up to but not including the group item name specified in the ADD CORRESPONDING statement.

The syntax of the ADD CORRESPONDING statement is:

```
ADD  {CORR          }  group-1 TO group-2
     {CORRESPONDING }
```

where

group-1 and group-2

are the identifiers of group items in which some or all of the elementary items are numeric.

The totals are placed in the *group-2* items.

The following conventions apply to data items used with the CORRESPONDING phrase:

- A REDEFINES or OCCURS clause can be specified in the data description entry of any data item.
- Data items can be subordinate to a data description entry with a REDEFINES or OCCURS clause.
- No data item can be defined with a level number 66, 77, or 88.

Subordinate data items in two different groups correspond to each other according to the following rules:

- Both data items must have the same data name.
- All possible qualifiers for the sending data item, up to but not including a group name, must be identical to all possible qualifiers for the receiving data item up to but not including the receiving group name.
- Only elementary numeric data items are considered.
- Any data item subordinate to a data item that is not eligible for correspondence is ignored.
- FILLER data items are ignored.

In the following example, all item names except *staples* and *paper* correspond. Those two items are skipped in the add operations. Notice that correspondence depends on the names of the items (and qualifiers other than the highest level ones), and not on their physical order.

```
WORKING-STORAGE SECTION.

01  cabinet-supplies.
    05  writing-tools.
        10  pencils        PIC 99.
        10  pens           PIC 99.
        10  erasers        PIC 99.
    05  paper-clips        PIC 99.
    05  staples            PIC 99.

01  stockroom-supplies.
    05  writing-tools.
        10  pencils        PIC 99.
        10  erasers        PIC 99.
        10  pens           PIC 99.
    05  paper-clips        PIC 99.
    05  paper              PIC 99.
  .
  .
  .
PROCEDURE DIVISION.
  .
  .
  .
   ADD CORRESPONDING cabinet-supplies TO stockroom-supplies.
```

In the following example, only one item (6-12-years) corresponds between the groups:

```
01   test-group-1.
     05  children.
         10  1-5-years
         10  6-12-years
     05  teen-agers.
         10  13-15-years
         10  16-19-years
     05  adults
         10  women
         10  men
```

```
01  test-group-2.
    05  children.
        10  1-3-years
        10  4-5-years
        10  6-12-years
    05  teen-agers
```

Assuming all items are numeric, the following statement sums 6-12-years of children of *test-group-1* with 6-12-years of children of *test-group-2:*

```
ADD CORRESPONDING test-group-1 TO test-group-2
```

BEGIN-TRANSACTION Statement

The BEGIN-TRANSACTION statement marks the beginning of a sequence of operations that are to be treated as a single transaction. When this statement executes, the terminal enters transaction mode. Transaction mode is an operating mode in which PATHWAY servers that are configured to run under the Transaction Monitoring Facility (TMF) can lock and update audited files.

TMF starts a new transaction and assigns a transaction ID number to the terminal. This number is placed in special register TRANSACTION-ID. Two other special registers are set: RESTART-COUNTER is set to 0 to indicate that the transaction is being started for the first time, and TERMINATION-STATUS is set to 1 to indicate that the transaction has started.

The syntax of the BEGIN-TRANSACTION statement is:

```
BEGIN-TRANSACTION [ ON ERROR imperative-statement ]
```

where

ON ERROR

provides a point of control if an error is encountered. No test is made against the transaction restart limit; the transaction is restarted and the ON ERROR branch is taken.

imperative-statement

is the statement to be executed if an error occurs or the transaction is being restarted.

If the ON ERROR phrase is omitted and the number of restarts equals the transaction restart limit, the terminal is suspended, but can be restarted.

If the transaction fails for any reason while the terminal is in transaction mode, TMF backs out any updates performed on the data base for the current transaction. If the transaction was not terminated deliberately by execution of the ABORT-TRANSACTION statement, terminal execution is restarted at the BEGIN-TRANSACTION statement if the ON ERROR phrase is specified or if the ON ERROR phrase is not specified and the number of restarts has not exceeded the transaction restart limit. (The maximum number of times a logical transaction can be automatically restarted is specified with the MAXTMFRESTARTS parameter of the PATHCOM SET PATHWAY command.) TMF assigns a new transaction ID number to the terminal; the TCP marks the screen for screen recovery and increments by 1 the special register RESTART-COUNTER. The special register TERMINATION-STATUS remains at 1.

If the terminal is in transaction mode when the BEGIN-TRANSACTION statement is executed, the current transaction is backed out and the terminal is suspended for a pending abort. Terminal execution cannot be resumed.

The special register TERMINATION-STATUS is set by the BEGIN-TRANSACTION statement to indicate the result of this statement's execution. The possible values of TERMINATION-STATUS are listed in Table 6-3.

Table 6-3. TERMINATION-STATUS Error Numbers

TERMINATION-STATUS Error Number	Meaning
1	The transaction is started or restarted.
2	TMF is not installed. Action without the ON ERROR phrase: the terminal is suspended for pending abort.
3	TMF is not running. Action without the ON ERROR phrase: the terminal is suspended, but can be restarted.
4	A fatal error was encountered while attempting to start the transaction. Action without the ON ERROR phrase: the terminal is suspended for pending abort.

Refer to the *Introduction to Transaction Monitoring Facility (TMF)* and the *Transaction Monitoring Facility (TMF) Users Guide* for additional information about programming with TMF.

CALL Statement

The CALL statement transfers control from one SCREEN COBOL program to another SCREEN COBOL program.

The syntax of the CALL statement is:

```
CALL  {data-name        }  [ USING { identifier } ,... ]
      {program-unit-name}

  [ ON ERROR imperative-statement ]
```

where

data-name

is a nonnumeric data item in the Working-Storage Section or Linkage Section; the value of the data item gives the PROGRAM-ID of another SCREEN COBOL program, as specified in the Identification Division of that program. The *data-name* specification allows the PROGRAM-ID of the called program to be specified dynamically. →

```
program-unit-name
```

is a nonnumeric literal that gives the PROGRAM-ID of another SCREEN COBOL program, as specified in the Identification Division of that program.

```
USING
```

passes data to the program called. A USING phrase must be specified in the Procedure Division header of the called program.

```
identifier
```

is the name of an argument passed to the called program. This *identifier* cannot exceed 2047 bytes; it must be an 01 or 77 level data item in the Working-Storage Section or Linkage Section of the program that is calling the other program. The *identifiers* in the USING phrase must correspond exactly in number and structure to the number and structure of the *identifiers* specified in the USING phrase of the Procedure Division header of the called program. Correspondence is by position in the USING lists.

```
ON ERROR
```

provides a point of control if an error is encountered in a descendent program unit.

If a suspend class error is encountered, control is returned to the next higher level program unit having a CALL statement containing an ON ERROR clause. (A suspend class error condition is one that without the use of the CALL...ON ERROR feature would cause the terminal to become suspended.) If a program unit containing an ON ERROR clause does not exist, the terminal is suspended at the statement where the error occurred. If the terminal is in transaction mode when a suspend class error occurs and the point-of-control CALL...ON ERROR is beyond the scope of the current transaction, the current transaction is aborted.

```
imperative-statement
```

is the statement to be executed if an error occurs.

The data area of a program is initialized each time the program is called; therefore, variables do not retain their values between calls.

If the ON ERROR branch is taken, the special register TERMINATION-STATUS contains an error code describing the error, and the special register TERMINATION-SUBSTATUS contains a value or an error code further describing the error. TERMINATION-STATUS and corresponding TERMINATION-SUBSTATUS error codes are listed in Table 6-4. The error code in TERMINATION-SUBSTATUS is dependent upon the error.

Table 6-4. TERMINATION-STATUS/TERMINATION-SUBSTATUS Error Codes for CALL Statement

TERMINATION-STATUS	TERMINATION-SUBSTATUS
0001 — INVALID PSEUDOCODE DETECTED	TCP P-Register
0002 — DEPENDING VARIABLE VALUE TOO BIG	Descriptor number
0003 — INVALID SUBSCRIPT VALUE	
0004 — SCREEN RECOVERY EXECUTED ILLEGAL INSTRUCTION	
0005 — CALL: ACTUAL NUMBER PARAMETERS MISMATCH FORMAL	
0006 — CALL: ACTUAL PARAMETER SIZE MISMATCHES FORMAL	
0007 — SCREEN OPERATION DONE WITHOUT BASE DISPLAYED	
0008 — INVALID DATA REFERENCE	
0009 — REFERENCED SCREEN IS ILLEGAL FOR TERMINAL TYPE	TCP P-Register
0010 — INTERNAL ERR IN TERMINAL FORMAT ROUTINES	TCP P-Register
0011 — ILLEGAL TERMINAL TYPE SPECIFIED	TCP P-Register
0012 — SCREEN REFERENCED BUT NOT DISPLAYED	Screen number
0013 — OVERLAY SCREEN DISPLAYED IN TWO AREAS	
0014 — ILLEGAL TERMINAL IO PROTOCOL WORD	
0015 — ARITHMETIC OVERFLOW	
0016 — TERMINAL STACK SPACE OVERFLOW	TDA Size (words)
0017 — ERROR DURING TERMINAL OPEN	File Error Code
0018 — ERROR DURING TERMINAL IO	File Error Code
0019 — WRONG TRANSFER COUNT IN TERMINAL IO	Transferred Count
0020 — CALLED PROGRAM UNIT NOT FOUND	

Table 6-4. TERMINATION-STATUS/TERMINATION-SUBSTATUS Error Codes for CALL Statement (Continued)

TERMINATION-STATUS	TERMINATION-SUBSTATUS
0021 — TRANSACTION MSG SEND FAILURE	
0022 — SEND: SERVER CLASS NAME INVALID	
0023 — PSEUDOCODE SIZE TOO BIG	
0024 — TCLPROG DIRECTORY ENTRY IS BAD	
0025 — TERMINAL INPUT DATA STREAM INVALID	
0026 — PROGRAM UNIT HAS WRONG TERMINAL TYPE	
0027 — TRANSACTION MODE VIOLATION	
0028 — TRANSACTION I/O ERROR	File Error Code
0029 — TRANSACTION RESTART LIMIT REACHED	
0030 — TMF NOT CONFIGURED	
0031 — TMF NOT RUNNING	
0040 — INVALID NUMERIC ITEM	
0041 — INVALID PRINTER SPECIFICATION	
0042 — DEVICE REQUIRES ATTENTION	File Error Code
0043 — PRINTER I/O ERROR	File Error Code
0044 — CALL: PROGRAM UNIT NAME INVALID	
0113 — TRANSACTION MSG SIZE EXCEEDS LIMIT	
0114 — MAXIMUM REPLY SIZE EXCEEDS LIMIT	

The TERMINATION-STATUS error numbers related to CALL...ON ERROR correspond directly to the TCP-generated PATHWAY error messages, that is, those errors in the 3000 to 3999 range. For example, TERMINATION-STATUS error 114 corresponds to PATHWAY error message 3114.

Note that TERMINATION-SUBSTATUS becomes undefined when TERMINATION-STATUS is set for reasons other than CALL...ON ERROR return.

Refer to the EXIT PROGRAM statement for additional information on programmatic control of error conditions.

CHECKPOINT Statement

The CHECKPOINT statement causes the current context for the terminal, such as working storage items, to be checkpointed.

The syntax of the CHECKPOINT statement is:

```
CHECKPOINT
```

This statement causes an additional checkpoint. Automatic checkpointing occurs as follows:

- When the terminal is not in transaction mode, the TCP performs a full context checkpoint before execution of a SEND statement and performs a checkpoint of the reply after execution of a SEND statement.
- When the terminal is in transaction mode, the TCP only performs checkpoints at the BEGIN-TRANSACTION and END-TRANSACTION statements. No checkpoints are performed at any SEND statements while in transaction mode.

If the CHECKPOINT statement is issued while the terminal is in transaction mode, the terminal is suspended for pending abort.

CLEAR Statement

The CLEAR statement prepares the terminal for a new set of input.

The syntax of the CLEAR statement is:

```
CLEAR INPUT
```

This statement stores null values into all unprotected fields of the screens currently displayed and resets the Modified Data Tag (MDT) bits of all unprotected fields on terminals that use MDT. Except for the MDT, the attributes of the fields are not affected.

The CLEAR statement differs from the RESET statement as follows:

The CLEAR statement affects all unprotected fields on the display screen.	The RESET statement affects only those fields specified in the statement, whether or not the fields are protected.
The CLEAR statement causes all unprotected fields to become blank.	The RESET statement returns all fields to the initially declared values.
The CLEAR statement does not affect field attributes, although the MDT bits are cleared.	The RESET ATTR statement sets all display attributes to their initial values.
The CLEAR statement requires only a short data sequence.	The RESET statement requires a data sequence from the TCP for each field referenced in the statement.

COMPUTE Statement

The COMPUTE statement evaluates an arithmetic expression and then stores the result in one or more data items.

The syntax of the COMPUTE statement is:

```
COMPUTE { result } ,... = expression
```

where

result

is the identifier of a numeric elementary item.

expression

is an arithmetic expression calculated according to precedence rules described in Section 2.

As with other arithmetic operations, consider truncation situations and how they should be handled.

The following example illustrates the COMPUTE statement:

```
WORKING-STORAGE SECTION.

77  compute-result      PIC 999      VALUE ZEROS.
77  ws-result           PIC S9(9)    VALUE ZEROS.
77  ws-99               PIC S99      VALUE 99.
77  ws-five-ones        PIC S9(5)    VALUE 11111.
01  exponent            PIC 9(5)     VALUE ZERO   COMP.
.
.
.
COMPUTE compute-result = (((24.0 + 1) * (60 - 10)) / 125).
```

(compute-result = 10)

COPY Statement

The COPY statement inserts sections of code into a program for use at compile time. This allows code that is common to several programs to be written once and be maintained easily.

The syntax of the COPY statement is:

```
COPY copy-text  [ {OF} library-name ] .
                  {IN}
```

where

copy-text

is a unique section name in a SCREEN COBOL copy library file.

library-name

is the Tandem file containing the text to be copied. The name is expanded to a full file name using the default subvolume in effect for the compilation. If you specify the library name with a subvolume and a file name, you must enclose the entry in quotation marks. For example, "subvol.afile".

If the *library-name* clause is omitted and copy text exists, the default library name COPYLIB is used for the compilation.

Even though the COPY statement is described as a Procedure Division statement, the statement can be included in a SCREEN COBOL program wherever a character string or separator can appear; the only exception is within another COPY statement. Keyword COPY cannot be split over two lines, but text that follows the keyword can be continued.

Library text is copied into the source program. The SCREEN COBOL copy library must be in the correct format, and each copy text must be written in correct SCREEN COBOL syntax.

A copy library must be an EDIT disc file in the following form:

```
?SECTION copy-text-1  [ , {ANSI   } ]
                         {TANDEM }
text-1
?SECTION copy-text-2  [ , {ANSI   } ]
                         {TANDEM }
text-2
```

Each SECTION line identifies the beginning of a copy-text; the question mark must be in column 1. The content of the text is arbitrary and can be any length. No text line can begin with ?SECTION.

The compiler assumes the source format (ANSI standard reference format or Tandem standard reference format) of the library text is the same as that of the line containing the COPY statement. When the format option is specified, the format overrides the compiler's assumption, permitting a library text to be copied irrespective of the format of the source program. Also, the library text itself can have compiler commands, which are obeyed when the text is copied. Note that after copying is complete, the compiler always reverts to the format in force when it encountered the COPY statement.

During program compilation, copy-text is found by locating the SECTION command whose copy-text name matches *copy-text* in the COPY statement. Text is copied starting at the line after the SECTION line and continues until either another SECTION line is recognized or end-of-file is reached.

In the following example, *text-0* has no SECTION command and could never be copied:

```
text-0

?SECTION copy-text-1
text-1
```

When a library file begins like this, this text could be comments about the library contents.

In the following example, notice that *employee-detail* of the COPY statement is not qualified because the copy library, named COPYLIB, resides on the default volume and subvolume for the compilation:

Contents of copy library COPYLIB

```
?SECTION employee-detail
01   emp-data-in.
     05   emp-no          PIC X(05).
     05   emp-name        PIC X(20).
     05   dept            PIC X(03).
     05   job-class       PIC X(05).
     05   hourly-rate     PIC 9(3)V99.
     05   deductions      PIC 9(3)V99.
     05   salary          PIC 9(7)V99.
```

Source SCREEN COBOL code

```
DATA DIVISION.
WORKING-STORAGE SECTION.

 COPY employee-detail.
```

Compile listing of object code

```
     DATA DIVISION.
     WORKING-STORAGE SECTION.
      COPY employee-detail.

<    01   emp-data-in.
<    05   emp-no            PIC X(05).
<    05   emp-name          PIC X(20).
<    05   dept              PIC X(03).
<    05   job-class         PIC X(05).
<    05   hourly-rate       PIC 9(3)V99.
<    05   deductions        PIC 9(3)V99.
<    05   salary            PIC 9(7)V99.
```

All lines from a copy library are marked by < in the compile listing.

DELAY Statement

The DELAY statement provides a means to delay program execution for a specified period of time.

The syntax of the DELAY statement is:

```
DELAY  {numeric-literal}
       {identifier     }
```

where

numeric-literal

is a numeric value representing one-second units. No limitation is imposed on the value.

identifier

is the identifier of an integer data item representing one-second units. No limitation is imposed on the value.

This statement is intended for use in situations where an error has occurred (such as a terminal I/O error because power to the terminal is off) and the operation encountering the error is to be retried periodically. The following example illustrates the DELAY statement:

```
* highest level program-unit.

 loop.
   CALL menu ON ERROR PERFORM analyze-error.
   IF retry = 1
* delay five minutes, then retry.
      DELAY 300
      GO TO loop
   ELSE
      GO TO giveup.

 analyze-error.
   IF TERMINATION-STATUS = 18 AND
      ( TERMINATION-SUBSTATUS = 171 OR
        TERMINATION-SUBSTATUS = 173 )
       MOVE 1 TO retry
    ELSE
      MOVE 0 TO retry.

* suspend.
 giveup.
   EXIT PROGRAM WITH ERROR.
```

DISPLAY Statements

The DISPLAY statements transmit and display data to the terminal screen. The forms of the DISPLAY statements are:

```
DISPLAY BASE
DISPLAY OVERLAY
DISPLAY RECOVERY
DISPLAY
```

Each form is described in the following paragraphs.

DISPLAY BASE. The DISPLAY BASE statement operations for terminals in block mode are slightly different from operations for terminals operating in conversational mode.

If the terminal associated with the SCREEN COBOL program is operating in block mode, DISPLAY BASE performs the following:

- Clears the current screen display.
- Displays literals, null, and fill characters declared in the Screen Section for the base screen.

If the terminal associated with the SCREEN COBOL program is operating in conversational mode, DISPLAY BASE selects the screen for subsequent operations, but does not display the screen.

DISPLAY BASE establishes the foundation for all other screen operations. Therefore, the statement must be executed before attempting any other display operations. A second DISPLAY BASE statement can be specified at any time to establish a new screen, or to reestablish the same screen.

The syntax of the DISPLAY BASE statement is:

```
DISPLAY BASE base-screen-name
```

where

base-screen-name

is the name of the base screen.

During execution of the first DISPLAY BASE statement for a SCREEN COBOL program, the I/O startup messages prepare the terminal for PATHWAY operation. The program can then act on any terminal I/O errors through the CALL ON ERROR clause.

BLOCK MODE DISPLAY BASE. The input and output fields of the screen are villed with the value specified in the VALUE clause for each field. A field that has no VALUE clause is filled with the fill character. All occurrences of a variable length table are filled, regardless of the current value of the table's controlling variable.

A running SCREEN COBOL program has at most one current base screen; the current base screen is defined by the most recently executed DISPLAY BASE statement. The program can have at most one current overlay screen associated with each of the overlay areas of the current base screen; the current overlay screen is defined by the most recently executed DISPLAY OVERLAY statement for each of the areas. With the exception of the DISPLAY BASE and DISPLAY OVERLAY statements, all screen operations must deal only with the current screens.

The definition of a screen is local to a SCREEN COBOL program; therefore, a program cannot make use of a current screen that was established by another program, even if the declaration of the current screen is identical to the declaration of the screen in the currently executing program. Consequently, the program must perform a DISPLAY BASE to make use of a screen.

If a program has current screens defined and calls another program that has screen declarations, the current screens become undefined for the first program. If the first program is to make use of the screens it previously displayed, the first program must execute DISPLAY BASE/OVERLAY statements after the call to the program has completed.

If a program calls another program that has no screen declarations, the definition of the current screens remains unchanged.

When a program that has defined the current screens executes an EXIT PROGRAM statement, the current screens become undefined. The program must display the screens again to make use of them, even if no intervening screen operations have occurred since its exit.

CONVERSATIONAL MODE DISPLAY BASE. If the terminal associated with the SCREEN COBOL program is operating in conversational mode, DISPLAY BASE selects the screen description for subsequent operations, but does not display the screen. A DISPLAY or an ACCEPT statement must follow the DISPLAY BASE for data to be output to the terminal.

DISPLAY OVERLAY. The DISPLAY OVERLAY statement operations for terminals in block mode are slightly different from operations for terminals in conversational mode.

If the terminal associated with the SCREEN COBOL program is operating in block mode, DISPLAY OVERLAY performs the following:

- Associates an overlay screen description with an overlay area.
- Performs the initial display of the screen in the area, replacing any previous screen in that area.

If the terminal associated with the SCREEN COBOL program is operating in conversational mode, DISPLAY OVERLAY selects an overlay screen description for subsequent operations, but does not display the screen.

The syntax of the DISPLAY OVERLAY statement is:

```
DISPLAY OVERLAY  {overlay-screen-name}  AT overlay-area
                 {SPACES              }
```

where

`overlay-screen-name`

is the name of the overlay screen to be displayed.

`AT overlay-area`

is the name of the overlay area of the currently displayed base screen into which the overlay screen is to be placed.

`SPACES`

causes the overlay area to become blank and restores the area to the state it was in immediately after the base screen was displayed. Any association of an overlay screen with the overlay area is broken.

If the DISPLAY BASE statement does not appear before the DISPLAY OVERLAY statement, an error is generated.

The overlay area must be at least as large as the overlay screen. An overlay screen cannot be displayed in more than one overlay area at the same time.

DISPLAY RECOVERY. The DISPLAY RECOVERY statement initiates screen recovery. A program can use this statement to implement a terminal operator request for screen recovery, thus eliminating duplication of code for recovery actions.

The syntax of the DISPLAY RECOVERY statement is:

```
DISPLAY RECOVERY
```

When DISPLAY RECOVERY executes, the standard error recovery procedure is executed. The recovery process performs the equivalent of a DISPLAY BASE statement for the current base screen followed by a DISPLAY OVERLAY for all currently active overlay screens. The screen recovery process then executes any screen recovery declarative procedures that have been provided in the SCREEN COBOL program.

DISPLAY. The DISPLAY statement transmits data to selected output fields of the screen.

The syntax of the DISPLAY statement is:

```
DISPLAY  [TEMP     ]  [ nonnumeric-literal IN ]
         [TEMPORARY]

 { screen-identifier } ,...

 [DEPENDING [ ON ] identifier]
 [SHADOWED                    ]
```

where

TEMP or TEMPORARY

marks the fields so that they will be reset to their default values when the next RESET TEMP or ACCEPT statement is executed.

If the terminal is operating in conversational mode, this phrase is ignored and DISPLAY performs normally. To temporarily change the value of a screen item, the current value of the associated working storage item must be saved, the value changed, the new value displayed, and then the previous current value must be restored.

nonnumeric-literal

is a value that is sent to the terminal for each selected field. The value is not converted; it is truncated or extended with the fill character if necessary.

If this clause is omitted, the data for a selected screen field is obtained from the working storage data item specified in the FROM or USING clause of the screen field description. The data is converted and edited according to the screen field declaration, and those characters are placed in the field on the terminal display.

→

`screen-identifier`

is a screen, screen group, or elementary output item of any active screen. When *screen-identifier* is not an elementary item, all subordinate elementary items that have a FROM or USING clause in their definitions are referenced.

`DEPENDING ON identifier`

selects either zero or one *screen-identifier* from the list. The statement whose position in the *screen-identifier* list is the same as the value in *identifier* is selected. If the value in *identifier* is less than one or greater than the number of *screen-identifiers*, no *screen-identifier* is selected.

`SHADOWED`

selects from the *screen-identifier* list only those fields that have SHADOWED items in which the SELECT bit is set. Fields in the *screen-identifier* list that do not have SHADOWED items are not selected.

If neither the DEPENDING ON modifier nor the SHADOWED modifier is specified, all fields in the list are selected.

For terminals operating in conversational mode, the DISPLAY statement presents output in order by rows. A screen field value appears on the screen at the column number position specified in the screen field description. Blank lines are not generated (for formatting purposes) such that screen lines generally do not correspond with the line numbers specified in the Screen Section.

To display fully line-formatted screens, define at least one item for every line (row) of the screen. If a row of spacing is required, define the screen item for that row with a VALUE clause specifying blanks. For example, VALUE " ". Then, display the entire screen by specifying the screen name as the *screen-identifier* in the DISPLAY statement.

For terminals operating in conversational mode, the DISPLAY statement performs as follows:

- The DISPLAY statement positions screen items on the output line, in the column location specified in the Screen Section. Another screen item having the same line number description, but not named in the DISPLAY statement will not appear in the screen display.

  If you specify a screen group name to display multiple screen fields, each screen field will appear in the column described for that field. However, the screen fields will be on consecutively numbered lines regardless of the screen field descriptions.

- Any non-filler screen item must be defined with a TO, FROM, or USING clause in the Screen Section. If a screen item is defined with both a VALUE clause and a TO, FROM, or USING clause, the literal in the VALUE clause is never displayed. The DISPLAY statement output is always from the associated working storage data items.

- If *nonnumeric-literal* is listed in a DISPLAY statement and the *screen-identifier* list contains more than one field, the literal appears in each of the screen fields named in the list.

DIVIDE Statements

The DIVIDE statements divide one data item into another and store the results in one or more data items. The forms of the DIVIDE statements are:

```
DIVIDE INTO
DIVIDE GIVING
DIVIDE BY GIVING
```

Each form is described in the following paragraphs.

DIVIDE INTO. The DIVIDE INTO statement divides one data item into one or more other data items.

The syntax of the DIVIDE INTO statement is:

```
DIVIDE divisor INTO { dividend } ,...
```

where

divisor

is either a numeric literal or the identifier of an elementary numeric data item.

dividend

is the identifier of an elementary numeric data item that is the dividend and receiving field for the quotient.

DIVIDE GIVING. The DIVIDE GIVING statement divides one data item into another and stores the quotient in one or more data items.

The syntax of the DIVIDE GIVING statement is:

```
DIVIDE divisor INTO dividend GIVING { quotient } ,...
```

where

divisor

is either a numeric literal or the identifier of an elementary numeric data item.

dividend

is either a numeric literal or the identifier of an elementary numeric data item.

quotient

is the identifier of an elementary numeric data item where the quotient is stored.

DIVIDE BY GIVING. The DIVIDE BY GIVING statement is the same as DIVIDE GIVING, except the dividend is specified first.

The syntax of the DIVIDE BY GIVING statement is:

```
DIVIDE dividend BY divisor GIVING { quotient } ,...
```

where

dividend

is either a numeric literal or the identifier of an elementary numeric data item.

divisor

is either a numeric literal or the identifier of an elementary numeric data item.

quotient

is the identifier of an elementary numeric item where the quotient is stored.

The following example illustrates the DIVIDE BY GIVING statement:

```
WORKING-STORAGE SECTION.
77  leap-year          PIC 9   VALUE ZERO.
77  divide-result      PIC 99  VALUE ZERO.
01  invoice-date.
    05 inv-month       PIC 99.
    05 inv-day         PIC 99.
    05 inv-year        PIC 99.
    .
    .
    .
 PROCEDURE DIVISION.
    .
    .
    .
    DIVIDE inv-year BY 4 GIVING divide-result.
    .
    .
    .
```

END-TRANSACTION Statement

The END-TRANSACTION statement marks the completion of a sequence of operations that are treated as a single transaction. When this statement executes, the terminal leaves transaction mode. Transaction mode is an operating mode in which PATHWAY servers that are configured to run under the Transaction Monitoring Facility (TMF) can lock and update audited files.

The syntax of the END-TRANSACTION statement is:

```
END-TRANSACTION
```

If TMF accepts this statement, any data base updates made during the transaction become committed, the terminal leaves transaction mode, and the special register TRANSACTION-ID is set to SPACES. If TMF rejects this statement, transaction restart occurs.

If the terminal is not in transaction mode when the END-TRANSACTION statement is executed, the terminal is suspended for a pending abort.

Refer to the *Introduction to Transaction Monitoring Facility (TMF)* and the *Transaction Monitoring Facility (TMF) Users Guide* for additional information about programming with TMF.

EXIT Statements

The EXIT statements mark the end of a procedure or the exiting point of a subprogram. The forms of the EXIT statements are:

```
EXIT
EXIT PROGRAM
```

Each form is described in the following paragraphs.

EXIT. The EXIT statement marks the end of a procedure. The statement performs no operation.

The syntax of the EXIT statement is:

```
EXIT .
```

EXIT PROGRAM. The EXIT PROGRAM statement marks the logical end of a called program. When this statement is executed in a called program, control returns to the calling program. If the program executing the EXIT PROGRAM statement is the initial program used when the terminal was started, the terminal is stopped.

The syntax of the EXIT PROGRAM statement is:

```
EXIT PROGRAM [ WITH ERROR ] .

where

  WITH ERROR

    is an option that provides a way to reassert the error condition described by special
    registers TERMINATION-STATUS and TERMINATION-SUBSTATUS. The error con-
    dition states that if a suspend class error is encountered, control is returned to the next
    higher level program unit having a CALL statement with an ON ERROR clause; if a pro-
    gram unit with the CALL...ON ERROR feature does not exist, the terminal is suspended
    without possibility of restart.

    Note that the program can change the contents of TERMINATION-STATUS and
    TERMINATION-SUBSTATUS before executing the EXIT PROGRAM WITH ERROR
    statement. Values for TERMINATION-STATUS must be in the range of 0 through 255.
```

The EXIT PROGRAM statement must appear in a sentence by itself and must be the only sentence in the paragraph.

GO TO Statements

The GO TO statements pass control from one part of the Procedure Division to another. The forms of the GO TO statements are:

```
GO TO
GO TO DEPENDING
```

Each form is described in the following paragraphs.

GO TO. The GO TO statement unconditionally passes control from one part of the Procedure Division to another.

The syntax of the GO TO statement is:

```
GO [ TO ] procedure-name

where

  procedure-name

    is the name of the procedure to which control is transferred.
```

GO TO DEPENDING. The GO TO DEPENDING statement passes control to one of several procedures depending on a variable data item.

The syntax of the GO TO DEPENDING statement is:

```
GO TO { procedure-name } ,... DEPENDING [ ON ] depend
```

where

procedure-name

is a series of procedure names. Only one is chosen, based on the value of *depend.*

depend

is the identifier of an elementary numeric integer data item. This item acts like an index because its value selects the procedure name to which the program branches. If the value of *depend* is outside the range of *procedure-name,* no branching occurs and control passes to the next statement.

The following example illustrates the GO TO DEPENDING statement:

```
procedure-branch.
     GO TO proc-1,
           proc-2,
           proc-3, DEPENDING ON branch-flag.
     MOVE 0 to branch-flag.
```

If *branch-flag* is 1, control passes to proc-1.
If *branch-flag* is 2, control passes to proc-2.
If *branch-flag* is 3, control passes to proc-3.
If *branch-flag* is less than 1 or greater than 3, control passes to the statement immediately following the GO TO DEPENDING statement.

IF Statement

The IF statement evaluates a condition and then transfers control depending on whether the value of the condition is true or false.

The syntax of the IF statement is:

```
IF condition  {statement-1   }    [ELSE  {statement-2   }]
              {NEXT SENTENCE}    [      {NEXT SENTENCE}]
```

where

condition

is any conditional expression.

→

`statement-1, statement-2`

are imperative or conditional statements. Each statement can contain an IF statement, in which case the statement is referred to as a nested IF statement.

`NEXT SENTENCE`

is a substitution for *statement-1* or *statement-2*. The phrase performs no operation, but is used to preserve the syntactical structure or to emphasize that one value of *condition* elicits no action.

IF statements within IF statements are considered as paired IF and ELSE statements, proceeding from left to right. An ELSE is assumed to apply to the immediately preceding IF that has not already been paired with an ELSE.

The following conventions apply to the IF statement:

If condition is true	*statement-1* is executed; if NEXT SENTENCE has been substituted for *statement-1*, no operation takes place.
If condition is false	*statement-2* is executed; if NEXT SENTENCE has been substituted for *statement-2* or if the ELSE clause has been omitted, no operation is performed.
If a GO TO statement that causes a transfer of control is executed as part of *statement-1* or *statement-2*	control is unconditionally transferred to the target of the GO TO.
If control is not unconditionally transferred by execution of a GO TO statement as part of *statement-1* or *statement-2*	control passes to the next executable statement following the IF after all statements executed as part of the IF have completed.

The following example illustrates a simple IF statement:

```
IF julian-days IS GREATER THAN 59,
   ADD leap-year TO julian-days.
```

The following example illustrates a simple IF ELSE statement:

```
IF tally GREATER THAN 0
   MOVE 0 TO tally
   MOVE 3 TO msg-index
   PERFORM print-error-routine
ELSE
   MOVE 1 TO flag.
```

The following example illustrates nested IF statements:

```
IF employee-number NOT EQUAL TO SPACES
    PERFORM read-routine
    IF no-error
       PERFORM list-record-out
       IF yes
          PERFORM delete-master
          IF no-error
             ADD 1 TO delete-count
          ELSE
             NEXT SENTENCE
       ELSE
          MOVE 0 TO flag
    ELSE
       NEXT SENTENCE
ELSE
   MOVE 1 TO flag.
```

MOVE Statements

The MOVE statements transfer data from one data item to one or more other data items in accordance with editing rules. The forms of the MOVE statements are:

```
MOVE
MOVE CORRESPONDING
```

Each form is described in the following paragraphs.

MOVE. The MOVE statement copies data from one data item and stores it in one or more other data items.

The syntax of the MOVE statement is:

```
MOVE data-name-1 TO { data-name-2 } ,...
```

where

data-name-1

is the sending item. The item can be an identifier or a literal.

data-name-2

is the receiving item. The item is an identifier.

Any subscripting or indexing for *data-name-1* is evaluated only once, immediately before data is moved to the first receiving item.

The following statement:

```
MOVE item-1(b) TO item-2, item-3(b)
```

is equivalent to:

```
MOVE item-1(b) TO temp
MOVE temp TO item-2
MOVE temp TO item-3(b).
```

The following examples illustrate the MOVE statement:

```
WORKING-STORAGE SECTION.
01 record-in.
   05  item-a     PIC X(5).
   05  item-b     PIC 99V99.
   77  temp1      PIC X(4).
   77  temp2      PIC X(8).
   77  temp3      PIC 9(5)V999.
   77  temp4      PIC 9V9.

PROCEDURE DIVISION.
begin-processing.
   MOVE item-a TO temp1.       <- Item-a is truncated to fit temp1.
   MOVE item-a TO temp2.       <- Remainder of temp2 is blank filled.
   MOVE item-b TO temp3.       <- Remainder of temp3 is zero filled.
   MOVE item-b TO temp4.       <- If the value in item-b is greater
   MOVE SPACES TO record-in.       than 9.9, this move will cause the
   MOVE ZEROS TO item-b.           SCREEN COBOL program to be
                                   suspended by the TCP with an
                                   arithmetic overflow error.
```

MOVE CORRESPONDING. The MOVE CORRESPONDING statement moves selected data items of one group to corresponding data items of another group.

The syntax of the MOVE CORRESPONDING statement is:

```
MOVE  {CORR          }  group-1 TO group-2
      {CORRESPONDING }
```

where

group-1

is the group name of sending data items.

group-2

is the group name of receiving data items.

Group-1 and *group-2* must be defined in the Working-Storage Section or the Linkage Section, not in the Screen Section.

The following conventions apply to data items used with the CORRESPONDING phrase:

- A REDEFINES or OCCURS clause can be specified in the data description entry of any data item.
- Data items can be subordinate to a data description entry with a REDEFINES or OCCURS clause.
- No data item can be defined with a level number 66, 77, or 88.

Subordinate data items in two different groups correspond to each other according to the following rules:

- Both data items must have the same data name.
- All possible qualifiers for the sending data item, up to but not including a group name, must be identical to all possible qualifiers for the receiving data item up to but not including the receiving group name.
- At least one of the corresponding sending/receiving items must be elementary. The class of any corresponding pair of data items can differ.
- Any data item subordinate to a data item that is not eligible for correspondence is ignored.
- FILLER data items are ignored.

Examples of corresponding items are shown in the following examples:

Example 1: All items in the following two groups correspond:

```
01  detail-in.
    05  social-security
    05  employee-name
    05  address
        10  street
        10  city
        10  state
        10  zip-code
```

```
01  report-line.
    03  social-security
    03  FILLER
    03  employee-name
    03  FILLER
    03  address
        05  street
        05  FILLER
        05  city
        05  FILLER
        05  state
        05  FILLER
        05  zip-code
```

The following sentence would fill in *report-line:*

```
MOVE CORRESPONDING detail-in TO report-line
```

Example 2: Only *pencils* items in the following groups correspond; even though all other elementary names are alike, they do not have the same qualifiers:

```
01  stock-items.
    05  erasers
        10  gum
        10  pink
        10  ink
    05  pencils
        10  mechanical
        10  non-mechanical
    05  felt-tip-pens
    05  ball-point-pens
    05  fountain-pens
```

```
01  shelf-items.
    05  pens
        07  felt-tip-pens
        07  ball-point-pens
        07  fountain-pens
    05  eradicators
        10  ink
        10  pink
        10  gum
    05  pencils
        10  mechanical
        10  non-mechanical
```

The following sentence would move only the data items of the *pencils* group:

```
MOVE CORRESPONDING stock-items TO shelf-items
```

MOVE RESTRICTIONS. Move operations between the following types of data items should not be attempted:

- An alphabetic data item or the figurative constant SPACE to a numeric data item.
- A numeric literal, a numeric data item, or the figurative constant ZERO to an alphabetic data item.
- A noninteger numeric literal or a noninteger numeric data item to an alphanumeric data item.
- A numeric data item to a numeric data item that does not have at least the same number of positions to the left of the decimal position.

MOVE CONVENTIONS. Data is converted and stored according to the data category of the receiving field. The conventions are as follows:

- Alphanumeric or alphabetic receiving data item
  - Data is stored beginning at the leftmost position in the receiving field.
  - If the data in the sending item is shorter, the data is space filled in the receiving field according to the standard alignment rules described in Section 2.
  - If the data in the sending item is longer, the data is truncated on the right to the length of the receiving field.
  - If the sending item is described as signed numeric, the operational sign is not moved to the receiving field; this applies whether the sign is a part of the data item or is a separate character.

- Numeric receiving item
  - Data is aligned by decimal point, zero filled as necessary.
  - If the receiving field is signed and the sending field is signed, the sign is moved and converted; if the sending field is not signed, the value is signed as positive.
  - If the receiving field is not signed, the absolute value of the sending field data is moved.
  - If the sending field is alphanumeric, the value of the sending field is treated as an unsigned numeric integer.

Group moves are treated as alphanumeric to alphanumeric moves, with no data conversion. The receiving area is filled without regard to individual or subgroup items in either the sending or receiving items.

Move conventions are summarized in Table 6-5.

Table 6-5. Move Summary Table

	Category of Receiving Data Item		
Category of Sending Data Item	**Alphabetic**	**Alphanumeric**	**Numeric Integer Numeric Noninteger**
Alphabetic	X	X	
Alphanumeric	X	X	X
Numeric Integer		X	X
Numeric Noninteger			X
X = legal			

MULTIPLY Statements

The MULTIPLY statements multiply two or more numeric items and place the result in a specified data item. A multiply operation can easily produce a value that does not fit into the receiving field; when defining a receiving field, thought should be given to the size of that field. The forms of the MULTIPLY statements are:

```
MULTIPLY BY
MULTIPLY GIVING
```

Each form is described in the following paragraphs.

MULTIPLY BY. The MULTIPLY BY statement multiplies one numeric data item by one or more other numeric data items. The product replaces the value of each multiplier.

The syntax of the MULTIPLY BY statement is:

```
MULTIPLY value BY { multiplier } ,...
```

where

value

is the multiplicand, which is a numeric literal or an identifier of an elementary numeric data item.

multiplier

is the identifier of an elementary numeric data item. The result of the multiply operation is stored as the new value of *multiplier*. The sum of the number of digits in value and *multiplier* must not exceed 18.

MULTIPLY GIVING. The MULTIPLY GIVING statement multiplies two numeric data items and stores the product in one or more other data items.

The syntax of the MULTIPLY GIVING statement is:

```
MULTIPLY value BY multiplier GIVING { result } ,...
```

where

value

is the multiplicand, which is a numeric literal or an elementary numeric data item.

multiplier

is a numeric literal or the identifier of an elementary numeric data item. The sum of the number of digits of *value* and *multiplier* must not exceed 18.

result

is the identifier of an elementary numeric data item into which the product is stored.

PERFORM Statements

The PERFORM statements execute one or more procedures in a program. When a single paragraph or section name is specified, control passes to the first statement of the paragraph or section; when execution of the paragraph or section completes, control passes to the PERFORM statement. If a group of paragraphs or procedures is specified, control passes to the first statement of the first paragraph or section; when execution of the last paragraph or section completes, control returns to the PERFORM statement.

The forms of the PERFORM statement are:

```
PERFORM
PERFORM TIMES
PERFORM UNTIL
PERFORM VARYING
PERFORM ONE
```

In each of these forms, the parameters *proc-1* and *proc-2* appear. *Proc-1* and *proc-2* have no special relationship; they represent a consecutive sequence of operations to be executed beginning at *proc-1* and ending with the execution of *proc-2*. GO TO and PERFORM statements can occur within the range of *proc-1* and *proc-2*. If two or more logical paths lead to the return point, *proc-2* could be a paragraph consisting of an EXIT statement, to which all of these paths must lead.

If control passes to these procedures by a means other than a PERFORM statement, control passes through the last statement of the procedure to the next executable statement as if no PERFORM statement referenced these procedures.

The range of a PERFORM statement is logically all those statements that are executed as a result of the PERFORM statement, through the transfer of control to the statement following the PERFORM statement. The range includes all statements executed as a result of a GO TO, PERFORM, or CALL statement in the range of the original PERFORM statement, as well as all statements in the Declaratives Section that might be executed. Statements in the range of a PERFORM are not required to appear consecutively.

If a sequence of statements referred to by a PERFORM statement includes another PERFORM statement, the sequence of procedures for the nested PERFORM must be either totally included in, or totally excluded from, the logical sequence referred to by the original PERFORM statement. Thus, an active PERFORM statement whose execution point begins within the range of another active PERFORM statement must not allow control to pass to the exit of the other active PERFORM statement. Furthermore, two or more such active PERFORM statements must not have a common exit.

Each form of the PERFORM statement is described in the following paragraphs.

PERFORM. The PERFORM statement executes a procedure, or group of procedures as established by the THROUGH phrase, one time. When execution completes, control passes to the statement following the PERFORM statement.

The syntax of the PERFORM statement is:

```
PERFORM proc-1    [ { THROUGH } proc-2 ]
                  [ { THRU    }        ]
```

where

proc-1 and proc-2

are the procedure paragraphs or sections to be executed.

The following example illustrates a PERFORM of one paragraph:

```
IF report-a
PERFORM do-report-a.
```

The following example illustrates a PERFORM of several paragraphs:

```
IF reports
     PERFORM do-reports THRU do-reports-exit.
     .
     .
     .

do-reports.
     .
     .
     .
  (several paragraphs to create the reports)
     .
     .
     .
do-reports-exit.
  EXIT.
```

PERFORM TIMES. The PERFORM TIMES statement executes a procedure, or group of procedures as established by the THROUGH phrase, a specified number of times. When the specified number of executions complete, control passes to the statement following the PERFORM TIMES statement.

The syntax of the PERFORM TIMES statement is:

```
PERFORM proc-1    [ { THROUGH } proc-2 ]  count TIMES
                  [ { THRU    }        ]
```

where

proc-1 and proc-2

are the procedure paragraphs or sections to be executed.

count

is an integer literal or the identifier of an integer data item. The procedure, or group of procedures, is executed as many times as the value of *count*.

The following example illustrates the PERFORM TIMES statement:

```
PERFORM list-transactions 2 TIMES.
```

PERFORM UNTIL. The PERFORM UNTIL statement executes a procedure, or group of procedures as established by the THROUGH phrase, based on a condition. The condition is checked before each PERFORM cycle. When the condition is met, control passes to the statement following the PERFORM UNTIL statement.

The syntax of the PERFORM UNTIL statement is:

```
PERFORM proc-1   [ { THROUGH } proc-2 ]  UNTIL condition
                 [ { THRU    }        ]
```

where

proc-1 and proc-2

are the procedure paragraphs or sections to be executed.

condition

is any conditional expression.

The following example illustrates the PERFORM UNTIL statement:

```
WORKING-STORAGE SECTION.
01  flag                PIC 9.
    88  bad                     VALUE 0.
    88  good                    VALUE 1.
    88  no-more-adds            VALUE 1.
          .
          .
          .
PROCEDURE DIVISION.
          .
          .
          .
   PERFORM add-routine UNTIL no-more-adds.
          .
          .
          .
add-routine.
   MOVE 0 to flag.
          .
          . Once the add routine is successful, a 1 is moved to
          . flag; otherwise, flag remains 0. As long as flag is
          . 0, the procedure is reexecuted.
delete-routine.
```

PERFORM VARYING. The PERFORM VARYING statement executes a procedure, or group of procedures as established by the THROUGH phrase, while varying a data item until specified conditions are true. When AFTER phrases are specified, the range is within a nested loop. The innermost loop is defined by the last AFTER phrase; the outermost loop is defined by the first set of parameters in the VARYING clause. When execution completes, control passes to the statement following the PERFORM VARYING statement.

The syntax of the PERFORM VARYING statement is:

```
PERFORM proc-1    [ { THROUGH } proc-2 ]
                  [ { THRU    }        ]

 VARYING vary-1 FROM base-1 BY step-1 UNTIL condition-1

 [ AFTER vary-2 FROM base-2 BY step-2 UNTIL condition-2 ] ...
```

where

proc-1 and proc-2

are the procedure paragraphs or sections to be executed.

vary-1 and vary-2

are the identifiers of integer numeric data items.

base-1 and base-2

are integer numeric literals or identifiers of numeric data items.

step-1 and step-2

are integer numeric literals or identifiers of numeric data items. Their value must not be zero.

condition-1 and condition-2

are any conditional expressions.

The following example illustrates the PERFORM VARYING statement:

```
WORKING-STORAGE SECTION.
01  command-data.
    05  FILLER  PIC X(36)  VALUE "ADD      - ADD A NEW RECORD".
    05  FILLER  PIC X(36)  VALUE "DELETE   - DELETE A RECORD".
              .
              .
              .
01  command-table REDEFINES command-data.
    05  command-entry   PIC X(36)   OCCURS 10 TIMES.

77  no-of-commands    PIC 99   VALUE 9.
77  command-index     PIC 99   VALUE 1 COMP.

PROCEDURE DIVISION.

PERFORM list-commands VARYING command-index FROM 1 BY 1
    UNTIL command-index GREATER THAN no-of-commands.

list-commands.
    .
    .
    .
    .
```

PERFORM ONE. The PERFORM ONE statement executes just one procedure, or one group of procedures as established by the THROUGH phrase, as determined by the value of an identifier.

The syntax of the PERFORM ONE statement is:

```
PERFORM ONE [ OF ]  { proc-1  [ { THROUGH } proc-2 ] }  ,...
                                { THRU    }
 DEPENDING [ ON ] identifier
```

where

proc-1 and proc-2

are the procedure paragraphs or sections to be executed.

identifier

is an integer numeric literal or the identifier of an integer data item. The value determines which procedure, or group of procedures, is to be performed.

Each procedure, or group of procedures, in the list is assigned an index; the index indicates the position of the procedure, or group of procedures, in the list. If the value of the *identifier* matches one of these indexes, the procedure, or group of procedures, with that index is executed. When execution completes, control passes to the statement following the PERFORM ONE statement. If the value of *identifier* does not match any procedure index, no procedures are executed.

The following example illustrates the PERFORM ONE statement:

```
PERFORM ONE OF
        A THRU B
        C
        D
    DEPENDING ON I.
```

PRINT SCREEN Statement

The PRINT SCREEN statement causes the current screen image to be printed on an attached or non-attached printer.

The syntax of the PRINT SCREEN statement is:

```
PRINT SCREEN [ ON ERROR imperative-statement ]
```

where

ON ERROR

provides a point of control if an error occurs while attempting the print operation. The special register TERMINATION-STATUS is set with an error code indicating the type of error; the *imperative-statement* is then executed.

If the clause is omitted and an error occurs, default system action is taken.

imperative-statement

is the statement to be executed if an error is detected.

If an attached printer has been specified via the PATHCOM SET TERM command for the T16-6520 terminal, the screen image is directed to a printer attached directly to the terminal (T16-6524 is the terminal with an attached printer). If an attached printer has been specified via the PATHCOM SET TERM command for the IBM-3270 terminal, the screen image is directed to the device specified in the special register TERMINAL-PRINTER; this device must be attached to the same control unit as the terminal.

If a printer is not attached, printing occurs on the GUARDIAN file specified in the special register TERMINAL-PRINTER. Device types terminal, printer, and process are supported. This type of print operation does not read the screen to form the screen image to be printed; instead, the print operation forms the screen image by using the screen description contained in the SCREEN COBOL object file and the working storage items associated with the screen. The use of working storage items as the basis for data fields poses three problems:

- It is possible that the value in the working storage item has never been displayed or contains data that is different from what is presently displayed on the screen; the program must take care to ensure that intended results occur.
- Direct displays, such as DISPLAY "XYZ" IN SCREEN-FIELD, have no effect on working storage.
- If a screen field has an associated FROM field and a different associated TO field, an anomaly exists. The PRINT SCREEN statement resolves this by assigning the following precedence when selecting associated working storage items:

  highest —> USING association
  —> TO association
  lowest —> FROM association

The TERMINATION-STATUS error codes set by the PRINT SCREEN statement are listed in Table 6-6.

Table 6-6. TERMINATION-STATUS Error Codes for PRINT SCREEN Statement

TERMINATION-STATUS	Meaning
Error code 0	No error has occurred.
Error code 1	The base screen is not displayed. Default system action: The terminal is suspended without possibility of restart, and a message describing the error is logged to the log file.
Error code 2	The printer specification is invalid for the terminal type, or the printer device type is not supported: the IS-ATTACHED modifier of the PATHCOM SET TERM PRINTER command is specified for T16-6510; the IS-ATTACHED modifier is specified for IBM-3270 and the printer is not attached to the same controller as the terminal; or a file having a device type other than terminal, printer, or process has been specified.

Table 6-6. TERMINATION-STATUS Error Codes for PRINT SCREEN Statement (Continued)

TERMINATION-STATUS	Meaning
Error code 2 (continued)	Default system action: The terminal is suspended without possibility of restart, and a message describing the error is logged to the log file.
Error code 3	The printer requires attention (for example, it is in NOT READY state). During I/O to the printing device, a GUARDIAN file error code indicating the device requires human intervention was returned. Default system action: If the special register DIAGNOSTIC-ALLOWED is set to YES, a diagnostic screen informing the terminal operator of the condition is displayed. If the terminal operator presses the T16-6510 or T15-6520 F1 key (or equivalent IBM-3270 key), the terminal operator has corrected the condition; screen recovery is invoked, then the copy is restarted from the beginning. If the terminal operator presses the T16-6510 or T16-6520 F2 key (or equivalent IBM-3270 key), the print screen operation is aborted, a diagnostic screen indicating the permanent condition is displayed, the terminal is suspended with the possibility of restart, and a message describing the error is logged to the log file. If the special register DIAGNOSTIC-ALLOWED is set to NO, the terminal is suspended with the possibility of restart, and a message describing the error is logged to the log file.
Error code 4	A fatal error has occurred. During I/O to the device, a GUARDIAN file error code indicating a fatal error condition was returned. Default system action: The terminal is suspended with the possibility of restart, and a message describing the error is logged to the log file.

I/O PERFORMED BY THE PRINT SCREEN STATEMENT. The PRINT SCREEN I/O sequence begins with a top-of-form operation. Each screen line is written in a separate record; trailing blanks and trailing null values are suppressed. Printing starts with the line at the top of the screen and proceeds through the line at the bottom of the screen.

DIAGNOSTIC SCREENS. A diagnostic screen, which is described in Appendix A, can be displayed when an error occurs during a PRINT SCREEN sequence. An example of the default diagnostic screen is shown in Figure 6-3.

```
PATHWAY ERROR REPORT: 04MAY82,12:42
TERMINAL: TERM-1

PRINTER REQUIRES ATTENTION
  PRINTER: $LP
  PRESS F1 TO RETRY, F2 TO ABORT
```

Figure 6-3. Sample Diagnostic Screen

IBM-3270 ATTACHED PRINTERS. To permit a screen on an IBM-3270 terminal to be printed, an input field must be declared starting at screen position 1,2. If a protected field is in this position, the screen is locked for screen copy operations and a PRINTER I/O ERROR (179) occurs.

The destination device of a PRINT SCREEN operation must have a device type of 10 and must use the CRT protocol of the AM3270 access method. Refer to the *AXCESS Data Communications Programming Manual* for additional information.

RECONNECT MODEM Statement

The RECONNECT MODEM statement gives a SCREEN COBOL program control of the connection to a PATHWAY terminal across a dial-in switched line (a standard communication line used by the public telephone system). PATHWAY does not support a dial-out capability over a switched line.

The syntax of the RECONNECT MODEM statement is:

```
RECONNECT MODEM
```

If the connection to the PATHWAY terminal is over a switched line, the RECONNECT MODEM statement breaks the terminal's connection with the SCREEN COBOL program and causes the program to wait for another incoming call. After the next incoming call completes connection to the terminal, the SCREEN COBOL program resumes execution at the next program instruction.

If a RECONNECT MODEM statement is executed while the PATHWAY terminal is not connected over a switched line, the program resumes immediately at the next program instruction.

After a RECONNECT MODEM statement is executed, all screen definitions are lost. A DISPLAY BASE statement must precede the next screen operation.

RECONNECT MODEM lets a SCREEN COBOL program perform the following operations for PATHWAY terminals connected over switched lines:

- disconnect the terminal at the end of a terminal session (the caller does a logoff)
- recover from a modem error (an accidental hang up or line disconnect), and wait for the next terminal to call.

Each terminal caller should access the SCREEN COBOL program in a consistent state. You should initialize local variables and have previous transactions completed before the RECONNECT MODEM statement executes.

The RECONNECT MODEM statement causes a full context checkpoint. If PATHWAY is running under TMF and a terminal is in transaction mode, this statement causes the current transaction to be backed out and suspends the terminal such that the terminal cannot be resumed. If the ABORT-TRANSACTION statement precedes the RECONNECT MODEM statement, PATHWAY can attempt to resume terminal communications after a modem error.

The following example illustrates the RECONNECT MODEM statement:

```
START-PROGRAM.
  CALL SEARCH-PROGRAM ON ERROR GO TO VERIFY-RECONNECT.
  RECONNECT MODEM.
  GO TO START-PROGRAM.

VERIFY-RECONNECT.
  IF TERMINATION-STATUS IS = 18 AND
     TERMINATION-SUBSTATUS IS = 140
*  This is a modem error - return to a consistent state and
*  wait for the next terminal caller.
     DELAY 10
     RECONNECT MODEM
     GO TO START-PROGRAM.
*  Processes other error conditions.

  DISPLAY BASE SEARCH-SCREEN.
    .
    .
```

RESET Statement

The RESET statement restores the display attributes and/or the data of screen fields to the states declared in the screen definition. The statement restores only the terminal display, not the internal data.

The syntax of the RESET statement is:

```
RESET [ TEMP      ]   [ ATTR ]  {screen-identifier} ,...
      [ TEMPORARY ]   [ DATA ]

  [ DEPENDING [ ON ] identifier ]
  [ SHADOWED                    ]
```

where

TEMP or TEMPORARY

specifies that the selected fields are to be reset only if they have received their current values or attributes from a DISPLAY TEMP or TURN TEMP statement.

ATTR

resets the display attributes of the selected fields to the value specified in the screen definition.

DATA

resets the characters displayed in the selected fields to the value specified in the VALUE field characteristic clause of the field. If a value is not specified, the standard fill character fills the field.

If neither ATTR nor DATA is specified, both the attributes and data of the selected fields are reset to initial values.

NOTE

If the display is for a terminal operating in conversational mode, RESET DATA has no effect. Either RESET ATTR or RESET TEMP ATTR must be specified to reset the display attributes.

→

```
screen-identifier
```

specifies the fields to be RESET. Each identifier can be the name of an entire screen, a screen group, or an elementary item of any base or overlay screen that is currently displayed. If *screen-identifier* is not an elementary item, all subordinate elementary items that have a TO, FROM, or USING clause in their definitions are reset.

```
DEPENDING ON identifier
```

selects zero or one *screen-identifier* from the list. The statement whose position in the *screen-identifier* list is the same as the value in *identifier* is selected. If the value in *identifier* is less than one or greater than the number of *screen-identifiers*, no *screen-identifier* is selected.

```
SHADOWED
```

selects from the *screen-identifier* list only those fields that have SHADOWED items in which the SELECT bit is set; fields that do not have SHADOWED items are not selected.

If neither the DEPENDING ON modifier nor the SHADOWED modifier is specified, all fields in the *screen-identifier* list are selected.

When the RESET statement is executed, the attributes and/or data of the selected fields are reset to their initial values.

RESTART-TRANSACTION Statement

The RESTART-TRANSACTION statement restarts the transaction of a terminal operating in transaction mode. Transaction mode is an operating mode in which PATHWAY servers that are configured to run under the Transaction Monitoring Facility (TMF) can lock and update audited files.

Execution of this statement indicates the current attempt to perform the transaction failed because a transient problem occurred. The statement requests TMF to back out any updates made on a data base during this transaction; terminal execution resumes at the BEGIN-TRANSACTION statement. TMF assigns a new transaction ID number to the terminal; the TCP marks the screen for screen recovery and increments by 1 the special register RESTART-COUNTER. The special register TERMINATION-STATUS remains at 1.

The syntax of the RESTART-TRANSACTION statement is:

```
RESTART-TRANSACTION
```

The execution of this statement can cause suspension of a terminal for a pending abort for two reasons:

1. The terminal is not in transaction mode when this statement executes.
2. A fatal error occurs while attempting to back out the updates made on the data base.

Refer to the *Introduction to Transaction Monitoring Facility (TMF)* and the *Transaction Monitoring Facility (TMF) System Management and Operations* Guide for NonStop Systems or the *Transaction Monitoring Facility (TMF) System Management and Operations Guide* for NonStop II Systems for additional information about programming with TMF.

SCROLL Statement

The SCROLL statement moves the contents of an overlay area up or down. This statement can be used only with the T16-6510 terminal.

The syntax of the SCROLL statement is:

```
SCROLL  { UP   }  overlay-area-name
        { DOWN }
```

where

UP

moves the data displayed in the overlay area of the screen up one line toward the top of the screen. A blank line appears at the bottom of the overlay area, and the top line in the overlay area is lost.

DOWN

moves the data displayed in the overlay area of the screen down one line toward the bottom of the overlay. A blank line appears at the top of the overlay area, and the last line in the overlay area is lost.

overlay-area-name

is the name of the screen overlay area. The overlay screen associated with the area can contain only output or literal fields. Literal fields are displayed only when the overlay screen is initially displayed in the area.

SEND Statement

The SEND statement sends a transaction request message to a server process and receives a reply from that process.

The syntax of the SEND statement is:

```
SEND [ identifier-1] ,... TO server-class-name
 [ UNDER PATHWAY pathmon-name]
 [ AT SYSTEM system-name]
 REPLY { CODE { reply-code-value } ...
 YIELDS { identifier-2 } ...} ...
 [ ON ERROR imperative-statement ]
```

where

identifier-1

is a data item to be sent to the server. The data item represented by this identifier cannot exceed 2047 bytes. If *identifier-1* is a variable length data item, the SEND statement sends only the currently defined occurrence (a variable length data item is an item defined with an OCCURS DEPENDING ON clause).

If this parameter is omitted, zero bytes are sent to the server. A reply will still be returned from the server. →

`server-class-name`

identifies the server class for which the message is intended. The *server-class-name* can be a nonnumeric literal or a data item. The size of the field containing *server-class-name* can be 1 through 15 characters.

This is the logical server class name used in the PATHCOM ADD SERVER command.

`pathmon-name`

is the name of the PATHMON process that controls the links to the server class named in the *server-class-name* parameter above. The *pathmon-name* can be a nonnumeric literal or a data item. The size of the field containing *pathmon-name* can have 15 characters, but the TCP will pass only the first five characters in network communications.

A SEND statement directed through an external PATHMON must specify a valid network process name. The value specified for *pathmon-name* must begin with a $ and can be followed by one to four alphanumeric characters.

If this parameter is omitted, the PATHMON that controls the server class is assumed to have the same name as the PATHMON process that controls the TCP.

`system-name`

is the name of the Tandem system on which the above named PATHMON is running. The *system-name* can be a nonnumeric literal or a data item. The field containing *system-name* can have 15 characters, but the TCP will pass only the first eight characters in network communications.

The value specified for *system-name* must be a valid network system name that begins with a \ and can be followed by one to seven alphanumeric characters.

If this parameter is omitted, the Tandem system name of the PATHMON that controls the server class is assumed to be the same as the system name of the PATHMON that controls the TCP.

`reply-code-value`

is an integer literal or integer data item that specifies an expected reply code from the server.

`identifier-2`

is a data name into which a portion of the contents of the reply message is to be placed.

`ON ERROR`

provides a point of control if an error occurs in sending the message.

If this clause is omitted and an error is detected, standard system action is performed. Depending on the error, system action involves either waiting for a resource to become available or suspending execution of the program.

`imperative-statement`

is the statement to be executed if an error is detected.

→

Specifying *reply-code-value* after the CODE keyword identifies the structure of the reply. When the send operation receives the reply from the server, the first two bytes are interpreted as a 16-bit integer. This code must match one of the CODE reply code values. The entire reply is then distributed to the items in *identifier-2* list associated with *reply-code-value*. The special register TERMINATION-STATUS is set to a number corresponding to the position of the particular reply code value in the list. Each *reply-code-value* corresponds to a unique number setting for TERMINATION-STATUS whether or not a reply code value yields the same working storage data item. If there is no match or if the reply message data does not exactly fill the data items in the *identifier-2* list, an error is indicated.

In the SEND statement, the position of a reply code affects the value set for the TERMINATION-STATUS special register as illustrated in the following example:

```
.
.
SEND HEADER, LASTNAME OF EMP-REC TO "PERS-DEPT"
     REPLY CODE 1, 21,  31 YIELDS R-CODE, NEW-SALARY
           CODE 2, 42,  62 YIELDS NEW-RATE, STOCK-OPTION, BENE-FIT
           CODE 0, 200    YIELDS TERMINATION-NOTICE
     ON ERROR PERFORM SERVER-LIST.
.
.
```

In this example, the positions of the reply codes cause these corresponding values to be set for TERMINATION-STATUS:

REPLY CODE	TERMINATION-STATUS
1	1
21	2
31	3
2	4
42	5
62	6
0	7
200	8

Consider the following example of the SEND statement:

```
77 YEARLY-REVIEW PIC 999 VALUE 3.

MOVE YEARLY-REVIEW TO TRANSCODE OF HEADER.
SEND HEADER, LASTNAME OF PERSONAL-REC TO "SALARY-UPDATE"
     REPLY CODE 1 YIELDS  R-CODE, NEW-SALARY
           CODE 2 YIELDS  R-CODE, NEW-SALARY, STOCK-OPTION
           CODE 0 YIELDS  R-CODE, TERMINATION-NOTICE
     ON ERROR PERFORM SERVER-DUMB.
```

This example is executed as follows:

1. The transaction message is constructed using the values of HEADER and LASTNAME from the SCREEN COBOL program data area. This message is sent to a server process of the server class SALARY-UPDATE and the requester waits for a reply.

2. When the reply arrives, the reply is identified and is moved into NEW-SALARY, NEW-SALARY and STOCK-OPTION, or TERMINATION-NOTICE, depending on the reply code. The number moved into special register TERMINATION-STATUS will be 1, 2, or 3, depending on the reply code interpreted from the server.

3. The ON ERROR clause takes special action in case a problem occurs in sending the message. The possible problems include a freeze being in effect on the server class, the unavailability of an appropriate server, and an unrecognizable reply from the server. If such a condition arises, TERMINATION-STATUS is set to a value indicating the type of error and the imperative statement PERFORM SERVER-DUMB is executed.

4. If the ON ERROR clause had not been included and an error occurred, the standard system action would be performed.

The following program example illustrates two ways you can use the SEND statement to access a server class controlled by an external PATHMON (a PATHMON in a different PATHWAY system from the requesting TCP).

```
      .
      .
DATA DIVISION.
 WORKING-STORAGE SECTION.
      .
      .
  01 WS-DEFAULT-NAMES.
     05 WS-DEFAULT-SERVER        PIC X(15) VALUE "SERV-1".
     05 WS-DEFAULT-PATHMON       PIC X(5) VALUE "$PWT".
     05 WS-DEFAULT-SYSTEM        PIC X(8) VALUE "\TS".
      .
      .
  01 WS-SCRN1-FIELDS.
     05 WS-SERV-NAME             PIC X(15) VALUE " ".
     05 WS-SCRN-PATHMON          PIC X(5) VALUE " ".
     05 WS-SCRN-SYSTEM           PIC X(8) VALUE " ".
      .
      .
PROCEDURE DIVISION.
      .
      .
 SEND MSGID, EMPLOYEE-REC TO "SERV-1"
      UNDER PATHWAY "$PWT"
      AT SYSTEM "\TS"
      REPLY CODE 1 YIELDS R-CODE, EMPLOYEE-REC
            CODE 2 YIELDS R-CODE, HIRE-DATE
      ON ERROR PERFORM 899-SEND-ERROR.
      .
      .
 MOVE WS-DEFAULT-SERVER TO WS-SERV-NAME.
 MOVE WS-DEFAULT-PATHMON TO WS-SCRN-PATHMON.
 MOVE WS-DEFAULT-SYSTEM TO WS-SCRN-SYSTEM.
      .
      .
 SEND MSGID, EMP-TRANSFER TO WS-SERV-NAME
      UNDER PATHWAY WS-SCRN-PATHMON
      AT SYSTEM WS-SCRN-SYSTEM
      REPLY CODE 1 YIELDS R-CODE, EMPLOYEE-LOC
            CODE 2 YIELDS R-CODE,
            CODE 3 YIELDS R-CODE
      ON ERROR PERFORM 899-SEND-ERROR.
      .
      .
```

Errors that can occur during execution of a SEND statement are listed in Table 6-7. The number given is the special register TERMINATION-STATUS value returned if the SEND statement includes the ON ERROR clause.

Table 6-7. SEND Statement Errors

Number	Message	Meaning	Action Without ON ERROR Clause
1	SERVER CLASS FROZEN	The server class to which the message is directed is frozen.	The system waits until the server class is thawed by execution of a PATHCOM THAW SERVER statement, then continues with SEND statement processing.
2,3	RESOURCE UNAVAILABLE	A free control block cannot be found, generally because the initial configuration of the TCP did not provide enough control blocks.	The system waits until the resource is available, then continues with the send operation.
4	LINK DENIED	The request to PATHCOM for a link to the server class has been denied for indeterminate reasons, and the TCP has no previously established links to the class.	The system periodically rerequests a link and when successful, continues with the send operation.
5	SERVER CLASS UNDEFINED	The request to PATHMON for a link to the server class has been denied because no class of that name has been defined.	The system periodically rerequests a link. When the server class is added to the configuration, the send operation continues.
6	ILLEGAL SERVER CLASS NAME	The value given for the name of the server class does not have the format of a valid server class name.	The system suspends the terminal with a fatal error.
7	MESSAGE TOO LARGE	The message to the server is larger than allowed by the configuration of the TCP.	
8	MAXIMUM REPLY TOO LARGE	The size of one or more replies specified in the SEND statement would be larger than allowed by the configuration of the TCP.	
9	Unused		

Table 6-7. SEND Statement Errors (Continued)

Number	Message	Meaning	Action Without ON ERROR Clause
10	Undefined Reply	The reply code found in the reply from the server does not match any codes specified in the SEND statement.	The system suspends the terminal. If the terminal is resumed by execution of a PATHCOM RESUME statement, the send operation will be reattempted. If the terminal is in transaction mode, the transaction is backed out and the terminal is suspended. If terminal execution is resumed, the transaction is restarted.
11	REPLY LENGTH INVALID	The length of the reply received from the server is not equal to the length implied by the selected YIELDS list.	
12	I/O ERROR	A file system error occurred during the WRITEREAD to the server, or a timeout on the server occurred.	
13	TRANSACTION MODE VIOLATION	A send to a non-TMF server was attempted while the terminal was in transaction mode.	The system suspends the terminal for pending abort.
14	NO PMCB AVAILABLE	The request requires the TCP to communicate with an external PATHMON, but the maximum number of PATHMON processes the TCP can communicate with has been reached. The value specified in the SET TCP MAXPATHWAYS parameter sets this maximum.	The system suspends the terminal and an error message is sent to the PATHWAY log file.
15	UNDEFINED SYSTEM	The system name given is not known to the network	
16	ILLEGAL SYSTEM NAME	The value given for *system-name* does not have the correct format. For example, the first character is not a \.	
17	ILLEGAL PATHMON NAME	The value given for *pathmon-name* does not have the correct format. For example, the first character is not a $.	
18	PATHMON I/O ERROR	An I/O error occurred during the OPEN or WRITEREAD message to an external PATHMON.	

SET Statement

The SET statement stores the position of the indicated screen field into the special register NEW-CURSOR. This value, which could be further modified by the program, is used at the beginning of the next ACCEPT statement to establish the position of the cursor on the screen. This value also can be examined by the program for some specific purpose.

The syntax of the SET statement is:

```
SET NEW-CURSOR AT { screen-identifier } ,...

   [DEPENDING [ ON ] identifier]
   [SHADOWED                    ]
```

where

`screen-identifier`

specifies the fields whose positions are to be stored. Each *identifier* can be the name of an entire screen, a screen group, or an elementary item of any base or overlay screen that is currently displayed. If *screen-identifier* is not an elementary item, all subordinate elementary items that have a TO, FROM, or USING phrase in their definitions are referenced.

The default cursor position is the first screen field defined with a TO or USING clause for the current ACCEPT statement.

`DEPENDING ON identifier`

selects zero or one *screen-identifier* from the list. The statement whose position in the *screen-identifier* list is the same as the value in *identifier* is selected. If the value in *identifier* is less than one or greater than the number of *screen-identifiers*, no *screen-identifier* is selected.

`SHADOWED`

selects from the *screen-identifier* list only those fields that have SHADOWED items in which the SELECT bit is set; fields that do not have SHADOWED items are not selected.

If neither the DEPENDING ON modifier nor the SHADOWED modifier is specified, all fields in the *screen-identifier* list are selected.

The SET statement selects a field in sequence from top to bottom and left to right. For example, a field having the lowest row (line) number is selected before a field with a higher row number. For fields in the same row the field having the lowest column number is selected before the field with a higher column number.

The SET statement places the row and column numbers of the leftmost character of the first selected field into the special-register NEW-CURSOR. The implied structure of NEW-CURSOR is as follows:

```
01  NEW-CURSOR.
    02 NEW-CURSOR-ROW  PIC 9999 COMP.
    02 NEW-CURSOR-COL  PIC 9999 COMP.
```

If the value specified in the special register NEW-CURSOR is not a valid screen position when an accept operation begins, the cursor is positioned to the first unprotected field of the ACCEPT statement. After execution of an ACCEPT statement, the special register is set to zero; this causes the cursor position for the next ACCEPT statement to be the first field of that ACCEPT statement.

STOP RUN Statement

The STOP RUN statement causes the executing program to stop immediately after this statement executes.

The syntax of the STOP RUN statement is:

```
STOP RUN
```

If the executing program is a called program unit and a STOP RUN statement is executed, control does not return to the calling program.

SUBTRACT Statements

The SUBTRACT statements subtract elementary numeric data items and set the results in specific data items. The forms of the SUBTRACT statements are:

```
SUBTRACT
SUBTRACT GIVING
SUBTRACT CORRESPONDING
```

Each form is described in the following paragraphs.

SUBTRACT. The SUBTRACT statement totals all data items before keyword FROM and then subtracts that sum from the current value of each data item after keyword FROM.

The syntax of the SUBTRACT statement is:

```
SUBTRACT { sub-1 } ,... FROM { sub-2 } ,...
```

where

sub-1

is either a numeric literal or the identifier of an elementary numeric data item.

sub-2

is the identifier of an elementary numeric data item.

SUBTRACT GIVING. The SUBTRACT GIVING statement is the same as the SUBTRACT statement, except the result is stored in separate data items.

The syntax of the SUBTRACT GIVING statement is:

```
SUBTRACT { sub-1 } ,... FROM sub-2 GIVING { result } ,...
```

where

sub-1

is either a numeric literal or the identifier of an elementary numeric data item.

sub-2

is either a numeric literal or the identifier of an elementary numeric data item.

result

is the identifier of an elementary numeric data item.

SUBTRACT CORRESPONDING. The SUBTRACT CORRESPONDING statement subtracts elementary items in one group from any corresponding items in another group. Items correspond when they have the same names and qualifiers up to but not including the group item name specified in the SUBTRACT CORRESPONDING statement.

The syntax of the SUBTRACT CORRESPONDING statement is:

```
SUBTRACT  {CORR          }  group-1 FROM group-2
          {CORRESPONDING }
```

where

group-1 and group-2

are the identifiers of group items in which some or all of the elementary items are numeric.

The results are placed in the *group-2* items.

The following conventions apply to data items used with the CORRESPONDING phrase:

- A REDEFINES or OCCURS clause can be specified in the data description entry of any data item.
- Data items can be subordinate to a data description entry with a REDEFINES or OCCURS clause.
- No data item can be defined with a level number 66, 77, or 88.

Subordinate data items in two different groups correspond to each other according to the following rules:

- Both data items must have the same data name.
- All possible qualifiers for the sending data item, up to but not including a group name, must be identical to all possible qualifiers for the receiving data item up to but not including the receiving group name.
- Only elementary numeric data items are considered.
- Any data item subordinate to a data item that is not eligible for correspondence is ignored.
- FILLER data items are ignored.

Refer to the ADD CORRESPONDING statement for examples of corresponding items.

TURN Statement

The TURN statement changes the display attributes of screen variable fields.

The syntax of the TURN statement is:

```
TURN  [TEMP      ]  {mnemonic-name                                  }   IN
      [TEMPORARY ]  {RECEIVE FROM  {ALTERNATE                     }}
                                   {ALTERNATE OR TERMINAL         }
                                   {TERMINAL                      }
                                   {TERMINAL OR ALTERNATE         }

 { screen-identifier } ,...  [DEPENDING [ ON ] identifier]
                             [SHADOWED                   ]
```

where

`TEMP or TEMPORARY`

indicates the fields are to be reset to their initial display attributes when the next RESET TEMP or ACCEPT statement is executed.

`mnemonic-name`

specifies the display attributes to be used. The *mnemonic-name* must be associated with one or more display attributes through an IS phrase in the SPECIAL-NAMES paragraph in the Environment Division of the program.

```
RECEIVE FROM  {ALTERNATE                }
              {ALTERNATE OR TERMINAL    }
              {TERMINAL                 }
              {TERMINAL OR ALTERNATE    }
```

specifies the type of device from which data can be accepted for a screen field. This clause is supported only for applications running on Tandem 6530 terminals with Tandem 6AI (revision A00) firmware.

`screen-identifier`

specifies the fields whose attributes can be changed. Each identifier can be the name of an entire screen, a screen group, or an elementary item of any base or overlay screen that is currently displayed. If *screen-identifier* is not an elementary item, all subordinate elementary items that have a TO, FROM, or USING clause in their definitions are referenced.

`DEPENDING ON identifier`

selects zero or one *screen-identifier* from the list. The statement whose position in the *screen-identifier* list is the same as the value in *identifier* is selected. If the value in *identifier* is less than one or greater than the number of *screen-identifiers*, no *screen-identifier* is selected. →

SHADOWED

selects from the *screen-identifier* list only those fields that have SHADOWED items in which the SELECT bit is set; fields that do not have SHADOWED items are not selected.

If neither the DEPENDING ON modifier nor the SHADOWED modifier is specified, all fields in the *screen-identifier* list are selected.

The attributes of the selected fields are changed to those specified by *mnemonic-name*. The settings for attributes not specified by *mnemonic-name* are determined by the initial attributes of the field.

USE Statement

The USE statement is used in the DECLARATIVES portion of the Procedure Division. This statement marks a procedure as the one to restore the terminal display when errors occur while the specified base screens are active. Screen recovery might be necessary during any screen operation due to terminal or communication line errors, processor failure, or terminal suspensions.

The syntax of the USE statement is:

```
USE [ FOR SCREEN ] RECOVERY

 [ ON { base-screen-name-n } ,... ] .
```

where

ON base-screen-name-n

specifies the base screens for which the declarative procedures are to be used.

If this phrase is omitted, the declarative procedures are used for all screens not mentioned in another USE statement.

The recovery process performs the equivalent of a DISPLAY BASE for the current base screen followed by a DISPLAY OVERLAY for all currently active overlay screens. The applicable declarative procedure statements are then executed. When the declarative procedures complete execution, control returns to the statement that was being executed when the problem was detected; the statement is executed again.

The USE statement must immediately follow a section header in the DECLARATIVES portion of the PROCEDURE DIVISION.

The following example illustrates the USE statement:

```
PROCEDURE DIVISION.
DECLARATIVES.
S-R SECTION.  USE FOR SCREEN RECOVERY.
RECOV-1.
  Move "SCREEN RECOVERY" TO error-msg,
  DISPLAY msg-1.
END DECLARATIVES.
MAIN SECTION.
```

SECTION 7

COMPILATION

The SCREEN COBOL compiler is called with the SCREEN COBOL run command. The name to be used on the compiler run is SCOBOL. If all default parameters are selected, the compiler run appears as

```
SCOBOL
```

USING THE COMPILER

The SCREEN COBOL source program is usually run from the Command Interpreter. The syntax is:

```
SCOBOL [ / [ IN source-file ] [ , OUT [ list-file ] ]
           [ , run-option ] ] / ] [ tclprog-file ] [ , copy-library ]
       [ ; compiler-command ] [ , compiler-command ] ...
```

where

`source-file`

is a Tandem file containing SCREEN COBOL statements and compiler commands. The SCREEN COBOL compiler reads *source-file* as 132-byte records.

If this parameter is omitted, input is taken from the current input file of the Command Interpreter; this is typically the home terminal.

`OUT list-file`

directs listing output to a named file that is of the same form as *source-file*. If *list-file* is an unstructured disc file, each *list-file* record is 132 characters (partial lines are blank filled through column 132).

If *list-file* is omitted and OUT is present, no listing is produced.

If the entire option is omitted, the listing output is directed to the Command Interpreter OUT file; this is typically the home terminal.

→

`run-option`

is one of following GUARDIAN command options that can be specified when running a SCREEN COBOL program:

`IN file-name`	input files
`OUT file-name`	output files
`NAME $process-name`	symbolic process name
`CPU cpu-number`	processor module
`PRI priority`	execution priority
`MEM num-pages`	maximum number of data pages
`NOWAIT`	Command Interpreter does not suspend while the program runs

For more information about these options, refer to the *GUARDIAN Operating System Command Language and Utilities* manual.

`tclprog-file`

is used to derive the names of a pair of disc files into which the SCREEN COBOL object program is placed. *tclprog-file* has the disc file name of the following form:

```
[ \ sysname . ] [ $volume-name . ]
 [ subvol-name . ] disc-file-name
```

Disc-file-name must not exceed 5 characters.

If this parameter is omitted, POBJ is used.

The names of the actual disc files are formed by adding COD (the code file) and DIR (the directory to the code file) to the end of the name supplied. If these files do not exist, SCREEN COBOL creates them and stores the object program in them. If these files already exist, SCREEN COBOL adds the object program to those already present in the file. The addition is done in a way that does not disrupt concurrent users of the file, even if the program ID of the object program being added is the same as one already present. This allows additions to be made to object files while they are in use by a TCP.

When the object files are referenced in PATHCOM commands, the short form (without the COD or DIR) is used. When the object files are referenced by GUARDIAN commands, the actual names (including the COD or DIR) must be used. The actual files can be renamed or copied to another volume as long as the two new file names are related in the same way; the files must be the same except that one ends with COD and the other ends with DIR.

`copy-library`

is the name of an EDIT disc file. This file is used as the default library for source code when expanding a COPY statement that does not include a specific library name. *Copy-library* has the form of a disc file name.

If this parameter is omitted, COPYLIB is used. →

```
compiler-command

  is any compiler command indicated by the following keywords:

    ANSI
    COMPILE
    CROSSREF/NOCROSSREF
    ERRORS
    HEADING
    LINES
    LINES
    LIST/NOLIST
    MAP/NOMAP
    OPTION
    SYMBOLS/NOSYMBOLS
    SETTOG
    SYNTAX
    TANDEM
    WARN/NOWARN
```

The following examples illustrate the command to run the SCREEN COBOL compiler:

```
SCOBOL/IN mysource,OUT $SPOOL, NOWAIT/mprog;MAP

SCOBOL/IN aprog/
```

COMPILER COMMANDS

Compiler commands are used to specify the source format, control listing features, control selective compilation of portions of the source, and request compilation options.

A single compiler command or a single OPTION command followed by a series of available option commands can be entered in the *compiler-command* field of the SCREEN COBOL run command.

Compiler commands can also be included as part of the source text. These commands occupy one line each and can appear at any point in the source text, including those portions retrieved from a source library file with the COPY statement; compiler command lines in the source text cannot be interspersed with multiline COPY statements.

The format of a compiler command in the source text is:

```
?compiler-command

where

  ?

    must be in the indicator field (column 1 for Tandem standard reference format and either
    column 1 or 7 for ANSI standard reference format).

A compiler-command is any of the compiler commands described in this section.
```

The question mark is a source text format indicator and not part of the *compiler-command.* The *compiler-command* entered as part of the SCREEN COBOL run command is not preceded by a question mark.

Compiler commands are listed by category in Table 7-1 and described in alphabetic order in the following paragraphs.

Table 7-1. Compiler Commands

Command Category	Command Name	Purpose
Option Commands	ANSI COMPILE CROSSREF ERRORS HEADING LINES LIST MAP NOCROSSREF NOLIST NOMAP NOSYMBOLS NOWARN OPTION SYMBOLS SYNTAX TANDEM WARN	Specify the source text input format, the source listing options, the title field of the page header, and compilation options.
Toggle Commands	ENDIF IF IFNOT RESETTOG SETTOG	Selectively compile portions of the source text. Up to 15 flags, called toggles, can be turned on (set), turned off (reset), or tested.
Section Command	SECTION	Identifies individual texts in a SCREEN COBOL source library accessed by a COPY statement.

ANSI Command

The ANSI command specifies that the following source text is in ANSI standard reference format. The syntax is:

```
ANSI
```

Lines longer than 80 characters are truncated; shorter lines are padded with trailing blanks. The positions following Margin R (columns 73 through 80) form the identification field. This field, which can contain any ASCII characters, is treated as a comment and has no effect on the meaning of a program.

If this command is omitted, TANDEM is the source test format.

COMPILE Command

The COMPILE command requests full compilation and production of an object file. The syntax is:

```
COMPILE
```

CROSSREF/NOCROSSREF Command

The CROSSREF command causes a list of the SCREEN COBOL program identifiers to be added to the compiled program output. The list is a cross reference showing where the identifiers are described, read, or written throughout the program. This command contains a list of classes into which program identifiers are classified. Selections from the class list determine the identifiers to be included in the cross reference listing.

The NOCROSSREF command is the default and disables the CROSSREF command. The syntax is:

```
CROSSREF  [ ONLY    ]  [ class ] ...
          [ INCLUDE ]
          [ EXCLUDE ]

NOCROSSREF
```

where

ONLY

requests information for just the classes specified.

INCLUDE

adds a class of identifiers to an existing class list. →

EXCLUDE

deletes a class of identifiers from an existing class list.

class

is one of the following SCREEN COBOL identifiers:

CONDITIONS

items tested in the program that have condition names

DATANAMES or VARIABLES

data items defined in the Working-Storage Section

LABELS or PROCNAMES

paragraph names and section names

LITERALS

numeric and nonnumeric literals

MNEMONICS

mnemonic names associated with display attributes

PROGRAMS

program unit names for called programs

SCREEN

screen groups or fields described in the Screen Section

UNREFS

items defined in the program, but never referenced.

Specifiying CROSSREF with no options or classes produces a list of the following program identifiers:

CONDITIONS
DATANAMES or VARIABLES
LABELS or PROCNAMES
MNEMONICS
PROGRAMS
SCREENS

If the NOLIST or SYNTAX commands are specified in the SCREEN COBOL source program or at compile time, no cross reference listing is produced. For a complete description of CROSSREF, refer to the *CROSSREF Users Manual.*

ENDIF Command

The ENDIF command terminates the effect of a preceding IF or IFNOT command. The syntax is:

```
ENDIF toggle-number
```

where

toggle-number

is the *toggle-number* specified in the IF or IFNOT command.

ERRORS Command

The ERRORS command sets the maximum number of severe errors allowed during compilation. The syntax is:

```
ERRORS nnnnn
```

where

nnnnn

is an integer from 0 through 32767.

If this command is omitted, the default limit is 100 errors.

If the limit set by the ERROR command is exceeded, the compilation terminates.

HEADING Command

The HEADING command replaces or sets to blanks the heading portion of the standard top-of-page line that appears on each page of the compilation listing. The syntax is:

```
HEADING [ "character-string" ]
```

where

"character-string"

is a string of any ASCII characters enclosed in quotation mark characters. At least one character must appear. If a quotation mark character is part of the string, it must be represented as two contiguous quotation marks. The character string is used in all subsequent top-of-page lines.

If the *character-string* option is omitted, the heading portion of these lines is set to all blanks.

IF Command

The IF command causes the compiler to ignore subsequent source text unless the specified toggle is turned on with a SETTOG command. The syntax is:

```
IF toggle-number

where

  toggle-number

    is an integer from 1 through 15.
```

COPY statements are not affected by this command; these statements are still processed and expanded.

The following example illustrates the IF command:

```
?RESETTOG 1, 2
     .
     .
     .
?IF 2
     .
     .
     .
    text
     .
     .
     .
?ENDIF 2
```

The source text bounded by the IF 2 and ENDIF 2 commands will be ignored.

IFNOT Command

The IFNOT command causes the compiler to ignore subsequent source text unless the specified toggle is turned off either with the RESETTOG command or by default (never set). The syntax is:

```
IFNOT toggle-number

where

  toggle-number

    is an integer from 1 through 15.
```

COPY statements are not affected by this command; these statements are still processed and expanded.

The following example illustrates the IFNOT command:

```
?RESETTOG 1, 2
     .
     .
     .
?IFNOT 1
     .
     .
     .
    text
     .
     .
     .
?ENDIF 1
```

The source text bounded by the IFNOT 1 and ENDIF 1 commands will be compiled.

LINES Command

The LINES command sets the number of lines listed on each page. Whenever the next line to be listed would overflow the line count given, a page is ejected and the standard page heading and two blank lines are listed at the top of the next page, followed by the pending line. The syntax is:

```
LINES nnnnn
```

where

nnnnn

is an integer from 10 through 32767.

The line limit is ignored if paging does not apply to the compilation list device.

If this command is not issued, the default number of lines per page is 60.

LIST/NOLIST Command

The LIST command transmits each source image to the compilation list device. The NOLIST command disables the LIST option. The syntax is:

```
LIST or NOLIST
```

A MAP command is not effective unless LIST is enabled.

MAP/NOMAP Command

The MAP command lists a table of user-defined symbols following the listing of the program or subprogram source text. The NOMAP command disables the MAP option. The syntax is:

```
MAP or NOMAP
```

The MAP command is not effective unless LIST is enabled.

OPTION Command

The OPTION command controls the source text input format, the source listing options, the title field of the page header, and compilation options. The syntax is:

```
OPTION command-option [ , command-option ] ...

where

  command-option

    is any of the following commands:

      ANSI
      COMPILE
      CROSSREF
      ERRORS
      HEADING
      LINES
      LIST
      MAP
      NOCROSSREF
      NOLIST
      NOMAP
      NOSYMBOLS
      NOWARN
      OPTION
      SYMBOLS
      SYNTAX
      TANDEM
      WARN
```

A single option command can contain any combination of the available options, in any order. An option takes effect at the beginning of the next source text line. If a command contains two or more conflicting options, the last option specified overrides all the others. For example:

```
OPTION LIST, ERRORS 20, LIST, NOLIST
```

is equivalent to

```
OPTION ERRORS 20, NOLIST
```

RESETTOG Command

The RESETTOG command turns off all specified toggles. The syntax is:

```
RESETTOG [ toggle-number [ , toggle-number ] ... ]

where

  toggle-number

    is an integer from 1 through 15.

    If the toggle-number option is omitted, all toggles are turned off.
```

SECTION Command

The SECTION command is used to identify individual texts in a SCREEN COBOL source library accessed by a COPY statement. The command is ignored if it appears in the text of the compilation source file. The syntax is:

```
SECTION text-name [ , library-text-format ]

where

  text-name

    is a SCREEN COBOL word (1 to 30 letters, digits, and hyphens, but not all digits).
```

The compiler assumes that the format of the library text is the same as the current source text format. Although this can be overridden by a compiler command as the first line following the SECTION command, the ANSI or TANDEM command is usually more convenient for this purpose.

SETTOG Command

The SETTOG command turns on all specified toggles. The syntax is:

```
SETTOG [ toggle-number [ , toggle-number ] ... ]

where

  toggle-number

    is an integer from 1 through 15.

    If the toggle-number option is omitted, all toggles are turned on.
```

SYMBOLS/NOSYMBOLS

The SYMBOLS command causes a symbol table file to be built for the SCREEN COBOL program. This file is used by INSPECT to examine and debug programs. The NOSYMBOLS command disables the SYMBOLS command. The syntax is:

```
SYMBOLS or NOSYMBOLS
```

If the SYMBOLS command is omitted, the compiler will not generate a symbol table file.

The SYMBOLS command is not effective unless MAP is enabled.

The file for the symbol table is given the name used in the SCOBOL run command with SYM appended. In the following example, the SCOBOL compiler compiles the program in TESTFILE and adds the symbol table to a file named MANUFSYM.

```
SCOBOL/IN TESTFILE/ manuf;
                    MAP,
                    SYMBOLS
```

SYNTAX Command

The SYNTAX command requests a syntax check of the source text only. No object file is produced. The syntax is:

```
SYNTAX
```

This command checks only syntax and does not produce object files. If this command is specified with the CROSSREF command, no cross-reference listing is produced.

TANDEM Command

The TANDEM command specifies that the following source text is in Tandem standard reference format. The syntax is:

```
TANDEM
```

Lines in Tandem standard reference format are not restricted to a fixed length and can have up to 132 characters (longer lines are truncated). The source text does not include either the initial six-character sequence number area or the final six-character identification field of the ANSI standard reference format.

WARN/NOWARN Command

The WARN command allows minor error conditions to be reported in the source text. The NOWARN command disables the WARN option. The syntax is:

```
WARN or NOWARN
```

If LIST is not enabled, the last line of source text scanned by the compiler is also listed to provide a point of reference.

COMPILATION STATISTICS

Statistics are printed at the end of every compilation. A sample listing is shown in Figure 7-1.

```
OBJECT FILE NAME IS \TS.$DATA.MYSUBVOL.POBJ
PROGRAM NAME IS TESTER1
PROGRAM VERSION IS 4
NO. ERRORS = 0;      NO. WARNINGS = 0
CODE SIZE = 1495
DATA SIZE = 1382
NUMBER OF SOURCE LINES READ = 619
MAXIMUM SYMBOL TABLE SIZE = 3342 WORDS
ELAPSED TIME — 0:01:27
```

OBJECT FILE NAME IS ...

is the short form of the object file name. In the example, the object code for the program is placed on system \TS in the files: $DATA.MYSUBVOL.POBJCOD, POBJDIR, and POBJSYM.

PROGRAM NAME IS ...

is the name of the program. This line is printed only if no errors occurred.

PROGRAM VERSION IS ...

is the version number of the program. This line is printed only if no errors occurred.

NO. ERRORS = ...

is the total number of error messages issued.

NO. WARNING = ...

is the total number of warning messages issued.

CODE SIZE = ...

is the total number of bytes used for all Procedure Division code in the object file name.

DATA SIZE = ...

is the total number of bytes of user-allocated working storage, plus compiler-allocated working storage.

NUMBER OF SOURCE LINES READ = ...

is the total number of source lines read by the SCREEN COBOL compiler, including any COPY lines.

Figure 7-1. Sample Compilation Statistics

MAXIMUM SYMBOL TABLE SIZE = ...

is the number of words that the compiler needed for its symbol table. (This is a snapshot view and should be considered only a rough estimate.)

ELAPSED TIME = ...

is the wall clock elapsed time for the compilation.

Figure 7-1. Sample Compilation Statistics (Continued)

STOPPING THE COMPILER

A compilation is performed by three compiler processes. To stop a compilation before normal completion, do the following:

1. Press the BREAK key.
2. Type STOP to terminate SCOBOL, the first compiler process.
3. Type either STOP and the SCOBOL2 PID number or PAUSE to terminate the second and third compiler processes, SCOBOL2 and SYMSERV respectively. If you type PAUSE, SCOBOL2 issues an error message (**FAILURE 10 ** COMPILER COMMUNICATION LOST : 00) and terminates.
4. Press the BREAK key to return to the Command Interpreter.
5. Type a STATUS command to check that the unwanted processes have terminated.

CONSERVING DISC SPACE

You can prevent unnecessary use of disc space by monitoring the number of object files allowed to accumulate during program development. Each time a SCREEN COBOL source program is compiled, a new version of the object program is added to the code file and an entry is made in the directory file.

For information on how to manipulate and maintain multiple versions of SCREEN COBOL object files, refer to the SCREEN COBOL Utility Program (SCUP) information in the *PATHWAY Programming Aids* manual.

SECTION 8

PATHWAY APPLICATION EXAMPLE

This section provides a sample PATHWAY application. The example includes the commands that create the PATHMON and PATHCOM processes, sample commands that configure PATHWAY and define the components to be used, two SCREEN COBOL programs to illustrate general programming concepts for terminals operating in block mode and for terminals operating in conversational mode, and an associated server program written in COBOL.

This example can be duplicated exactly as shown by taking the following steps in the order indicated:

1. Code the sample COBOL server program.

2. Compile and run the sample COBOL server program. Rename the default object file RUNUNIT to EXSERV to correspond to the PATHWAY configuration.

3. Code the sample SCREEN COBOL application program.

4. Compile the sample SCREEN COBOL application program for terminals operating in block mode. The object files default to POBJCOD and POBJDIR.

   Compile and run the sample SCREEN COBOL application program for terminals operating in conversational mode. This program is just for illustration and does not communicate with the server program included in this section.

5. Code the PATHWAY configuration file, named PWCONFIG. Change the process names $term01 and $term02 in the two SET TERM commands to legal terminal names in your installation. For convenience, the second set of SET TERM commands specifying $term02 can be eliminated without interfering with the operation.

6. Set up an obey file that contains the PATHMON and PATHCOM run commands. The PATHMON name $PM can be changed to any appropriate five-letter name.

7. Issue an OBEY obey command.

8. Issue the PATHCOM RUN command for the SCREEN COBOL application program.

The following rules should be noted before attempting to establish this application:

- The PATHWAY configuration is established through a command terminal. The command terminal cannot be included in the configuration.
- A PATHWAY system should allow more than one PATHCOM to communicate with PATHMON at a time; therefore, the SET PATHWAY MAXPATHCOMS command should never be set to 1. (The default is 5.)
- The individual at the command terminal can issue appropriate commands and alter the PATHWAY configuration on-line. This is accomplished by typing PATHCOM and responding to the PATHCOM = prompt. If the PATHMON name is not $PM, this PATHCOM command must include the $process-name parameter.
- If a configured terminal has a command interpreter, the terminal operator must type PAUSE to activate the SCREEN COBOL application on the terminal.

The sample SCREEN COBOL requester program describes a base screen only that appears as pictured in Figure 8-1. The program accepts operator input that consists of a name and address until an appropriate function key is pressed.

If the operator keys the name SMITH and presses the F2 key to enter data, the server returns a reply-code of 999 and an error-code of 1; this causes the message SMITH IS ALREADY ON FILE to be displayed in the advisory field of the terminal.

If the operator keys the name JONES and presses the F2 key to enter data, the server returns a reply-code of 999 and an error-code of 2; this causes the message JONES IS ALREADY ON FILE to be displayed in the advisory field of the terminal.

If the operator keys any other name and presses the F2 key to enter data, the server returns a reply-code of 0 and writes the record; this causes the entered record to be displayed on the terminal screen.

```
                    EXAMPLE SCREEN COBOL PROGRAM

DEPARTMENT : MKT            PASSWORD :

NAME : --------------------------------
ADDR : --------------------

MONTH : FEBRUARY      DAY : 15    YEAR : 82

REPLY -



F1 - ENTER PASSWORD                  F5  - BLINK REPLY
F2 - ENTER DATA                      F6  - RESET ATTR REPLY
F3 - CLEAR INPUT                     F7  - RESET DATA REPLY
F4 - RESET DATA SCREEN               F16 - EXIT PROGRAM
```

Figure 8-1. PATHWAY Application Example Screen

The sample SCREEN COBOL requester program for terminals operating in conversational mode describes a base screen only and illustrates the SCREEN COBOL characteristics for conversational mode programs. The program displays a screen header and prompts the operator, accepts operator input that consists of a name and an address, contains a function selection, and responds to the input control characters named in the program, but no server response is provided.

To execute the program use the PATHCOM RUN command.

PATHMON AND PATHCOM PROCESS CREATION

```
PATHMON/NAME $PM,CPU 0,NOWAIT/
PATHCOM/IN PWCONFIG/$PM
```

where PWCONFIG contains the following commands;

```
SET PATHMON    BACKUPCPU 1
SET PATHWAY    MAXTCPS 1
SET PATHWAY    MAXTERMS 5
SET PATHWAY    MAXSERVERCLASSES 5
SET PATHWAY    MAXSERVERPROCESSES 5
SET PATHWAY    MAXSTARTUPS 5
SET PATHWAY    MAXASSIGNS 5
SET PATHWAY    MAXPARAMS 5
SET PATHWAY    MAXPATHCOMS 3
SET PATHWAY    MAXPROGRAMS 1

START PATHWAY COLD

RESET TCP
SET TCP        CPUS 1:2
SET TCP        PROGRAM $SYSTEM.SYSTEM.PATHTCP
SET TCP        PRI 141
SET TCP        PROCESS $XTCP
SET TCP        TCLPROG pobj
SET TCP        MAXTERMS 5
ADD TCP        ex-tcp

RESET TERM
SET TERM       FILE $term01
SET TERM       TCP ex-tcp
SET TERM       INITIAL example
SET TERM       TMF OFF
ADD TERM       t1

RESET TERM
SET TERM       LIKE t1
SET TERM       FILE $term02
ADD TERM       t2

RESET PROGRAM
SET PROGRAM    TCP ex-tcp
SET PROGRAM    TYPE T16-6520 INITIAL example
SET PROGRAM    ERROR-ABORT ON
SET PROGRAM    TMF OFF
ADD PROGRAM    exprog

RESET SERVER
SET SERVER     PROGRAM exserv
SET SERVER     CPUS 0:1
SET SERVER     NUMSTATIC 1
SET SERVER     MAXSERVERS 3
ADD SERVER     example-server

START TCP      ex-tcp
START TERM *
```

The SCREEN COBOL programs and an associated COBOL server program appear on the following pages.

SCREEN COBOL PROGRAM FOR BLOCK MODE

```
IDENTIFICATION DIVISION.                                                    1
  PROGRAM-ID. EXAMPLE.                                                      2
                                                                            3
ENVIRONMENT DIVISION.                                                       4
  CONFIGURATION SECTION.                                                    5
  SOURCE-COMPUTER. T16.                                                     6
  OBJECT-COMPUTER. T16,                                                     7
   TERMINAL IS T16-6520.                                                    8
  SPECIAL-NAMES.                                                            9
   F1-KEY IS F1, F2-KEY IS F2, F3-KEY IS F3, F4-KEY IS F4                   10
   F5-KEY IS F5, F6-KEY IS F6, F7-KEY IS F7, F16-KEY IS F16                 11
   ATTENTION IS BLINK, HIDDEN IS HIDDEN.                                    12
                                                                            13
DATA DIVISION.                                                              14
   WORKING-STORAGE SECTION.                                                 15
   01 WS.                                                                   16
      02 ERROR-MSG           PIC X(77).                                     17
      02 PASSWORD            PIC X(3).                                      18
      02 DEPT-HEADER         PIC X(3).                                      19
   01 EXIT-FLAG              PIC S9          VALUE 0.                       20
      88 EXIT-PROGRAM         VALUE 1.                                      21
   01 ENTRY-MSG.                                                            22
      02 PW-HEADER.                                                         23
        04 REPLY-CODE        PIC S9(4) COMP.                                24
        04 APPLICATION-CODE  PIC XX.                                        25
        04 FUNCTION-CODE     PIC XX.                                        26
        04 TRANS-CODE        PIC 99.                                        27
        04 TERM-ID           PIC X(15).                                     28
        04 LOG-REQUEST       PIC X.                                         29
      02 ENTRY-GROUP.                                                       30
        04 NAME-IN           PIC A(30).                                     31
        04 ADDR-IN           PIC X(20).                                     32
        04 DATE-GRP.                                                        33
          06 MONTH-IN        PIC A(10).                                     34
          06 DAY-IN          PIC 99.                                        35
          06 YEAR-IN         PIC 99.                                        36
                                                                           37
   01 ENTRY-REPLY.                                                         38
      02 PW-HEADER.                                                        39
        04 REPLY-CODE        PIC S9(4) COMP.                               40
        04 FILLER            PIC X(22).                                    41
      02 SERVER-RECORD       PIC X(64).                                    42
                                                                          43
   01 ERROR-REPLY.                                                        44
      02 REPLY-CODE          PIC S9(4) COMP.                               45
      02 FILLER              PIC X(22).                                    46
      02 ERROR-CODE          PIC S999  COMP.                               47
```

Lines 1-2
These lines give the program name that is specified in the SET TERM INITIAL command. This program is used when a terminal is first started.

Lines 23-29
These lines illustrate a sample header for the transaction messages. Lines 25-29 are not required.

Lines 38-47
Two reply messages are used to limit the amount of data sent between the server and the SCREEN COBOL program. When only an error code is returned from the server, ERROR-REPLY is used. When data is returned, ENTRY-REPLY is used.

Line 40 and Line 45
These lines show the reply code that is required by PATHWAY.

```
SCREEN SECTION.                                                          1
01 EXAMPLE-SCREEN BASE SIZE 24, 80.                                      2
  03 FILLER       AT 1, 20 VALUE "EXAMPLE SCREEN COBOL PROGRAM".         3
  03 FILLER       AT 3, 1  VALUE "DEPARTMENT :".                         4
  03 DEPT-HEADER AT 3, 14 PIC X(3) FROM DEPT-HEADER OF WS.               5
  03 FILLER       AT 3, * + 10 VALUE "PASSWORD :".                       6
  03 PASSWORD     AT 3, * + 2  PIC X(3) LENGTH 1 THRU 3, HIDDEN,         7
           UPSHIFT INPUT, MUST BE "AAA", "X", TO PASSWORD OF WS.         8
  03 DATA-IN.                                                            9
     05 FILLER       AT 5, 1  VALUE "NAME :".                           10
     05 NAME-IN      AT 5, 8  PIC A(30) LENGTH 1 THRU 30                11
                    TO NAME-IN OF ENTRY-MSG, FILL "_".                  12
     05 FILLER       AT 6, 1  VALUE "ADDR :".                           13
     05 ADDR-IN      AT 6, 8  PIC X(20) LENGTH 1 THRU 20                14
                    TO ADDR-IN OF ENTRY-MSG, FILL "_".                  15
     05 DATE-GRP     AT 8, 1.                                           16
        07 FILLER    AT @, 1  VALUE "MONTH :".                          17
        07 MONTH-IN AT @, * + 2  PIC A(10) LENGTH 1 THRU 10             18
              MUST BE "JANUARY", "FEBRUARY" USING MONTH-IN OF           19
              ENTRY-MSG, UPSHIFT INPUT, VALUE "FEBRUARY".               20
        07 FILLER    AT @, * + 4 VALUE "DAY :".                         21
        07 DAY-IN    AT @, * + 2 PIC Z9 LENGTH 1 THRU 2, VALUE "15"     22
              MUST BE 1 THRU 31, USING DAY-IN OF ENTRY-MSG.             23
        07 FILLER    AT @, * + 4  VALUE "YEAR :".                       24
        07 YEAR-IN   AT @, * + 2 PIC Z9 MUST BE 79, 82, 85 THRU 88      25
              USING YEAR-IN OF ENTRY-MSG, VALUE "82".                   26
                                                                        27
  03 FILLER          AT 10, 1 VALUE "REPLY -".                          28
  03 SERVER-RECORD   AT 10, * + 2  PIC X(64)                            29
                     FROM SERVER-RECORD OF ENTRY-REPLY.                 30
  03 FILLER          AT 18, 1 VALUE                                     31
     "F1 - ENTER PASSWORD                F5  - BLINK REPLY".          32
  03 FILLER          AT 19, 1 VALUE                                     33
     "F2 - ENTER DATA                    F6  - RESET ATTR REPLY".     34
  03 FILLER          AT 20, 1 VALUE                                     35
     "F3 - CLEAR INPUT                   F7  - RESET DATA REPLY".     36
  03 FILLER          AT 21, 1 VALUE                                     37
     "F4 - RESET DATA SCREEN             F16 - EXIT PROGRAM".         38
  03 ERROR-MSG       AT 24, 2 PIC X(76)  ADVISORY                       39
                     FROM ERROR-MSG OF WS.                              40
```

Line 3
This literal is displayed on the screen starting at line 1, column 20.

Line 4
This literal is displayed on the screen starting at line 3, column 1.

Line 5
If a data name is used in a screen section, a PIC clause must be associated with that data name. The FROM (data association clause) specifies an output association. Data is moved from DEPT-HEADER OF WS to this position on the screen.

Line 6
The asterisk means relative to the current position; therefore, the literal PASSWORD is displayed on the screen at line 3, column 26 (16 + 10).

Lines 7-8
The data entered for PASSWORD is hidden from the operator as it is entered. The password is upshifted and tested for the correct value. If the password is correct, the password is moved to the data name PASSWORD of working storage.

Lines 11-12
The operator must key in from 1 to 30 alphabetic characters that are moved to ENTRY-MSG. A fill character of underscore is present on the screen in these 30 positions.

Line 17
The at sign (@) indicates the position is relative to the home position of the group. This literal is displayed on line 8, column 1.

Lines 39-40
These lines identify the field to be used for information and error messages generated by the TCP. The programmer also can use this field.

```
PROCEDURE DIVISION.                                                    1
A-MAIN.                                                                2
    DISPLAY BASE EXAMPLE-SCREEN.                                       3
    MOVE "MKT" TO DEPT-HEADER OF WS.                                   4
    DISPLAY DEPT-HEADER OF EXAMPLE-SCREEN.                             5
    ACCEPT PASSWORD OF EXAMPLE-SCREEN UNTIL F1-KEY.                    6
    PERFORM CASE-MANAGER UNTIL EXIT-PROGRAM.                           7
A-EXIT.                                                                8
    EXIT PROGRAM.                                                      9
                                                                       10
CASE-MANAGER.                                                          11
    ACCEPT DATA-IN OF EXAMPLE-SCREEN UNTIL F2-KEY                      12
      ESCAPE ON F3-KEY F4-KEY F5-KEY F6-KEY F7-KEY F16-KEY.            13
    PERFORM ONE OF                                                     14
      DATA-ENTERED, CLEAR-INPUT, RESET-DATA, BLINK-REPLY               15
      RESET-ATTR-REPLY, RESET-DATA-REPLY, SET-EXIT                     16
    DEPENDING ON TERMINATION-STATUS.                                   17
                                                                       18
DATA-ENTERED.                                                          19
    MOVE SPACES TO PW-HEADER OF ENTRY-MSG.                             20
    PERFORM SEND-DATA.                                                 21
CLEAR-INPUT.                                                           22
    CLEAR INPUT.                                                       23
RESET-DATA.                                                            24
    RESET DATA EXAMPLE-SCREEN.                                         25
BLINK-REPLY.                                                           26
    TURN ATTENTION IN SERVER-RECORD OF EXAMPLE-SCREEN.                 27
RESET-ATTR-REPLY.                                                      28
    RESET ATTR SERVER-RECORD OF EXAMPLE-SCREEN.                        29
RESET-DATA-REPLY.                                                      30
    RESET DATA SERVER-RECORD OF EXAMPLE-SCREEN.                        31
SET-EXIT.                                                              32
    MOVE 1 TO EXIT-FLAG.                                               33
                                                                       34
SEND-DATA.                                                             35
    SEND ENTRY-MSG TO "EXAMPLE-SERVER"                                 36
      REPLY CODE 0    YIELDS ENTRY-REPLY                               37
            CODE 999  YIELDS ERROR-REPLY.                              38
    IF TERMINATION-STATUS = 2 AND ERROR-CODE = 1                       39
       MOVE "SMITH IS ALREADY ON FILE" TO ERROR-MSG OF WS              40
       PERFORM 901-DISPLAY-ADVISORY                                    41
    ELSE IF TERMINATION-STATUS = 2 AND ERROR-CODE = 2                  42
       MOVE "JONES IS ALREADY ON FILE" TO ERROR-MSG OF WS              43
       PERFORM 901-DISPLAY-ADVISORY                                    44
    ELSE                                                               45
       DISPLAY SERVER-RECORD OF EXAMPLE-SCREEN.                        46
                                                                       47
901-DISPLAY-ADVISORY.                                                  48
    DISPLAY TEMP ERROR-MSG OF EXAMPLE-SCREEN.                          49
    TURN TEMP ATTENTION IN ERROR-MSG OF EXAMPLE-SCREEN.                50
```

Line 3
This line displays the screen and the initial values, FILL characters, and default values.

Line 5
The value of DEPT-HEADER is moved to the screen at line 3, column 14.

Line 6
When the F1 key is pressed, the field is tested for validity. Data can be keyed into any other field on the screen, but only the PASSWORD field is used.

Lines 12-13
The UNTIL F2-KEY expects data to be entered before the F2 key is pressed and validity checks are performed. The ESCAPE series of function keys causes the statement to terminate without data being entered.

Line 17
The key that was pressed to terminate the ACCEPT statement has a positional value associated with it from the ACCEPT statement; the key is put into TERMINATION-STATUS.

Line 23
All unprotected fields are cleared.

Line 25
This line resets the fields to the initial values and FILL characters declared.

Line 27
This line causes SERVER-RECORD to blink by setting the BLINK attribute.

Line 29
This line stops the blinking of SERVER-RECORD by resetting the attribute to normal.

Line 31
This line resets the data portion of SERVER-RECORD to its original value (blank line).

Line 36
This line specifies the server class to be used. This can be a data name in working storage.

Line 46
The fields that comprise SERVER-RECORD are displayed.

Line 49
This line displays ERROR-MSG on the screen as temporary data.

Line 50
This line sets the blink attribute as a temporary attribute and makes the value of ERROR-MSG blink.

SCREEN COBOL PROGRAM FOR CONVERSATIONAL MODE

```
IDENTIFICATION DIVISION.                                                    1
 PROGRAM-ID. DCONV-EXP.                                                     2
                                                                            3
ENVIRONMENT DIVISION.                                                       4
 CONFIGURATION SECTION.                                                     5
 SOURCE-COMPUTER. T16.                                                      6
 OBJECT-COMPUTER. T16, TERMINAL IS CONVERSATIONAL.                          7
 SPECIAL-NAMES.                                                             8
               BELL IS BELL,                                                9
               NOBELL IS NOBELL.                                            10
                                                                            11
DATA DIVISION.                                                              12
WORKING-STORAGE SECTION.                                                    13
01 EMPLOYEE-REC.                                                            14
  05  EMP-LAST-NAME                  PIC X(10) VALUE SPACES.               15
  05  EMP-FIRST-NAME                 PIC X(10) VALUE SPACES.               16
  05  EMP-MIDDLE-INIT                PIC X(02) VALUE SPACES.               17
  05  EMP-ADDR                       PIC X(30) VALUE SPACES.               18
  05  EMP-CITY                       PIC X(10) VALUE SPACES.               19
  05  EMP-STATE                      PIC X(02) VALUE SPACES.               20
  05  EMP-ZIP                        PIC 9(05) VALUE ZEROS .               21
                                                                            22
01 WS-ADVISORY                       PIC X(70) VALUE SPACES.               23
                                                                            24
01 WS-FUNC                           PIC X(06) VALUE SPACES.               25
  88  WS-SEARCH-REQUEST                        VALUE "SEARCH".              26
  88  WS-ADD-REQUEST                           VALUE "ADD".                 27
  88  WS-DELETE-REQUEST                        VALUE "DELETE".              28
  88  WS-SHOW-REQUEST                          VALUE "SHOW".                29
  88  WS-EXIT-REQUEST                          VALUE "EXIT".                30
                                                                            31
01 EXIT-FLAG                         PIC 9(01) COMP VALUE ZERO.            32
  88 EXIT-PROGRAM                             VALUE 1.                      33
  88 INVALID-RESPONSE                         VALUE 2.                      34
                                                                            35
01 MESSAGE-ID                        PIC 9(04) COMP VALUE ZERO.            36
                                                                            37
01 R-CODE                            PIC 9(04) COMP VALUE ZERO.            38
  88 SEND-ERROR                               VALUE 999.                    39
```

Line 2

This line gives the program name that is specified in the SET TERM INITIAL command.

Line 7

This line specifies a conversational mode terminal type and identifies the terminal type that is specified in the SET PROGRAM TYPE and SET TERM TYPE commands.

```
 SCREEN SECTION.                                                          1
 01 EMPLOYEE-REC-SCREEN    BASE  SIZE 24, 80                             2
*                                    FIELD-SEPARATOR ","                 3
*                                    GROUP-SEPARATOR ";"                 4
                                     ABORT-INPUT     "AI"                5
                                     END-OF-INPUT    47                  6
                                                                         7
*   The keyboard character for END-OF-INPUT is "/"                       8
                                                                         9
                                     RESTART-INPUT   58, 58.             10
                                                                         11
*   The keyboard characters for RESTART-INPUT are "::"                   12
                                                                         13
    05 TITLE                 AT 1, 3    VALUE "PERSONNEL SYSTEM EXAMPLE". 14
    05 NAME-PROMPT           AT 2, 1    VALUE "LAST NAME: ".              15
    05 LAST-NAME-FLD         AT 3, 1    PIC X(10)                         16
                                        USING EMP-LAST-NAME               17
                                        LENGTH 1 THRU 10                  18
                                        PROMPT NAME-PROMPT.               19
                                                                          20
    05 FIRST-NAME-PROMPT     AT 2, 12   VALUE "FIRST NAME: ".            21
    05 FIRST-NAME-FLD        AT 3, 12   PIC X(10)                        22
                                        USING EMP-FIRST-NAME              23
                                        LENGTH 1 THRU 10                  24
                                        PROMPT FIRST-NAME-PROMPT.         25
                                                                          26
    05 MI-PROMPT             AT 2, 24   VALUE "MI: ".                    27
    05 MIDDLE-INIT-FLD       AT 3, 24   PIC X(2)                         28
                                        USING EMP-MIDDLE-INIT             29
                                        PROMPT MI-PROMPT.                 30
                                                                          31
    05 ADDR-PROMPT           AT 4, 1    VALUE "ADDRESS: ".               32
    05 ADDR-FLD              AT 4, 11   PIC X(30)                        33
                                        USING EMP-ADDR                    34
                                        PROMPT ADDR-PROMPT.               35
                                                                          36
    05 CITY-PROMPT           AT 5, 1    VALUE "CITY: ".                  37
    05 CITY-FLD              AT 5, 11   PIC X(10)                        38
                                        USING EMP-CITY                    39
                                        PROMPT CITY-PROMPT.               40
                                                                          41
    05 STATE-PROMPT          AT 5, 22   VALUE "STATE: ".                 42
    05 STATE-FLD             AT 5, 30   PIC X(10)                        43
                                        USING EMP-STATE                   44
                                        PROMPT STATE-PROMPT.              45
                                                                          46
    05 ZIP-PROMPT            AT 5, 45   VALUE "ZIP: ".                   47
    05 ZIP-FLD               AT 5, 51   PIC Z(5)                         48
                                        USING EMP-ZIP                     49
                                        PROMPT ZIP-PROMPT.                50
                                                                          51
    05 TYPEAHEAD-MSG         AT 10,     VALUE "TO GET TYPEAHEAD, ENTER    52
-                              "LAST NAME, FIRST NAME, MIDDLE INITIAL.".  53
```

Lines 3, 4, 8, and 12

These lines are instructive comments about the input control characters. They are not required by SCREEN COBOL.

Line 19

This is the first PROMPT clause for the screen. The value of this clause (line 15) will be displayed indicating the terminal is ready to accept data for this field.

Line 52 and 53

These lines identify the typeahead message that is included in the heading displayed at the beginning of the screen.

```
   05 PROMPT-AREA  AREA     AT 23, 1  SIZE 1, 80.                        1
                                                                         2
   05 ADVISORY-FLD          AT 24, 1  PIC X(70)                         3
                                      ADVISORY FROM WS-ADVISORY.         4
                                                                         5
 01 EMPLOYEE-REC-PROMPT     OVERLAY   SIZE 1, 80.                        6
                                                                         7
   05 FUNC-PROMPT           AT  1, 1 VALUE "(FUNCTION) SEARCH, ADD,      8
-                             "DELETE, SHOW, EXIT: ".                    9
                                                                        10
   05 FUNC-INPUT            AT  1, 45 PIC X(06)                        11
                                      TO WS-FUNC                        12
                                      UPSHIFT INPUT                     13
                                      LENGTH MUST BE 3 THRU 6           14
                                      PROMPT FUNC-PROMPT.               15
```

```
PROCEDURE DIVISION.                                                    1
                                                                       2
BEGIN-PROGRAM.                                                         3
    DISPLAY BASE EMPLOYEE-REC-SCREEN.                                  4
    DISPLAY TITLE, TYPEAHEAD-MSG.                                      5
    PERFORM LOOP UNTIL EXIT-PROGRAM.                                   6
                                                                       7
EXIT-PROG.                                                             8
    EXIT PROGRAM.                                                      9
                                                                       10
LOOP.                                                                  11
   ACCEPT EMPLOYEE-REC-SCREEN                                          12
      UNTIL INPUT                                                      13
      ESCAPE ON ABORT.                                                 14
                                                                       15
   IF TERMINATION-STATUS = 1                                           16
      PERFORM FUNCTION-DISPLAY                                         17
      PERFORM INIT-EMPLOYEE-REC                                        18
   ELSE                                                                19
      PERFORM EXIT-IT.                                                 20
                                                                       21
FUNCTION-DISPLAY.                                                      22
   DISPLAY OVERLAY EMPLOYEE-REC-PROMPT AT PROMPT-AREA.                 23
   MOVE 2 TO EXIT-FLAG.                                                24
   PERFORM OPERATION UNTIL NOT INVALID-RESPONSE.                       25
                                                                       26
OPERATION.                                                             27
   ACCEPT EMPLOYEE-REC-PROMPT                                          28
      UNTIL INPUT                                                      29
      ESCAPE ON ABORT.                                                 30
   IF TERMINATION-STATUS = 1                                           31
      PERFORM FUNCTION-SELECTION                                       32
   ELSE                                                                33
      PERFORM EXIT-IT.                                                 34
                                                                       35
FUNCTION-SELECTION.                                                    36
   MOVE ZERO TO EXIT-FLAG.                                             37
   IF WS-SEARCH-REQUEST                                                38
      PERFORM SEARCH-IT                                                39
   ELSE                                                                40
   IF WS-ADD-REQUEST                                                   41
      PERFORM ADD-IT                                                   42
   ELSE                                                                43
   IF WS-DELETE-REQUEST                                                44
      PERFORM DELETE-IT                                                45
   ELSE                                                                46
   IF WS-SHOW-REQUEST                                                  47
      PERFORM SHOW-IT                                                  48
   ELSE                                                                49
   IF WS-EXIT-REQUEST                                                  50
      PERFORM EXIT-IT                                                  51
   ELSE                                                                52
      PERFORM INVALID-FUNCTION.                                        53
```

```
SEARCH-IT.                                                              1
   MOVE 1 TO MESSAGE-ID.                                                2
   SEND MESSAGE-ID, EMPLOYEE-REC TO "USER-SERVER"                       3
     REPLY CODE 1 YIELDS R-CODE, EMPLOYEE-REC                           4
           CODE 2 YIELDS R-CODE                                         5
     ON ERROR MOVE 999 TO R-CODE.                                       6
   IF NOT SEND-ERROR                                                    7
     PERFORM ONE OF DISPLAY-EMPLOYEE-REC, EMPLOYEE-NOT-FOUND            8
     DEPENDING ON R-CODE                                                9
   ELSE                                                                 10
     PERFORM SEND-ERROR-NOTICE.                                         11
                                                                        12
ADD-IT.                                                                 13
   MOVE 2 TO MESSAGE-ID.                                                14
   SEND MESSAGE-ID, EMPLOYEE-REC TO "USER-SERVER"                       15
     REPLY CODE 1, 3 YIELDS R-CODE                                      16
     ON ERROR MOVE 999 TO R-CODE.                                       17
   IF NOT SEND-ERROR                                                    18
     PERFORM ONE OF EMPLOYEE-ADDED, EMPLOYEE-ALREADY-EXISTS             19
     DEPENDING ON R-CODE                                                20
   ELSE                                                                 21
      PERFORM SEND-ERROR-NOTICE.                                        22
                                                                        23
DELETE-IT.                                                              24
   MOVE 3 TO MESSAGE-ID.                                                25
   SEND MESSAGE-ID, EMPLOYEE-REC TO "USER-SERVER"                       26
     REPLY CODE 1, 2 YIELDS R-CODE                                      27
     ON ERROR MOVE 999 TO R-CODE.                                       28
   IF NOT SEND-ERROR                                                    29
    PERFORM ONE OF EMPLOYEE-DELETED, EMPLOYEE-NOT-FOUND                 30
      DEPENDING ON R-CODE                                               31
   ELSE                                                                 32
      PERFORM SEND-ERROR-NOTICE.                                        33
                                                                        34
SHOW-IT.                                                                35
   MOVE 4 TO MESSAGE-ID.                                                36
   SEND MESSAGE-ID, EMPLOYEE-REC TO "USER-SERVER"                       37
     REPLY CODE 1, 2 YIELDS R-CODE                                      38
     ON ERROR MOVE 999 TO R-CODE.                                       39
   IF NOT SEND-ERROR                                                    40
    PERFORM ONE OF DISPLAY-EMPLOYEE-REC, EMPLOYEE-NOT-FOUND             41
      DEPENDING ON R-CODE                                               42
   ELSE                                                                 43
      PERFORM SEND-ERROR-NOTICE.                                        44
                                                                        45
EXIT-IT.                                                                46
  MOVE 1 TO EXIT-FLAG.                                                  47
                                                                        48
DISPLAY-EMPLOYEE-REC.                                                   49
  DISPLAY EMPLOYEE-REC-SCREEN.                                          50
                                                                        51
EMPLOYEE-NOT-FOUND.                                                     52
  MOVE "EMPLOYEE DOES NOT EXIST" TO WS-ADVISORY.                        52
  DISPLAY ADVISORY-FLD.                                                 53
                                                                        54
EMPLOYEE-ADDED.                                                         55
  MOVE "EMPLOYEE ADDED" TO WS-ADVISORY.                                 56
  DISPLAY ADVISORY-FLD.                                                 57
```

```
EMPLOYEE-ALREADY-EXISTS.                                                1
  MOVE "EMPLOYEE ALREADY EXISTS" TO WS-ADVISORY.                        2
                                                                        3
  DISPLAY ADVISORY-FLD.                                                 4
                                                                        5
EMPLOYEE-DELETED.                                                       6
  MOVE "EMPLOYEE DELETED" TO WS-ADVISORY.                               7
  DISPLAY ADVISORY-FLD.                                                 8
                                                                        9
INIT-EMPLOYEE-REC.                                                      10
  MOVE SPACES TO EMPLOYEE-REC.                                          11
  MOVE ZEROES TO EMP-ZIP.                                               12
                                                                        13
INVALID-FUNCTION.                                                       14
  MOVE 2 TO EXIT-FLAG.                                                  15
  MOVE "INVALID FUNCTION REQUESTED" TO WS-ADVISORY.                     16
  DISPLAY ADVISORY-FLD.                                                 17
                                                                        18
SEND-ERROR-NOTICE.                                                      19
  MOVE "ERROR ACCESSING PERSONNEL SYSTEM" TO WS-ADVISORY.               20
  DISPLAY ADVISORY-FLD.                                                 21
```

SERVER PROGRAM IN COBOL

```
IDENTIFICATION DIVISION.
 PROGRAM-ID. EXAMPLE-SERVER.

ENVIRONMENT DIVISION.
 CONFIGURATION SECTION.
 SOURCE-COMPUTER. TANDEM/16.
 OBJECT-COMPUTER. TANDEM/16.
 INPUT-OUTPUT SECTION.
   FILE-CONTROL.
     SELECT MESSAGE-IN, ASSIGN TO $RECEIVE
       FILE STATUS IS RECEIVE-FILE-STATUS.
     SELECT MESSAGE-OUT, ASSIGN TO $RECEIVE
       FILE STATUS IS RECEIVE-FILE-STATUS.

DATA DIVISION.
FILE SECTION.
FD  MESSAGE-IN
    LABEL RECORDS ARE OMITTED.
  01 ENTRY-MSG.
     02 PW-HEADER.
       04 REPLY-CODE        PIC S9(4) COMP.
       04 APPLICATION-CODE  PIC XX.
       04 FUNCTION-CODE     PIC XX.
       04 TRANS-CODE        PIC 99.
       04 TERM-ID           PIC X(15).
       04 LOG-REQUEST       PIC X.
     02 ENTRY-GROUP.
       04 NAME-IN           PIC A(30).
       04 ADDR-IN           PIC X(20).
       04 DATE-GRP.
         06 MONTH-IN        PIC A(10).
         06 DAY-IN          PIC 99.
         06 YEAR-IN         PIC 99.

FD  MESSAGE-OUT
    LABEL RECORDS ARE OMITTED
    RECORD CONTAINS 1 TO 88 CHARACTERS.
  01 ENTRY-REPLY.
     02 PW-HEADER.
       04 REPLY-CODE        PIC S9(4) COMP.
       04 FILLER            PIC X(22).
     02 SERVER-RECORD       PIC X(64).

  01 ERROR-REPLY.
     02 REPLY-CODE          PIC S9(4) COMP.
     02 FILLER              PIC X(22).
     02 ERROR-CODE          PIC S999  COMP.

WORKING-STORAGE SECTION.
  01 RECEIVE-FILE-STATUS.
     02 STAT-1              PIC 9.
        88 CLOSE-FROM-REQUESTOR VALUE 1.
     02 STAT-2              PIC 9.
```

```
PROCEDURE DIVISION.
BEGIN-COBOL-SERVER.
    OPEN INPUT MESSAGE-IN.
    OPEN OUTPUT MESSAGE-OUT SYNCDEPTH 1.
    PERFORM B-TRANS UNTIL CLOSE-FROM-REQUESTOR.
    STOP RUN.

B-TRANS.
    MOVE SPACES TO ENTRY-REPLY, ENTRY-MSG.
    READ MESSAGE-IN, AT END STOP RUN.
    MOVE PW-HEADER OF MESSAGE-IN TO PW-HEADER OF MESSAGE-OUT.
    IF NAME-IN = "SMITH"
      MOVE 999 TO REPLY-CODE OF ERROR-REPLY
      MOVE 1 TO ERROR-CODE
      WRITE ERROR-REPLY
    ELSE IF NAME-IN = "JONES"
      MOVE 999 TO REPLY-CODE OF ERROR-REPLY
      MOVE 2 TO ERROR-CODE
      WRITE ERROR-REPLY
    ELSE
      MOVE 0 TO REPLY-CODE OF ENTRY-REPLY
      MOVE ENTRY-GROUP TO SERVER-RECORD
      WRITE ENTRY-REPLY.
```

APPENDIX A

MESSAGES

Three types of messages are provided by SCREEN COBOL. These messages are:

- Advisory messages displayed for the terminal operator during input checking.
- Diagnostic screen messages displayed for the terminal operator to report device error or termination conditions.
- Compilation diagnostic messages reported during source program compilation.

Each type of message is described in this appendix.

ADVISORY MESSAGES

The PATHWAY Terminal Control Process (TCP) displays messages in the advisory field. The advisory field is an alphanumeric output field defined in the ADVISORY field characteristic clause of the Screen Section. Messages in this field primarily describe errors detected during input checking. The text of each standard message is a brief indication of the condition that invoked the message.

The text of the message and the number used internally in the TCP to refer to the message are listed in Table A-1. The list of messages provides a reference for those installations that develop their own user conversion procedures.

Table A-1. Advisory Messages

Number	Message Text	Meaning
1	REQUIRED FIELD MISSING	The field does not allow zero length input.
2	PREVIOUS FIELD MISSING	For a required occurring field with a DEPENDING clause, an occurrence is present but a previous occurrence was absent.
3	EARLIER FIELD MISSING	For an occurring field with a DEPENDING clause, a different occurring field DEPENDING on the same item was required but absent for this occurrence number, and this field's occurrence is present.
4	FIELD TOO SHORT	The length of the field data, after stripping of fill characters and spaces, is shorter than allowed.
5	FIELD NOT CORRECT LENGTH	The input does not have an allowed length.
6	FIELD TOO LONG	The input is too long. Generally this occurs only when the terminal's formatting has been corrupted.
7	WRONG FORMAT	Input to an alphanumeric item does not obey the PICTURE.
8	WRONG FORMAT: DIGIT EXPECTED	Input to an alphanumeric item does not have a digit where a 9 symbol appeared in the PICTURE.
9	WRONG FORMAT: LETTER EXPECTED	Input to an alphanumeric item does not have a letter or space where an A appeared in the PICTURE.
10	INVALID NUMBER FORMAT	Input to a numeric item does not obey the PICTURE.
11	VALUE WRONG	The numeric value input is larger than allowed by the constraints imposed by the field and the receiving data item.
12	VALUE INCORRECT	The input value is not allowed by the MUST BE constraints.
13	MESSAGE:	This text is used to prefix a tell message.

Table A-1. Advisory Messages (Continued)

Number	Message Text	Meaning
14	DEP OCCUR FLD ERR-INPUT RESTARTED (a)	For multiple screen-identifiers in which the OCCURS DEPENDING ON clauses reference the same depend data item, one of the following is detected: (1) The depend data item value is greater than the maximum number of elements allowed for one of the screen-identifiers. (2) A required screen-identifier field occurs fewer times than the value of the associated depend data item.
15	ABORT NOT ALLOWED (a)	The ESCAPE ON ABORT phrase is not present; therefore, the abort-input control character is not effective. ACCEPT processing continues from where the control character is entered.

NOTE

(a) These advisory messages are displayed only for terminals operating in conversational mode.

An installation can replace the PATHWAY advisory message routine with a routine of its own. This might be done for the purpose of changing the text of the messages or adding error messages for use in association with user-provided conversion procedures.

To change the routine, the installation must write a procedure in the Tandem Transaction Language (TAL) and use the UPDATE program to store the procedure in the TCP object file. The declaration for the procedure is as follows:

```
PROC ADVISORY^MESSAGE( MSGNUM, BUF, MESSLEN );
INT      MSGNUM;   ! THE ERROR NUMBER
STRING .BUF;       ! PLACE MESSAGE HERE
INT    .MESSLEN;   ! RETURN MESSAGE LENGTH HERE (MAX 255)
```

The *MSGNUM* parameter is the internal message number as given in Table A-1, or as returned by the user conversion procedure. If new error/message numbers are to be used (via user conversion procedures), the numbers should be larger than 100 to avoid conflict with future PATHWAY numbers.

The *BUF* parameter is a string buffer where the text associated with *MSGNUM* should be placed. The text cannot be longer than 255 characters.

The *MESSLEN* parameter should be set to the length of the text returned.

DIAGNOSTIC SCREENS

Diagnostic screens are displayed to inform the terminal operator if an error condition or termination occurs. Diagnostic screens are displayed unless the PATHCOM SET TERM command DIAGNOSTIC parameter is set off. When the parameter is set on (the default setting), the special register DIAGNOSTIC-ALLOWED is initialized to YES.

Screen recovery is invoked following display of a diagnostic screen. This is especially important if the diagnostic screen is displayed because of an error during a PRINT SCREEN sequence.

The default diagnostic screen has the following form:

```
row  ┌─ ─ ─ ─ ─ ─ ─ ─ ─ ─ ─ ─ ─ ─ ─ ─ ─ ─ ─ ─
 1   | PATHWAY ERROR REPORT: timestamp
     |
 3   | TERMINAL: termname
     |
 5   | diagnostic-message
 6   |   [ device-name ]
 7   |   [ retry-info
```

Diagnostic Screen Messages

Table A-2 lists and describes the standard diagnostic messages and indicates the conditions that invoke the messages.

Table A-2. Diagnostic Screen Messages

Diagnostic-message	Meaning
TERMINAL STOPPED BY PROGRAM	The terminal stopped because the highest level program unit was exited.
TERMINAL STOPPED BY SYSTEM OPERATOR	The terminal was stopped or aborted by command from the system operator.
TERMINAL SUSPENDED BY SYSTEM OPERATOR	The terminal was suspended by command from the system operator.
TERMINAL SUSPENDED FOR SYSTEM ERROR	The terminal was suspended because an error occurred during program execution.
TERMINAL STOPPED FOR SYSTEM ERROR	The terminal was suspended without possibility of restart because an error occurred during program execution.
PRINTER BUSY	The print device that is the target of a PRINT SCREEN statement is currently in use.
PRINTER REQUIRES ATTENTION	The print device that is the target of a PRINT SCREEN statement needs to be placed into the ready state.
---	---
Default value for device-name is	PRINTER: filename
Default value for retry-info is	PRESS f1 TO RETRY, f2 TO ABORT f1 = F1 for T16-6510, T16-6520, and T16-6530; and PA1 for IBM-3270 f2 = F2 for T16-6510, T16-6520, and T16-6530; and PA2 for IBM-3270

Diagnostic Message Generation Procedure

Installations can replace the PATHWAY-generated diagnostic messages. For example, messages can be displayed in another language.

To change the messages, the PATHTCP routine DIAGNOSTIC^MESSAGE must be replaced by a user-written routine having the same name. This is handled in the same manner as the ADVISORY^MESSAGE procedure previously described in this appendix.

The declaration for the DIAGNOSTIC^MESSAGE procedure is as follows:

```
PROC DIAGNOSTIC^MESSAGE( DIAG^FORMAT, MESSAGE, MSGLEN, CONTEXT );
INT    .DIAG^FORMAT( DIAG^FORMAT^DEF );
                   ! Byte addressable diagnostic info struct.
STRING .MESSAGE;   ! Returned - Message to display (byte addr).
INT    .MSGLEN;    ! Returned - Length in bytes of message.
INT    .CONTEXT;   ! One word of "OWN" storage.
```

The procedure is called repeatedly to initialize the screen; one call for each row of the screen. The parameter *DIAG^FORMAT*, which is described in Figure A-1, defines the the error condition and the sequencing to build the screen. The parameter *CONTEXT* provides one word of storage that is not altered between successive calls to initialize a given screen; the parameter is set to zero prior to the first call in the initialization sequence.

```
STRUCT DIAG^FORMAT^DEF( * );
BEGIN                                ! - All string arrays are blank
                                     !   padded.
  STRING CLASS;                      ! Class: 1 = IBM-3270, 2 = T16-6510,
                                     !        3 = T16-6520, 4 = T16-6530.
  STRING SUBCLASS;                   ! Subclass for IBM-3270 (screen size)
                                     !       0 = 24 X 80 (NOT IBM^3270),
                                     !       1 = 12 X 40,
                                     !       2 = 24 X 80,
                                     !       3 = 24 X 80 - ALT 32 X 80,
                                     !       4 = 24 X 80 - ALT 43 X 80,
                                     !       5 = 12 X 40 - ALT 12 X 80,
  INT     ROW;                       ! row of screen format [1:NROWS].
                                     !   (Incr by one on each call to
                                     !    DIAGNOSTIC^SCREEN.)
  INT     ERRTYPE;                   ! error type.
                                     !   See DIAG^ERRTYPE^??? below.
  STRING LOG^TERM^NAME
         [ 0:NAME^LEN-1 ];           ! Pathway terminal name.
  STRING TERM^PRINTER[ 0:35 ];       ! Printer name, external form.
  INT    ERRNUM;                     ! Error number of suspension cause.
                                     !   (3000 to 3999).
  INT    ERRINFO;                    ! Additional error info.
  STRING PUNAME
         [ 0:15 ];                   ! Current program-unit name.
  INT    PUVERSION;                  ! Version of program unit.
  INT    INSTR^ADDR;                 ! Address of instruction at susp.
  STRING INSTR^CODE[ 0:19 ];         ! Instruction at suspension.
 END;

LITERAL ! DIAGNOSTIC DISPLAY ERROR TYPES.
  DIAG^ERRTYPE^STOP^BY^PROG     = 1, ! Term stopped by program.
  DIAG^ERRTYPE^STOP^BY^OP       = 2, ! Term stopped by operator.
  DIAG^ERRTYPE^ABRT^BY^OP       = 3, ! Term aborted by operator.
  DIAG^ERRTYPE^SUSP^BY^ERR      = 4, ! Term susp because of error.
  DIAG^ERRTYPE^SUSP^BY^ERR^NRS  = 5, ! Term susp because of error,
                                      !   not resumable.
  DIAG^ERRTYPE^SUSP^BY^OP       = 6, ! Term suspended by operator.
  DIAG^ERRTYPE^ATTN             = 7, ! Printer Requires attention.
  DIAG^ERRTYPE^BUSY             = 8; ! Printer in use.
```

Figure A-1. DIAGNOSTIC-FORMAT Parameter for Diagnostic Message Generation

SCREEN COBOL COMPILER DIAGNOSTIC MESSAGES

THE SCREEN COBOL compiler produces three kinds of diagnostic messages to report problems in the source text or compilation process. The message types WARNINGS, ERRORS, and FAILURES reflect the severity of the problem.

A warning message reports a questionable condition, but does not inhibit code generation. Some warnings merely report a minor deviation from the conventions of the SCREEN COBOL language. Other warnings indicate more important violations that could result in a different interpretation of the program than is intended. The explanation of a warning includes a brief description and any actions taken or assumptions made by the compiler. Warning messages can be suppressed with the NOWARN compiler command, as described in Section 7.

An error message reports a serious violation of SCREEN COBOL syntax or semantics. The compiler stops generating code and deletes any previously generated code; however, compilation continues for syntax checking purposes. Since information at this point would be incomplete or incorrect, correct syntax might be reported as an error.

A failure message reports a condition so severe that the compiler cannot continue. Any previously generated code is deleted.

Most warnings or errors pertain to a specific portion of the source text or a specific user-defined item. The compiler assists in locating the error as follows:

- When the problem is local, the line preceding the message contains a caret (^). The language element in error is in the last source line either at the position indicated or somewhere to the left. Occasionally, the language element to the left is actually on a source line preceding the last one listed.

- Some problems are not found until the entire program is examined. When the line preceding the message contains the phrase PROBLEM AT OR NEAR LINE nnnnn, it refers to a preceding portion of the program by line number. The cause of the problem, or one of several interrelated causes, will be found in the vicinity of the specified line.

- When a user-defined name appears at the end of a message, the message concerns the item specified.

SCREEN COBOL compiler error messages are listed and described in Table A-3. The explanations describe the problem in further detail or describe the language rule violated. When the same message can refer to different problems, the discussion includes several independent explanations.

With the exception of the ILLEGAL SYNTAX message, all messages carry a code number; are preceded by the word FAILURE, WARNING, or ERROR; and are surrounded by asterisks. For example:

```
** FAILURE nnn **
** WARNING nnn **
** ERROR nnn **
```

The Type column in Table A-3 indicates which word will appear by specifying an F (FAILURE), W (WARNING), or E (ERROR).

Table A-3. SCREEN COBOL Compiler Error Messages

Type	No.	Message	Meaning
		ILLEGAL SYNTAX	The sequence of character strings and separators does not conform to SCREEN COBOL language syntax. Misspelled reserved words are a common cause. The compiler cannot always recover to a known context after a syntax error; if the compiler fails in the attempt, following diagnostics might not be valid.
F	0	TOO MANY ERRORS	The number of ERROR diagnostics exceeds the limit specified (the default limit is 100).
F	1	UNABLE TO INVOKE COMPILER PROCESS	The compiler is unable to invoke one of its processes. The error code returned by the GUARDIAN NEWPROCESS procedure (bits 0-7) is appended to the message.
F	2	UNABLE TO OPEN $RECEIVE	The compiler is unable to open the job communication file. The error code returned by the GUARDIAN operating system is appended to the message.
F	3	UNABLE TO OPEN COMMUNICATION FILE	The compiler is unable to open the interprocess communication file. The error code returned by the GUARDIAN operating system is appended to the message.
F	4	UNABLE TO OPEN (SOURCE/LIST) FILE	The compiler is unable to open the specified file. The error code returned by the GUARDIAN operating system is appended to the message.
F	5	UNABLE TO USE (SOURCE/LIST) FILE	(1) The source file does not have read capability or the list file does not have write capability. (2) Access to the source file, an EDIT disc file, failed. The error code returned from the attempt to access the file is appended to the message. (3) The record length of the list file is less than 40 bytes (characters), or the list device is a printer/process and the initial control operation failed.
F	6	UNABLE TO CREATE WORK FILE	The compiler is unable to create one of its work files. The error code returned by the GUARDIAN operating system is appended to the message.
F	7	UNABLE TO OPEN WORK FILE	The compiler is unable to open one of its work files. The error code returned by the GUARDIAN operating system is appended to the message.
F	8	UNABLE TO OPEN COPY FILE	The compiler is unable to open a COPY library file. The error code returned by the GUARDIAN operating system is appended to the message.

Table A-3. SCREEN COBOL Compiler Error Messages (Continued)

Type	No.	Message	Meaning
F	9	UNABLE TO USE COPY FILE	(1) The default COPY library file name is not a legal Tandem file name. (2) The COPY library file is not an EDIT disc file, or has been modified since the start of this compilation. (3) An attempt to access the COPY library file failed. The error code returned from the attempt to access the file is appended to the message.
F	10	COMPILER COMMUNICATION LOST	Communication between compiler processes failed. The error code returned by the GUARDIAN operating system is appended to the message. If the code is 0, one of the compiler processes has ABENDed.
F	11	(SOURCE/LIST) FILE (READ/WRITE) FAILURE	The compiler is unable to access the specified file. The error code returned by the GUARDIAN operating system is appended to the message.
F	12	SOURCE FILE EDITREAD FAILURE	A read issued to the source file failed. The error code returned from the attempt to access the file is appended to the message.
F	13	COPY FILE EDITREAD FAILURE	A read issued to the COPY library file failed. The error code returned from the attempt to access the file is appended to the message.
F	14	UNABLE TO CREATE RUN UNIT FILE	The compiler is unable to create the object file. The error code returned by the GUARDIAN operating system is appended to the message.
F	15	UNABLE TO OPEN RUN UNIT FILE	The compiler is unable to open the object file. The error code returned by the GUARDIAN operating system is appended to the message.
F	17	COMPILER LOGIC ERROR	Internal consistency checking has discovered an error in the compiler logic. Report this failure to a Tandem Computers representative.
F	18	DICTIONARY OVERFLOW	Compiler dictionary space is insufficient for the number of items defined in the current program unit. The deficiency might be corrected by invoking the SCREEN COBOL compiler with a larger value for the MEM parameter. If the failure persists when MEM 64 is used, the program must be subdivided into smaller program units.
F	19	FILE ERROR ON WORK FILE	An operation on a compiler work file failed. The error code returned by the GUARDIAN operating system is appended to the message.

Table A-3. SCREEN COBOL Compiler Error Messages (Continued)

Type	No.	Message	Meaning
F	20	PROGRAM DATA SPACE OVERFLOW	Either the allocation requirements of the data items in a single program unit or the cumulative requirements for the object file exceed the maximum program data space available to SCREEN COBOL.
F	21	CONTROL DATA SPACE OVERFLOW	For each program unit, the SCREEN COBOL compiler allocates an auxiliary data space used for control purposes. The cumulative requirements for these control data spaces exceed the maximum available to SCREEN COBOL.
F	22	PROGRAM CODE SPACE OVERFLOW	Either the code requirements for a single program unit or the cumulative requirements for the entire object file exceed the maximum code space available to SCREEN COBOL.
F	23	FILE ERROR ON RUN UNIT FILE	An operation on the object file failed. The error code returned by the GUARDIAN operating system is appended to the message.
E	26	MISSING QUOTE CHARACTER	The terminating quotation mark character is missing from a nonnumeric literal.
E	27	NULL LITERAL	A nonnumeric literal contains no characters (has no value).
E	28	LITERAL EXCEEDS 120 CHARACTERS	A nonnumeric literal contains more than 120 characters.
E	29	LITERAL EXCEEDS 18 DIGITS	A numeric literal contains more than 18 digits.
E	30	WORD EXCEEDS 30 CHARACTERS	A SCREEN COBOL word contains more than 30 characters.
W	31	NOT SUPPORTED	SCREEN COBOL does not support some of the optional elements of the ANSI COBOL language. The message probably refers to one of the following language elements, which are not normally critical to correct program execution: (1) The Rerun facility. (2) File labels. (SCREEN COBOL does not have file handling capability.) (3) More than one system name in an ASSIGN clause. (NO ASSIGN)
E	31	NOT SUPPORTED	SCREEN COBOL does not support some of the optional elements of the ANSI COBOL language.
W	32	ILLEGAL CONTEXT FOR RESERVED WORD	A SCREEN COBOL reserved word is used as the text name or library name in a COPY statement.

Table A-3. SCREEN COBOL Compiler Error Messages (Continued)

Type	No.	Message	Meaning
E	32	ILLEGAL CONTEXT FOR RESERVED WORD	The indicated SCREEN COBOL reserved word cannot appear in this context. The cause for this message might be an attempt to define one of the reserved words as a user-defined name.
E	33	ILLEGAL CHARACTER	The character indicated is not permitted in this context. Since the character might be non-printing, the internal value of the character is listed with the message.
E	34	TOKEN EXCEEDS 120 CHARACTERS	An entry considered to be a character string contains more than 120 characters. If the character string is actually several adjacent language elements, the problem can be corrected by inserting blanks to separate them.
W	35	BLANK CONTINUATION LINE	A source line marked as a continuation line contains only blanks.
W	36	ILLEGAL INDICATOR CHARACTER	(1) The character in the indicator field of a source line is not – * / ? or blank. (2) A continuation line appears as part of a comment entry in a paragraph of the Identification Division.
W	37	MISSING SEPARATOR	(1) A character string is not followed by a separator. (2) A comma, semicolon, or period separator is not followed by a blank.
E	38	UNEXPECTED TEXT	(1) A section header or division header is followed by other text on the same source line. (2) The program-name in the PROGRAM-ID paragraph of the Identification Division is followed by other text on the same source line. (3) The Identification Division header or the PROGRAM-ID paragraph must be followed by an Identification Division paragraph header or the Environment Division header, and it must begin in Area A of the source line.
E	39	UNEXPECTED END OF TEXT	The source text ended before the appearance of all four required divisions.
E	40	INCORRECT NUMBER OF PARAMETERS	The number of operands in the USING clause of a CALL statement differs from the number of names in the USING Division header for the SCREEN COBOL subprogram it invokes.

Table A-3. SCREEN COBOL Compiler Error Messages (Continued)

Type	No.	Message	Meaning
E	41	NAME CONFLICT	(1) The definition of a user-defined name in one class conflicts with its prior definition in another class. (2) The name of a new data item cannot be distinguished from the name of a previous data item, even with full qualification.
E	42	AMBIGUOUS REFERENCE	A reference has insufficient qualifications to identify a unique object within the program unit.
E	50	EXPECTED 'IDENTIFICATION'	A SCREEN COBOL program unit must begin with an Identification Division header. The reserved word IDENTIFICATION must start in Area A of the source line.
E	51	EXPECTED UNSIGNED INTEGER	(1) A numeric literal in this context must be an unsigned integer. (2) Only an unsigned integer numeric literal is permitted in this context.
E	52	0 NOT PERMITTED IN THIS CONTEXT	The indicated integer numeric literal cannot be zero in this context.
E	53	INTEGER NOT IN EXPECTED RANGE	(1) The value of the integer numeric literal is too small for this context. (2) The value of the integer numeric literal is too large for this context.
E	54	ILLEGAL RANGE	(1) The first value in a numeric range exceeds the last value. (2) The first value in a nonnumeric range is greater than the last value.
W	55	OUT OF ORDER	The position of a phrase, clause, or paragraph does not conform to SCREEN COBOL language requirements.
E	55	OUT OF ORDER	(1) The REDEFINES clause must be the first clause in a data description entry. (2) A section of the Data Division occurs out of order.
E	56	DUPLICATE PHRASE	The indicated phrase duplicates the function of a preceding one.
E	57	DUPLICATE CLAUSE	The indicated clause duplicates the function of a preceding one.
E	58	DUPLICATE PARAGRAPH	The indicated paragraph header duplicates a preceding one.

Table A-3. SCREEN COBOL Compiler Error Messages (Continued)

Type	No.	Message	Meaning
E	59	DUPLICATE SECTION	The indicated section header duplicates a preceding one.
E	61	EXPECTED COMMAND WORD	A compiler command line must begin with the keyword of a command or one of the command-options defined for the OPTION command.
E	62	EXPECTED QUOTED STRING	The heading value in a HEADING command-option must be a quoted string (nonnumeric literal).
E	63	EXPECTED COMMA	(1) Compiler command-options must be separated by commas. (2) Toggle numbers in a SETTOG or RESETTOG command must be separated by commas.
E	64	MISSING TEXT NAME	(1) The text name is missing from a SECTION command. (2) The text name in a COPY statement cannot be found in the copy library.
E	65	COMMAND NOT PERMITTED AFTER OPTION	A command keyword follows one or more command-options. Only a single command is permitted on each command line.
E	66	TEXT NOT PERMITTED AFTER COMMAND	Additional text follows a complete command. Only a single command is permitted on each command line.
E	70	MISSING PROGRAM ID	The required PROGRAM-ID paragraph of the Identification Division is missing.
E	71	MISSING CONFIGURATION SECTION	The required Configuration Section of the Environment Division is missing.
W	72	MISSING SOURCE COMPUTER PARAGRAPH	The Configuration Section should contain the SOURCE-COMPUTER paragraph. The compiler assumes: SOURCE-COMPUTER. TANDEM/16.
W	73	MISSING OBJECT COMPUTER PARAGRAPH	The Configuration Section should contain the OBJECT-COMPUTER paragraph. The compiler assumes: OBJECT-COMPUTER. TANDEM/16.
E	77	ILLEGAL CURRENCY SYMBOL	Either the alternative currency symbol specified is not a single character or the specified character is not among the set permitted for this purpose.
E	94	ALPHABET NAME NOT FOUND	The indicated name is either not defined or not an alphabet name.

Table A-3. SCREEN COBOL Compiler Error Messages (Continued)

Type	No.	Message	Meaning
E	111	CLAUSE NOT PERMITTED FOR THIS ENTRY	The indicated clause appears in an entry whose level number prohibits it.
E	112	NOT PERMITTED IN THIS SECTION	A VALUE clause defining an initial value appears in the Linkage Section of the Data Division.
E	113	ILLEGAL LEVEL NUMBER	A level number is not 66, 77, 88, or in the range 01 to 49. The compiler converts the illegal level number to 50.
E	114	INCONSISTENT LEVEL NUMBER	A level number is neither greater than the level number of the preceding data description entry nor equal to that of some preceding data description entry in the same data structure.
E	115	MISSING 01 LEVEL ENTRY	A level number in the range 02-49 is not subordinate to a data description entry with level number 01; that is, it is not within a data structure.
E	116	PRECEDED BY VARIABLE OCCURRENCE TABLE	Within a data structure, the data description entry for a variable occurrence table (one containing an OCCURS clause with a range) cannot be followed by a data description entry with a lower level number.
E	117	NOT PRECEDED BY CONDITIONAL VARIABLE	The definition of a condition-name (a name whose data description entry has level number 88) must be preceded by the entry of the data item whose value it tests. Any intervening data description entries must also have level number 88.
E	118	NOT PRECEDED BY RECORD	A data description entry with level number 66 must be preceded by a data structure. Any intervening data description entries must also have level number 66.
E	119	FILLER PERMITTED ONLY FOR ELEMENTARY RECORD ITEM	The data description entry of a FILLER data item must have a level number in the range 02-49 and cannot be followed by descriptions of subordinate data items; that is, it must be an elementary data item defined within a data structure.
W	120	DO NOT QUOTE PICTURE STRING	A PICTURE character string should not be written as a nonnumeric literal. The SCREEN COBOL compiler accepts the contents of the nonnumeric literal as the PICTURE character string.

Table A-3. SCREEN COBOL Compiler Error Messages (Continued)

Type	No.	Message	Meaning
E	121	PICTURE STRING EXCEEDS 30 CHARACTERS	The SCREEN COBOL language limits the representation of a PICTURE character string to 30 characters. Data items with more character positions must be described with parenthesized repetition counts.
E	122	TOO MANY DIGIT POSITIONS	The SCREEN COBOL language supports a maximum of 18 digits in a numeric or numeric edited data item.
E	123	TOO MANY CHARACTER POSITIONS	SCREEN COBOL supports a maximum of 16383 character positions for an elementary data item.
E	124	ILLEGAL PICTURE STRING	The PICTURE character string does not conform to SCREEN COBOL syntax. Some of the causes for this message are illegal characters, unmatched parentheses, improper combinations of otherwise legal characters, and pictures with no positions for data characters.
W	125	LAST SYMBOL IS ',' OR '.'	After stripping the terminating comma, semicolon, period, or blank, the last character in the PICTURE character string is a comma or decimal point. The compiler accepts the picture and interprets the character in conformance with the presence or absence of the DECIMAL-POINT IS COMMA clause in the Special-Names paragraph.
E	127	SUBORDINATE USAGE CONFLICTS WITH GROUP USAGE	The data description entry for a containing group item has a USAGE clause. The description of the subordinate data item cannot specify a different usage.
E	128	DISPLAY USAGE REQUIRED IN GROUP WITH VALUE OR CONDITION NAME	The data description entry of a containing group item has a VALUE clause specifying an initial value or is followed by entries defining condition-names for the group item. The subordinate data item must have DISPLAY usage.
E	129	COMPUTATIONAL USAGE REQUIRES NUMERIC	The category of a data item must be numeric when its usage is COMPUTATIONAL.
E	130	SUBORDINATE SIGN CONFLICTS WITH GROUP SIGN	The data description entry for a containing group item has a SIGN clause. The description of the subordinate data item cannot specify different sign characteristics.

Table A-3. SCREEN COBOL Compiler Error Messages (Continued)

Type	No.	Message	Meaning
E	131	SIGN CLAUSE REQUIRES DISPLAY USAGE	(1) The data description entry for a containing group item has a SIGN clause. The subordinate numeric data item is signed (has an S in its picture); therefore, the item must have DISPLAY usage. (2) The data description entry for the current data item has a SIGN clause; therefore, the item must have DISPLAY usage.
E	132	SIGN CLAUSE REQUIRES SIGNED NUMERIC	The data description entry for the current data item has a SIGN clause; therefore, the item picture must specify category numeric and contain an S.
E	133	JUSTIFIED REQUIRES DISPLAY USAGE	A data item described with the JUSTIFIED clause must have DISPLAY usage.
E	134	JUSTIFIED NOT PERMITTED FOR NUMERIC OR EDITED	The JUSTIFIED clause cannot appear for a data item described as numeric.
E	135	JUSTIFIED NOT PERMITTED IN GROUP WITH VALUE OR CONDITION NAME	The data description entry of a containing group item has a VALUE clause specifying an initial value or is followed by entries defining condition names for the group item. The subordinate data item cannot be described with the JUSTIFIED clause.
E	137	SYNCHRONIZED NOT PERMITTED IN GROUP WITH VALUE OR CONDITION NAME	The data description entry of a containing group item has a VALUE clause specifying an initial value or is followed by entries defining condition names for the group item. The subordinate data item cannot be described with the SYNCHRONIZED clause.
E	138	BLANK WHEN ZERO REQUIRES DISPLAY USAGE	A data item described with the BLANK WHEN ZERO clause must have DISPLAY usage. (BLANK WHEN ZERO syntax is enforced when used, but data items using this syntax cannot be accessed by SCREEN COBOL programs.)
E	139	BLANK WHEN ZERO REQUIRES NUMERIC OR NUMERIC EDITED	Only a numeric data item can be described with the BLANK WHEN ZERO clause. (BLANK WHEN ZERO syntax is enforced when used, but data items using this syntax cannot be accessed by SCREEN COBOL programs.)
E	140	BLANK WHEN ZERO NOT COMPATIBLE WITH '*'	A data item cannot be described with both the BLANK WHEN ZERO clause and a picture containing the asterisk (*) symbol. (BLANK WHEN ZERO syntax is enforced when used, but data items using this syntax cannot be accessed by SCREEN COBOL programs.)

Table A-3. SCREEN COBOL Compiler Error Messages (Continued)

Type	No.	Message	Meaning
E	141	TOO MANY NESTED TABLES	The SCREEN COBOL language supports access to a data item with at most three subscripts. The OCCURS clause is subordinate to three or more other OCCURS clauses and thus would require four or more subscripts to access the data item it describes.
E	142	VARIABLE OCCURRENCE NOT PERMITTED FOR SUBORDINATE TABLE	The SCREEN COBOL language does not permit a variable occurrence table to be subordinate to a group table item.
E	143	VARIABLE OCCURRENCE NOT PERMITTED IN REDEFINITION	A data item described in a redefinition cannot be a variable occurrence table.
E	144	VARIABLE OCCURRENCE NOT COMPATIBLE WITH GROUP INITIAL VALUE	The data description entry of a containing group item has an initial value. The subordinate data item cannot be a variable occurrence table.
E	146	SUBORDINATE VALUE NOT PERMITTED WITH GROUP VALUE	The data description entry of a containing group item specifies an initial value for the group item. The subordinate data item cannot also specify an initial value.
E	147	ONLY ONE INITIAL VALUE PERMITTED	A data item cannot be initialized with more than one value.
E	148	RANGE NOT PERMITTED FOR INITIAL VALUE	A data item cannot be initialized with a range of values.
E	150	INITIAL VALUE NOT PERMITTED FOR TABLE ITEM	A data item that is described with an OCCURS clause or is subordinate to a group table item cannot be initialized.
E	151	INITIAL VALUE NOT PERMITTED FOR REDEFINITION	A data item described in a redefinition cannot be initialized.
E	152	SIGNIFICANCE RANGE OF LITERALS EXCEEDS 18 DIGITS	The number of significant digits to the left of the decimal point in one literal plus the number of significant digits to right of the decimal point in another literal exceeds 18.
E	153	NUMERIC LITERAL NOT COMPATIBLE WITH NONNUMERIC FIGURATIVE OR LITERAL	When a VALUE clause contains a numeric literal, all other values must also be numeric literals or one of the figurative constants ZERO, ZEROS, or ZEROES.

Table A-3. SCREEN COBOL Compiler Error Messages (Continued)

Type	No.	Message	Meaning
E	154	RENAME OBJECT NOT DATA ITEM	The RENAMES clause can only rename data items.
E	155	RENAME OBJECT IS 66 LEVEL ITEM	A RENAMES clause cannot rename a level 66 item.
E	156	RENAME OBJECT NOT SUBORDINATE TO PRECEDING RECORD	A data item referenced in the RENAMES clause must be defined within the preceding data description.
E	157	RENAME OBJECT IN TABLE OR HAS VARIABLE SIZE	The RENAMES clause cannot reference either a table item or a group data item that has a variable size (has a subordinate variable occurrence table).
E	158	ILLEGAL RENAMES OBJECT RANGE	The second data item in the range of a RENAMES clause must include some character positions that are not part of the first data item. However, the initial character position of the second data item cannot precede the initial character position of the first data item within their data structure.
E	159	REDEFINITION OBJECT NOT FOUND	Either the name in the REDEFINES clause cannot be found or it is not the name of a data item. Note that when a REDEFINES clause appears in a data structure, only that data item is searched for the data item to be redefined.
E	160	REDEFINITION OBJECT HAS CONFLICTING LEVEL NUMBER	The data item to be redefined must have the same level number as the redefining data description entry.
E	161	REDEFINITION OBJECT IS REDEFINITION	A data item described with a REDEFINES clause cannot itself be redefined. This restriction does not apply to a subordinate of a redefinition item unless its data description entry also contains a REDEFINES clause.
E	162	REDEFINITION OBJECT AND REDEFINITION NOT SUBORDINATE TO SAME LEVELS	When the redefined data item is subordinate to a set of group items, the redefinition item must also be subordinate to them.
E	163	REDEFINITION OBJECT NOT PRECEDING ITEM AT THIS LEVEL	The data description entry of a redefinition must not be separated from that of the redefined item by any other data description entry with the same level number, unless the intervening entry redefines the same data item.

Table A-3. SCREEN COBOL Compiler Error Messages (Continued)

Type	No.	Message	Meaning
E	164	REDEFINITION OBJECT IS TABLE OR HAS VARIABLE SIZE	A table item or a group item that has a variable size (has a subordinate variable occurrence table) cannot be redefined.
E	165	MISSING VALUE CLAUSE	The required VALUE clause is missing from a data description entry with level number 88.
E	166	MISSING RENAMES CLAUSE	The required RENAMES clause is missing from a data description entry with level number 66.
E	167	GROUP ITEM HAS ELEMENTARY ITEM CLAUSE	The data description entry of a group item has a BLANK WHEN ZERO, JUSTIFIED, SYNCHRONIZED, or PICTURE clause. These clauses can only describe an elementary data item. (BLANK WHEN ZERO syntax is enforced when used, but data items using this syntax cannot be accessed by SCREEN COBOL programs.)
W	168	GROUP WITH SIGN CLAUSE HAS NO SIGNED NUMERIC SUBORDINATE	The SCREEN COBOL language requires a group data item described with a SIGN clause to have at least one signed numeric subordinate data item. SCREEN COBOL reports nonconformance for informational purposes only.
E	169	ELEMENTARY ITEM HAS NO PICTURE	An elementary data item must be described with a PICTURE clause.
E	174	FIRST ELEMENTARY ITEM NOT DISPLAY AND NOT ALIGNED	The indicated data item cannot be aligned to the first character position of the area it redefines. SCREEN COBOL does not permit a redefinition that requires allocation of implicit FILLER character positions to align the first elementary item.
E	175	REDEFINITION HAS INCORRECT SIZE	The number of character positions occupied by a redefinition must equal the number of character positions occupied by the redefined data item(s), unless the redefinition begins at the 01 level.
E	176	NONNUMERIC FIGURATIVE OR LITERAL NOT PERMITTED FOR NUMERIC ITEM	The initial value for a numeric data item must be a numeric literal or one of the figurative constants ZERO, ZEROS, or ZEROES.
E	177	SIGNED LITERAL NOT PERMITTED FOR UNSIGNED NUMERIC ITEM	The initial value for an unsigned numeric data item must be an unsigned numeric literal or one of the figurative constants ZERO, ZEROS, or ZEROES.
E	178	TOO MANY FRACTION DIGITS IN NUMERIC LITERAL	Assignment of the initial value to the numeric data item would require truncation of nonzero digits to the right of the decimal point.

Table A-3. SCREEN COBOL Compiler Error Messages (Continued)

Type	No.	Message	Meaning
E	179	NUMERIC LITERAL VALUE TOO LARGE FOR ITEM	Assignment of the initial value to the numeric data item would require truncation of nonzero digits to the left of the decimal point.
E	180	NUMERIC LITERAL NOT PERMITTED FOR NONNUMERIC OR GROUP ITEM	A numeric literal can only be used as the initial value for an elementary numeric data item.
E	181	NONNUMERIC LITERAL EXCEEDS ITEM SIZE	Assignment of the initial value to the indicated data item would require truncation of one or more characters.
E	182	01 OR 77 LEVEL ITEM TOO LARGE	SCREEN COBOL supports a maximum of 16383 character positions for a level 01 or level 77 data item defined in the Working-Storage Section or Linkage Section.
E	190	DEPENDING ITEM NOT FOUND	A name referenced in the DEPENDING phrase of an OCCURS clause is not defined.
E	191	DEPENDING ITEM NOT SIMPLE UNSIGNED INTEGER DATA ITEM	Either the indicated name (referenced in the DEPENDING phrase of an OCCURS clause) does not identify an elementary unsigned integer data item, or access to the item requires subscripting.
E	192	DEPENDING ITEM IN TABLE	The indicated data item is allocated within the table it controls. The allocation is a result of an explicit or implicit redefinition.
E	205	USING OPERAND NOT FOUND IN LINKAGE SECTION	A name referenced in the USING phrase of the Procedure Division header is not defined in the Linkage Section of the Data Division.
E	206	USING OPERAND NOT DATA ITEM	The indicated name does not identify a data item. Only the names of data items can be specified in the USING phrase of the Procedure Division header.
E	207	USING OPERAND IS REDEFINITION OR NOT LEVEL 01 OR LEVEL 77 DATA ITEM	SCREEN COBOL requires that a data item in the USING phrase be a level 01 or level 77 item. SCREEN COBOL does not permit a redefinition, including one of a level 01 or level 77 item, to appear in the USING phrase.
E	208	DATA ITEM PERMITTED ONLY ONCE AS USING OPERAND	The same name cannot appear more than once in the USING phrase of the Procedure Division header.
E	209	TOO MANY USING OPERANDS	SCREEN COBOL supports a maximum of 29 names in the USING phrase of the Procedure Division header.

Table A-3. SCREEN COBOL Compiler Error Messages (Continued)

Type	No.	Message	Meaning
E	210	LINKAGE DATA ITEM MUST BE USING OPERAND	The indicated data item is defined in the Linkage Section but cannot be addressed. Addressable items are those specified in the USING phrase of the Procedure Division header; their subordinate items; and the redefinition, renaming, and condition-names of the subordinate items.
E	211	TOO MANY RECORDS	The program defines more data stuctures than the compiler can address.
E	212	TOO MANY ELEMENTARY ITEMS	The program defines more level 77 data items than the SCREEN COBOL compiler can address.
E	221	INSUFFICIENT SPECIFICATION TO DETERMINE TYPE OF SCREEN ITEM	Any screen item (other than the 01 level screen name) must be either a group, an overlay area, a literal field, an input field, an output field, or an input-output field. This item cannot be classified because it does not have the minimum requirements for definition.
E	222	THIS SCREEN ITEM MUST BE NAMED	A name is required for 01 levels (screen names) and overlay areas.
E	223	THIS SCREEN ITEM MUST HAVE BASE SPECIFICATION	The screen item must have a BASE clause specified.
E	224	THIS SCREEN ITEM MUST HAVE 'OVERLAY' SPECIFICATION	The screen item must have an OVERLAY clause specified.
E	225	THIS SCREEN ITEM MUST HAVE SIZE SPECIFIED	Overlay areas must have a SIZE clause.
E	226	THIS SCREEN ITEM MUST HAVE 'AREA' SPECIFICATION	Overlay areas must have an AREA clause.
E	227	THIS SCREEN ITEM MUST HAVE LOCATION ('AT') SPECIFIED	All screen items must have a location specified. Screen fields can use either the AT clause or the REDEFINES clause or both.
E	228	THIS SCREEN ITEM MUST HAVE LOCATION ('REDEFINES') SPECIFIED	All screen items must have a location specified. Screen fields can use either the AT clause or the REDEFINES clause or both.

Table A-3. SCREEN COBOL Compiler Error Messages (Continued)

Type	No.	Message	Meaning
E	229	THIS SCREEN ITEM MUST HAVE FROM (OR USING) DATA ITEM	The screen item must have a FROM or USING clause specified with an associated data item.
E	230	THIS SCREEN ITEM MUST HAVE TO (OR USING) DATA ITEM	The screen item must have a TO or USING clause specified with an associated data item.
E	231	THIS SCREEN ITEM MUST HAVE SHADOW DATA ITEM SPECIFIED	The screen item must have a SHADOW clause specified with an associated data item.
E	232	THIS SCREEN ITEM MUST HAVE PICTURE SPECIFICATION	Input fields, output fields, and input-output fields must have a PICTURE clause.
E	233	THIS SCREEN ITEM MUST HAVE INITIAL VALUE	Literal fields must have an initial value.
E	234	THIS SCREEN ITEM MUST HAVE FILL CHARACTER SPECIFIED	The screen item must have a FILL clause specified with a fill character.
E	235	THIS SCREEN ITEM MUST HAVE OCCURS SPECIFICATION	The screen item must have an OCCURS clause specified.
E	236	THIS SCREEN ITEM MUST HAVE ACCEPTABLE VALUE(S) ('MUST') SPECIFIED	The MUST BE clause for the screen item must specify a value that is compatible with the screen PICTURE clause.
E	237	THIS SCREEN ITEM MUST HAVE ACCEPTABLE LENGTH(S) SPECIFIED	The LENGTH clause for the screen item must specify a length that is compatible with the screen PICTURE clause.
E	238	THIS SCREEN ITEM MUST HAVE UPSHIFT SPECIFICATION	The screen item must have an UPSHIFT clause specified with a valid input or output specification.
E	239	THIS SCREEN ITEM MUST HAVE FULL ACTION ('WHEN FULL') SPECIFIED	The screen item must have a WHEN FULL clause specified.

Table A-3. SCREEN COBOL Compiler Error Messages (Continued)

Type	No.	Message	Meaning
E	240	THIS SCREEN ITEM MUST HAVE USER CONVERSION NUMBER SPECIFIED	The screen item must have a USER CONVERSION clause specified.
E	243	THIS SCREEN ITEM MUST HAVE PROMPT FIELD SPECIFIED	The screen item must have a PROMPT clause specified in the Screen Section.
E	254	THIS SCREEN ITEM MUST NOT BE NAMED	Literal fields must not be named.
E	255	THIS SCREEN ITEM MUST NOT HAVE BASE SPECIFICATION	The BASE clause is allowed only at the 01 level.
E	256	THIS SCREEN ITEM MUST NOT HAVE 'OVERLAY' SPECIFICATION	The OVERLAY clause is allowed only at the 01 level.
E	257	THIS SCREEN ITEM MUST NOT HAVE SIZE SPECIFIED	The SIZE clause is allowed only at the 01 level or for overlay area items.
E	258	THIS SCREEN ITEM MUST NOT HAVE 'AREA' SPECIFICATION	AREA can only be specified for overlay areas. This item either has conflicting clauses or subordinate items (is a group).
E	259	THIS SCREEN ITEM MUST NOT HAVE LOCATION ('AT') SPECIFIED	The screen item must not have an associated screen location. The AT clause is allowed only for screen groups and fields.
E	260	THIS SCREEN ITEM MUST NOT HAVE LOCATION ('REDEFINES') SPECIFIED	The screen item must not redefine another screen item. The REDEFINES clause is allowed only for elementary screen fields.
E	261	THIS SCREEN ITEM MUST NOT HAVE FROM (OR USING) DATA ITEM	Only output fields and input-output fields can have FROM or USING clauses.
E	262	THIS SCREEN ITEM MUST NOT HAVE TO (OR USING) DATA ITEM	Only input fields and input-output fields can have TO or USING clauses.
E	263	THIS SCREEN ITEM MUST NOT HAVE SHADOW DATA ITEM SPECIFIED	SHADOWED clauses are allowed only for input, output, or input-output fields.

Table A-3. SCREEN COBOL Compiler Error Messages (Continued)

Type	No.	Message	Meaning
E	264	THIS SCREEN ITEM MUST NOT HAVE PICTURE SPECIFICATION	PICTURE clauses are allowed only for input, output, or input-output fields.
E	265	THIS SCREEN ITEM MUST NOT HAVE INITIAL VALUE	VALUE clauses are allowed only for input, output, input-output, or literal fields.
E	266	THIS SCREEN ITEM MUST NOT HAVE FILL CHARACTER SPECIFIED	FILL clauses are allowed only for input, output, or input-output fields.
E	267	THIS SCREEN ITEM MUST NOT HAVE OCCURS SPECIFICATION	OCCURS clauses are allowed only for input, output, or input-output fields.
E	268	THIS SCREEN ITEM MUST NOT HAVE ACCEPTABLE VALUE(S) ('MUST') SPECIFIED	MUST clauses are allowed only for input or input-output fields.
E	269	THIS SCREEN ITEM MUST NOT HAVE ACCEPTABLE LENGTH(S) SPECIFIED	LENGTH clauses are allowed only for input or input-output fields.
E	270	THIS SCREEN ITEM MUST NOT HAVE UPSHIFT SPECIFICATION	UPSHIFT clauses are allowed only for input, output, or input-output fields.
E	271	THIS SCREEN ITEM MUST NOT HAVE FULL ACTION ('WHEN FULL') SPECIFIED	WHEN FULL clauses are allowed only for input or input-output fields.
E	272	THIS SCREEN ITEM MUST NOT HAVE USER CONVERSION NUMBER SPECIFIED	USER CONVERSION clauses are allowed only for input, output, or input-output fields.
E	275	THIS SCREEN ITEM MUST NOT HAVE PROMPT FIELD SPECIFIED	The screen item must not have a PROMPT clause specified.

Table A-3. SCREEN COBOL Compiler Error Messages (Continued)

Type	No.	Message	Meaning
E	276	THIS SCREEN ITEM MUST NOT HAVE FIELD-SEPARATOR CHARACTER SPECIFIED	The screen item must not have a FIELD-SEPARATOR clause specified. This clause can be specified only for an 01 screen level item.
E	277	THIS SCREEN ITEM MUST NOT HAVE GROUP-SEPARATOR CHARACTER SPECIFIED	The screen item must not have a GROUP-SEPARATOR clause specified. This clause can be specified only for an 01 screen level item.
E	278	THIS SCREEN ITEM MUST NOT HAVE ABORT-INPUT CHARACTERS SPECIFIED	The screen item must not have an ABORT-INPUT clause specified. This clause can be specified only for an 01 screen level item.
E	279	THIS SCREEN ITEM MUST NOT HAVE END-OF-INPUT CHARACTERS SPECIFIED	The screen item must not have an END-OF-INPUT clause specified. This clause can be specified only for an 01 screen level item.
E	280	THIS SCREEN ITEM MUST NOT HAVE RESTART-INPUT CHARACTERS SPECIFIED	The screen item must not have a RESTART-INPUT clause specified. This clause can be specified only for an 01 screen level item.
E	285	THIS SCREEN ITEM MUST NOT HAVE ADVISORY SPECIFICATION	ADVISORY clauses are allowed only for output or input-output fields.
E	286	THIS SCREEN ITEM HAS DUPLICATE NAMING	Duplicate clauses are not allowed.
E	287	THIS SCREEN ITEM HAS DUPLICATE BASE SPECIFICATION	Duplicate clauses are not allowed.
E	288	THIS SCREEN ITEM HAS DUPLICATE 'OVERLAY' SPECIFICATION	Duplicate clauses are not allowed.
E	289	THIS SCREEN ITEM HAS DUPLICATE SIZE SPECIFIED	Duplicate clauses are not allowed.

Table A-3. SCREEN COBOL Compiler Error Messages (Continued)

Type	No.	Message	Meaning
E	290	THIS SCREEN ITEM HAS DUPLICATE 'AREA' SPECIFICATION	Duplicate clauses are not allowed.
E	291	THIS SCREEN ITEM HAS DUPLICATE LOCATION ('AT') SPECIFIED	Duplicate clauses are not allowed.
E	292	THIS SCREEN ITEM HAS DUPLICATE LOCATION ('REDEFINES') SPECIFIED	Duplicate clauses are not allowed.
E	293	THIS SCREEN ITEM HAS DUPLICATE FROM (OR USING) DATA ITEM	Duplicate clauses are not allowed.
E	294	THIS SCREEN ITEM HAS DUPLICATE TO (OR USING) DATA ITEM	Duplicate clauses are not allowed.
E	295	THIS SCREEN ITEM HAS DUPLICATE SHADOW DATA ITEM SPECIFIED	Duplicate clauses are not allowed.
E	296	THIS SCREEN ITEM HAS DUPLICATE PICTURE SPECIFICATION	Duplicate clauses are not allowed.
E	297	THIS SCREEN ITEM HAS DUPLICATE INITIAL VALUE	Duplicate clauses are not allowed.
E	298	THIS SCREEN ITEM HAS DUPLICATE FILL SPECIFIED	Duplicate clauses are not allowed.
E	299	THIS SCREEN ITEM HAS DUPLICATE OCCURS SPECIFICATION	Duplicate clauses are not allowed.
E	300	THIS SCREEN ITEM HAS DUPLICATE ACCEPTABLE VALUE(S) ('MUST') SPECIFIED	Duplicate clauses are not allowed.

Table A-3. SCREEN COBOL Compiler Error Messages (Continued)

Type	No.	Message	Meaning
E	301	THIS SCREEN ITEM HAS DUPLICATE ACCEPTABLE LENGTH(S) SPECIFIED	Duplicate clauses are not allowed.
E	302	THIS SCREEN ITEM HAS DUPLICATE UPSHIFT SPECIFICATION	Duplicate clauses are not allowed.
E	303	THIS SCREEN ITEM HAS DUPLICATE FULL ACTION ('WHEN FULL') SPECIFIED	Duplicate clauses are not allowed.
E	304	THIS SCREEN ITEM HAS DUPLICATE USER CONVERSION NUMBER SPECIFIED	Duplicate clauses are not allowed.
E	307	THIS SCREEN ITEM HAS DUPLICATE PROMPT CLAUSE SPECIFIED	Duplicate clauses are not allowed.
E	308	THIS SCREEN ITEM HAS DUPLICATE FIELD-SEPARATOR CHARACTER SPECIFIED	Duplicate characters are not allowed in multiple input character clauses.
E	309	THIS SCREEN ITEM HAS DUPLICATE GROUP-SEPARATOR CHARACTER SPECIFIED	Duplicate characters are not allowed in multiple input character clauses.
E	310	THIS SCREEN ITEM HAS DUPLICATE ABORT-INPUT CHARACTER SPECIFIED	Duplicate characters are not allowed in multiple input character clauses.
E	311	THIS SCREEN ITEM HAS DUPLICATE END-OF-INPUT CHARACTER SPECIFIED	Duplicate characters are not allowed in multiple input character clauses.
E	312	THIS SCREEN ITEM HAS DUPLICATE RESTART-INPUT CHARACTER SPECIFIED	Duplicate characters are not allowed in multiple input character clauses.

Table A-3. SCREEN COBOL Compiler Error Messages (Continued)

Type	No.	Message	Meaning
E	317	THIS SCREEN ITEM HAS DUPLICATE ADVISORY SPECIFICATION	Duplicate clauses are not allowed.
E	318	INPUT SCREEN ITEMS (TO OR USING) MAY NOT BE PROTECTED	Input and input-output fields must not be protected; if they were, data entry would be impossible.
E	319	REDEFINED SCREEN ITEM HAS DIFFERENT LOCATION	A redefined field must have the same location as the field it redefines.
E	320	REDEFINED SCREEN ITEM HAS DIFFERENT LENGTH	A redefined field must have the same length as the field it redefines.
E	321	REDEFINED SCREEN ITEM HAS DIFFERENT DISPLAY ATTRIBUTE	A redefined field must have the same display attribute as the field it redefines.
E	322	REDEFINED SCREEN ITEM HAS DIFFERENT FULL ACTION	A redefined field must have the same full action (WHEN FULL) as the field it redefines.
E	323	REDEFINED SCREEN ITEM HAS DIFFERENT OCCURS SPECIFICATION	A redefined field must have the same occurs specification as the field it redefines.
E	324	DUPLICATE SPECIFICATION FOR DISPLAY ATTRIBUTE	A given type of display attribute has been declared more than once.
E	325	INITIAL VALUE MUST BE QUOTED STRING	Only string literals are allowed for initial values of screen items.
E	326	TOO MANY SEPARATIONS OR OFFSETS IN COLUMN SPACING LIST	The column spacing list must contain fewer entries than there are column occurrences.
E	327	UNKNOWN TERMINAL TYPE	Terminal type must be IBM-3270, T16-6510, T16-6520, or T16-6530.
E	328	NO TERMINAL TYPE SPECIFIED	A terminal type clause is required.
E	329	FUNCTION KEY NOT ALLOWED FOR THIS TERMINAL TYPE	The function key mentioned is not available for this terminal type.

Table A-3. SCREEN COBOL Compiler Error Messages (Continued)

Type	No.	Message	Meaning
E	330	DISPLAY ATTRIBUTE NOT ALLOWED FOR THIS TERMINAL TYPE	The display attribute mentioned is not available for this terminal type.
E	331	FROM (USING) DATA ITEM HAS DIFFERENT TYPE (NUMBER VS. STRING)	Numeric screen items must be associated with numeric data items and nonnumeric screen items must be associated with nonnumeric data items.
E	332	FROM (USING) DATA ITEM HAS INSUFFICIENT NUMBER OF OCCURRENCES	The screen item has more occurrences than the data item.
E	333	FROM (USING) DATA ITEM HAS INCOMPATIBLE SCALE	The scale specified for the TO or USING data item is not compatible with that specified by the screen item PICTURE. The scale should be adjusted for compatible editing of data.
E	334	TO (USING) DATA ITEM HAS DIFFERENT TYPE (NUMBER VS. STRING)	Numeric screen items must be associated with numeric data items and nonnumeric screen items must be associated with nonnumeric data items.
E	335	TO (USING) DATA ITEM HAS INSUFFICIENT NUMBER OF OCCURRENCES	The screen item has more occurrences than the data item.
E	336	TO (USING) DATA ITEM HAS INCOMPATIBLE SCALE	The scale specified for the FROM or USING data item is not compatible with that specified by the screen item PICTURE. The scale should be adjusted for compatible editing of data.
E	337	VALUE STRING LONGER THAN PICTURE	The value string must not be longer than the screen item.
E	338	OVERLAY AREA TOO LARGE	The overlay area is larger than the base screen.
E	339	OVERLAY SCREENS MAY NOT CONTAIN OVERLAY AREAS	Recursive definition of overlay areas is not allowed.
E	340	OVERLAY AREAS MUST BE FULL WIDTH FOR THIS TERMINAL TYPE	Overlay areas must be as wide as the base screen on T16-6510 terminals.
E	341	SCREEN TOO LARGE FOR TERMINAL TYPE	Screen size exceeds the largest supported size for this terminal type.

Table A-3. SCREEN COBOL Compiler Error Messages (Continued)

Type	No.	Message	Meaning
E	342	ERROR ENHANCEMENT MAY NOT SPECIFY PROTECTION ATTRIBUTE	An ERROR-ENHANCEMENT clause must not specify PROTECTED attribute. If a field in error is protected, correction of the error would not be possible.
E	343	SHADOWED DATA ITEM HAS INSUFFICIENT NUMBER OF OCCURRENCES	If a shadowed field contains an OCCURS clause, the shadowed data item must have the same number of occurrences as the field.
E	344	SCREEN ITEM TOO LONG	The maximum field length of 255 characters has been exceeded.
E	345	UNKNOWN CHARACTER SET	The character-set type specified in the OBJECT-COMPUTER paragraph of the Environment Division is not valid.
W	346	CHARACTER SET NOT VALID FOR THIS TERMINAL TYPE	The character set specified is not valid for the terminal type. The character set specification is ignored.
E	347	PROMPT SCREEN ITEM MUST BE A FIELD IN SAME SCREEN	(1) The screen item named in the PROMPT clause and the definition of the screen field must be in the same screen. (2) The screen item must be a field. (3) The screen item cannot be an overlay, a group, or a filler item.
E	348	PROMPT SCREEN ITEM MUST NOT BE THE CURRENT ITEM	The screen item named in the PROMPT clause cannot refer to the screen item containing the PROMPT clause. A screen field cannot be prompted by itself.
E	349	PROMPT SCREEN ITEM MUST HAVE A FROM (OR USING) DATA ITEM	If the screen item named in the PROMPT clause has an associated working storage data item, the screen item must have a FROM or USING clause. A TO clause generates this error.
E	350	DUPLICATE INPUT CONTROL CHARACTERS DEFINED	The same character cannot be defined for more than one input control character within each screen.
E	351	OPERAND MUST BE A SCREEN ITEM	The operand must be an item defined in the Screen Section.
E	352	OPERAND MUST BE A DATA ITEM	The operand must be an item defined in the Working-Storage Section.
E	353	OPERAND MUST BE A MNEMONIC-NAME	The operand must be a mnemonic name specified in the SPECIAL-NAMES paragraph.

Table A-3. SCREEN COBOL Compiler Error Messages (Continued)

Type	No.	Message	Meaning
E	354	INVALID CONVERSATIONAL SEPARATOR	A field or group separator is defined incorrectly—a nonnumeric literal must be one alphanumeric character enclosed in quotation marks and a numeric literal must be in the range 0 through 255.
E	355	TOO MANY COLUMN OCCURRENCES SPECIFIED FOR SCREEN FIELD	An OCCURS clause includes a column number greater than the number of columns in the size of the screen. For example, if a screen is defined as SIZE 20, 80, an OCCURS IN 82 COLUMNS will generate this message.
E	356	TOO MANY LINE OCCURRENCES SPECIFIED FOR SCREEN FIELD	An OCCURS clause includes a line number greater than the number of lines in the size of the screen. For example, if a screen is defined as SIZE 20, 80, an OCCURS IN 24 LINES will generate this message.
E	357	ILLEGAL SENDING OR RECEIVING ITEM IN MOVE STATEMENT	(1) A numeric data item cannot be moved into an alphabetic data item. (2) A noninteger numeric data item cannot be moved into an alphanumeric data item. (3) An alphabetic data item cannot be moved into a numeric data item.
E	358	UNABLE TO OPEN SCREEN COBOL LIBRARY FILE	The specified SCREEN COBOL library cannot be accessed. The library either does not exist or could not be shared at compile time.
E	359	UNABLE TO LIST LOAD MAP	The object file cannot be opened to list the internal procedure load map.
E	360	UNDEFINED DATA NAME	The referenced data item is not described in the Environment or Data Division.
E	361	ONLY A MNEMONIC NAME IS ALLOWED IN THIS CONTEXT	A system name (mnemonic-name) is required in this context.
E	362	NO CORRESPONDING DATA NAMES	No correspondence was found between the specified groups.
E	363	UNDEFINED OR AMBIGUOUS PROCEDURE ACCESS	Either a procedure name referenced in a PERFORM or GO TO statement was not encountered in the source text, or the name was not sufficiently qualified to avoid ambiguity.
E	364	INDEPENDENT SEGMENTS NOT SUPPORTED	SCREEN COBOL does not support independent segments.
E	365	ILLEGAL DATA ITEM IN IF STATEMENT	An operand in a conditional statement is an illegally defined data item; the operand is defined as a numeric edited data item.

Table A-3. SCREEN COBOL Compiler Error Messages (Continued)

Type	No.	Message	Meaning
E	367	ONLY AN ALPHABET NAME IS ALLOWED IN THIS CONTEXT	The name specified must refer to an alphabetic name in this context.
E	368	EMPTY GO TO NOT LABELED	A GO TO statement of this form can only appear in a single statement paragraph, which by definition is labeled.
E	370	EXPECT ELEMENTARY NUMERIC DATA ITEM IN THIS CONTEXT	A numeric data item is required in this context.
E	371	EXPECT GROUP DATA ITEM IN THIS CONTEXT	Only a group data item is legal in this context.
E	372	INVALID TABLE SUBSCRIPT OR INDEX	The item is not a data item; indexes are not allowed.
E	373	TOO MANY OR TOO FEW PARAMETERS	The number of parameters specified in the USING phrase of a CALL statement does not agree with the number specified in the USING phrase of the Procedure Division header.
E	374	EXPECT A DATA WORD OR IDENTIFIER IN THIS CONTEXT	Only a data item can be used in a class condition.
E	375	CATEGORY MUST BE ALPHANUMERIC OR ALPHABETIC	The category of the data item must be alphanumeric or alphabetic in this context.
E	376	CATEGORY MUST BE ALPHANUMERIC OR NUMERIC	The category of the data item must be alphanumeric or numeric in this context.
E	381	MISSING PROGRAM	A called program unit was neither found in a SCREEN COBOL program library nor encountered in the source text.
E	385	EXPECT A TABLE SPECIFIER	The description of the data item must contain an OCCURS clause.
E	386	INCORRECT NUMBER OF SUBSCRIPTS OR INDICES	An incorrect number of subscripts, possibly zero, are used to access the data item. (Indices are not allowed.)
E	387	REFERENCE DATA ITEM TOO LARGE	A data item has a PICTURE clause greater than 2048.
E	389	AN UNEXPECTED ERROR OCCURRED WHILE BUILDING RUN UNIT	An irrecoverable I/O error occurred while building the object file. The compilation must be restarted.

Table A-3. SCREEN COBOL Compiler Error Messages (Continued)

Type	No.	Message	Meaning
E	391	ILLEGAL USE OF INDEX NAME OR INDEX DATA ITEM	An index name or index data item is not allowed.
E	395	INVALID VARYING ITEM	The VARYING identifier in the PERFORM statement must be described as a numeric elementary data item without any positions to the right of the assumed decimal point.
E	398	ILLEGAL COMPARISON BETWEEN DISPLAY AND COMPUTATIONAL DATA	A comparison between a numeric computational data item and a nonnumeric data item is illegal.
E	399	NON-INTEGER OR CONTAINS P'S	An integer data item containing no P symbols in its PICTURE clause is required in this context.
E	400	EXPECT ALPHANUMERIC DATA ITEM	The data item must have an explicit or implied alphanumeric category.
E	401	EXPECT DISPLAY USAGE	The data item must have explicit or implied DISPLAY usage.
E	403	EXPECT LEVEL 01 OR 77 DATA ITEM	Only level 01 and level 77 data items can be specified in the USING phrase of a CALL statement.
E	404	INVALID DISPLAY ITEM	An item is not defined in the Data Division.
E	415	INVALID OPERATOR OR OPERAND IN ARITHMETIC EXPRESSION	The expression contains operators other than + − / * or contains one or more nonnumeric operands.
E	416	INVALID OPERATOR OR OPERAND IN CONDITIONAL EXPRESSION	The expression contains an illegal operator or illegal identifier form.
E	418	ILLEGAL TABLE OCCURRENCE NUMBER (BOUNDS VIOLATION)	An illegal occurrence number was detected.
E	427	ONLY AN ELEMENTARY ALPHANUMERIC DATA ITEM IS ALLOWED	The referenced data item must be an elementary alphanumeric data item in this context.

Table A-3. SCREEN COBOL Compiler Error Messages (Continued)

Type	No.	Message	Meaning
E	429	ILLEGAL DATA ITEM IN ARITHMETIC STATEMENT	An operand in an arithmetic statement is an illegally defined data item; the operand is defined as a numeric edited data item.
E	430	DIVIDE BY ZERO	Division by a literal with a value of zero was detected in a DIVIDE or COMPUTE statement.
E	431	ONLY A NUMERIC LITERAL IS ALLOWED	Only numeric literals and data items can be used in the composition of an arithmetic expression.
E	445	ILLEGAL PERFORM INVOCATION	A statement generates an illegal implied or explicit transfer of control between mutually exclusive sections. The statement is compiled as if the requested action were legal.
W	445	ILLEGAL PERFORM INVOCATION	A statement generates an illegal implied or explicit transfer of control between mutually exclusive sections. The statement is compiled as if the requested action were legal. 1) A debug section performs a declarative procedure. 2) A non-debug declarative section performs a non-debug declarative procedure. 3) A non-declarative section performs a non-declarative procedure.
E	446	ILLEGAL GO TO INVOCATION	A statement generates an illegal implied or explicit transfer of control between mutually exclusive sections. The statement is compiled as if the requested action were legal. 1) A GO TO transfers between a declarative section and a non-declarative section. 2) A GO TO transfers between a debug section and a non-debug section.
W	446	ILLEGAL GO TO INVOCATION	A statement generates an illegal implied or explicit transfer of control between mutually exclusive sections. The statement is compiled as if the requested action were legal.
W	468	ADDRESSING RANGE EXCEEDED	More than 16000 bytes of working storage was declared. This exceeds the addressing range of a pseudo object program on the TCP.
E	469	SOURCE ITEM MAY NOT EXCEED 18 DIGITS	Numeric data is limited to 18 digits.
E	473	PROTECTION ATTRIBUTE MAY NOT BE CHANGED	The PROTECTED attribute must not be changed for the T16-6510.

Table A-3. SCREEN COBOL Compiler Error Messages (Continued)

Type	No.	Message	Meaning
E	474	OVERLAY SCREEN LARGER THAN OVERLAY AREA	The overlay screen is larger than the overlay area.
E	475	ACCEPT TERMINATION MNEMONIC NAME MUST BE FUNCTION KEY	Display attributes must not be used to terminate an ACCEPT statement.
E	476	SERVER CLASS NAME MUST BE ALPHA OR ALPHANUMERIC	PATHCOM does not accept numeric server class names.
E	477	TURN MNEMONIC NAME MUST BE DISPLAY ATTRIBUTE	Function keys are not allowed in TURN statements.
E	478	MUST BE SCREEN NAME	Groups, overlay areas, and fields are not allowed in this context.
E	479	MUST BE BASE SCREEN NAME	Overlay screens must not be used in this context.
E	480	MUST BE OVERLAY SCREEN NAME	Base screens must not be used in this context.
E	481	MUST BE SCREEN ITEM	Data items are not allowed in this context.
E	482	SCREEN ITEM TOO CLOSE TO START OF SCREEN	Required separation between screen elements is not present.
E	483	SCREEN ITEM TOO CLOSE TO END OF SCREEN	Required separation between screen elements is not present.
E	484	SCREEN ITEM SPANS NOT-FULLWIDTH OVERLAY AREA LINE	A screen item cannot span an overlay screen line when that overlay screen is narrower than the base screen.
E	485	SCREEN ITEM OVERLAPS OR IS TOO CLOSE TO NEXT ITEM	Required separation between screen elements is not present.
E	486	SCREEN ITEM OVERLAPS OR IS TOO CLOSE TO PREVIOUS ITEM	Required separation between screen elements is not present.

Table A-3. SCREEN COBOL Compiler Error Messages (Continued)

Type	No.	Message	Meaning
E	487	SCREEN ITEM WITHOUT OCCURS CLAUSE IS SUBSCRIPTED	Only a screen item with an OCCURS clause can be subscripted.
E	488	SCREEN ITEMS MAY HAVE ONE (1) SUBSCRIPT ONLY	A screen item with an OCCURS clause is considered to be a single table.
E	489	SCREEN ITEM SUBSCRIPT TOO LARGE	The subscript exceeds the count of screen items.
E	490	SCREEN ITEM SUBSCRIPT MUST BE INTEGER	All subscripts must be integers.
E	491	MUST BE OVERLAY AREA SCREEN NAME	The overlay-area following the AT clause in a DISPLAY OVERLAY statement is not defined as an overlay area.
E	492	'LENGTH MUST BE' VALUE MUST BE LESS THAN 256	The maximum value that can be specified in a LENGTH clause is 255.
E	493	INCORRECT NUMBER OF OPERANDS IN ARITHMETIC EXPRESSION	An arithmetic expression contains either too many or too few operands.
E	494	SCALE OF MUST BE VALUE EXCEEDS SCALE OF ASSOCIATED DATA ITEM	A numeric literal named in the MUST BE clause contains too many digits to the right of the decimal.
E	495	MUST BE VALUE TOO LARGE FOR ASSOCIATED DATA ITEM	A numeric literal named in the MUST BE clause exceeds the size of the data item PICTURE clause.
E	496	TYPE OF MUST BE VALUE IS INCOMPATIBLE WITH ASSOCIATED DATA ITEM	The type of literal named in the MUST BE clause does not match the associated data item. A numeric literal must be associated with a numeric data item and a nonnumeric literal must be associated with a nonnumeric data item.
W	497	QUALIFIED NAME TOO LONG — CROSSREF LINE WILL BE TRUNCATED	This is a warning. During compilation, SCOBOL builds fully qualified names to send to CROSSREF. A name exceeded the length of the buffer and will appear in the CROSSREF listing in truncated form. This might occur with multiple levels of qualification.

APPENDIX B

SCREEN COBOL SYNTAX SUMMARY

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

Page

```
[ SPECIAL-NAMES.                                                    4-4

    [ { mnemonic-name IS  { system-name                 } }   ,... ]  4-4
    [ {                   { ( { system-name} ,...)      } }        ]

  [ , CURRENCY [ SIGN ] IS literal-1 ]                              4-5

  [ , DECIMAL-POINT IS COMMA ] . ]                                  4-5

[ INPUT-OUTPUT SECTION.                                             4-7

  SCREEN-CONTROL.                                                   4-7

    ERROR-ENHANCEMENT [ IS ] mnemonic-name  [ IN  { FIRST } ]       4-7
                                            [     { ALL   } ]

    [ WITH [ NO ] AUDIBLE ALARM ] . ]
```

DATA DIVISION

```
  DATA DIVISION.

[ WORKING-STORAGE SECTION.                                          5-2

    data-description-entries ]

[ LINKAGE SECTION.                                                  5-2

    data-description-entries ]
```

DATA DESCRIPTION CLAUSES

```
level-number   { data-name-1 }                                      5-5
               { FILLER      }

    [ { JUST      }   RIGHT ]                                       5-6
    [ { JUSTIFIED }         ]

    [ OCCURS  { max [ TIMES ]                                      } ]  5-7
    [         { min TO max [ TIMES ] DEPENDING [ ON ] depend       } ]

    [ { PIC     }   [ IS ] character-string ]                       5-8
    [ { PICTURE }                           ]

   [ REDEFINES data-name-2 ]                                        5-10

      [ SIGN [ IS ] ]   LEADING    [ SEPARATE [ CHARACTER ] ]       5-13
                        TRAILING
```

Page

```
[{SYNC        }    [RIGHT]]                                          5-14
 {SYNCHRONIZED}    [LEFT ]]

[ [ USAGE [ IS ] ]  {COMP        }]                                   5-16
                    {COMPUTATIONAL}
                    {DISPLAY     }]

  [ VALUE [ IS ] literal ]                                           5-17

66 new-name RENAMES old-name    [{THROUGH} end-name] .               5-12
                                 {THRU   }

88 condition-name ,  {VALUE [ IS ]  }                              5-18
                     {VALUES [ ARE ]}

     {value-1    [{THROUGH} value-2]}    ,...
                  {THRU   }

[ SCREEN SECTION.                                                    5-3

    input-control-entries
    screen-description-entries ]
```

INPUT CONTROL CLAUSES

```
screen-name  {[ BASE ] [ SIZE clause ]}                              5-23
             {OVERLAY  SIZE clause    }                             5-25

          [ ABORT-INPUT [ IS ]  {"nonnumeric-literal"      }]        5-29
                                {numeric-literal          }
                                {  [ , numeric-literal ]  }
                                {OFF                      }

          [ END-OF-INPUT [ IS ]  {"nonnumeric-literal"     }]       5-30
                                 {numeric-literal         }
                                 {  [ , numeric-literal ] }
                                 {OFF                     }

          [ FIELD-SEPARATOR [ IS ]  {"nonnumeric-literal"} ]        5-31
                                    {numeric-literal      }
                                    {OFF                  }

          [ GROUP-SEPARATOR [ IS ]  {"nonnumeric-literal"} ]        5-32
                                    {numeric-literal      }
                                    {OFF                  }

          [ RESTART-INPUT [ IS ]  {"nonnumeric-literal"     }]      5-33
                                  {numeric-literal         }
                                  {  [ , numeric-literal ] }
                                  {OFF                     }
```

Page

SCREEN DESCRIPTION CLAUSES

```
level-num  {field-name}    {[ AT ] line-spec, column-spec}          5-26/5-34
           {FILLER    }    {REDEFINES field-name-2        }          5-43

  [ ADVISORY ]                                                      5-34

  [ FILL nonnumeric-literal ]                                       5-35

  [ LENGTH [ MUST BE ]  {literal-1  [{THROUGH} literal-2]} ,... ]   5-35
                        {           [{ THRU  }         ]}

  [ mnemonic-name ] ...                                             5-36

  [ MUST [ BE ]  {literal-1  [{THROUGH} literal-2]}     ,... ]      5-36
                 {           [{THRU    }         ]}

  [ OCCURS  {lines-phrase [ columns-phrase ]}                       5-37
            {columns-phrase [ lines-phrase ]}

      [ DEPENDING [ ON ] data-name-1 ]]                             5-38

  [ {PIC    }  [ IS ] character-string]                             5-39
  [ {PICTURE}                        ]

  [ PROMPT screen-field ]                                           5-41

  [ RECEIVE [ FROM ]  {ALTERNATE                }  ]                5-43
                      {ALTERNATE OR TERMINAL    }
                      {TERMINAL                 }
                      {TERMINAL OR ALTERNATE    }

  [ REDEFINES field-name-2 ]                                        5-44

  [ SHADOWED [ BY ] data-name-1 ]                                   5-44

     [ {TO   }  data-name-1 ]                                       5-46
     [ {FROM }              ]
     [ {USING}              ]

  [ UPSHIFT  [ INPUT        ] ]                                     5-47
             [ OUTPUT       ]
             [ I-O          ]
             [ INPUT-OUTPUT ]

  [ USER [ CONVERSION ] numeric-literal ]                           5-47

  [ VALUE nonnumeric-literal ]                                      5-47

  [ WHEN  {ABSENT}    {CLEAR}]                                      5-48
          {BLANK }    {SKIP }]

  [ [ WHEN ] FULL  {TAB }]                                          5-49
                   {LOCK}]
```

Page

SCREEN COBOL COMPILER DEFINED SPECIAL REGISTERS

```
01 DIAGNOSTIC-ALLOWED      PIC AAA.                                    5-55

01 LOGICAL-TERMINAL-NAME    PIC X(16).                                 5-55

01 NEW-CURSOR.                                                         5-56
   02 NEW-CURSOR-ROW      PIC 9999 COMP.
   02 NEW-CURSOR-COL      PIC 9999 COMP.

01 OLD-CURSOR.                                                         5-56
   02 OLD-CURSOR-ROW      PIC 9999 COMP.
   02 OLD-CURSOR-COL      PIC 9999 COMP.

01 REDISPLAY      PIC AAA.                                             5-57

01 RESTART-COUNTER     PIC 9999 COMP.                                  5-57

01 STOP-MODE      PIC 9999 COMP.                                       5-57

01 TELL-ALLOWED      PIC AAA.                                          5-58

01 TERMINAL-FILENAME      PIC X(24).                                   5-58

01 TERMINAL-PRINTER      PIC X(36).                                    5-58

01 TERMINATION-STATUS      PIC 9999 COMP.                              5-58

01 TERMINATION-SUBSTATUS      PIC 9999 COMP.                           5-59

01 TRANSACTION-ID     PIC X(8).                                        5-59
```

PROCEDURE DIVISION

```
PROCEDURE DIVISION [ USING data-name-1 [ , data-name-2 ] ... ] .       6-1

[ DECLARATIVES.                                                        6-3
{ [ section-name SECTION . ]
   [ paragraph-name . [ sentence ] ... ] ... } ...
{ paragraph-name . [ sentence ] ... } ...
END DECLARATIVES. ]

{ [ section-name SECTION . ]                                           6-3
   [ paragraph-name . [ sentence ] ... ] ... } ...
{ paragraph-name . [ sentence ] ... } ...
```

Page

PROCEDURE DIVISION STATEMENTS

Page

```
DIVIDE divisor INTO { dividend } ,...                                    6-30

DIVIDE divisor INTO dividend GIVING { quotient } ,...                    6-30

DIVIDE dividend BY divisor GIVING { quotient } ,...                      6-31

END-TRANSACTION                                                          6-32

EXIT .                                                                   6-32

EXIT PROGRAM [ WITH ERROR ] .                                            6-33

GO [ TO ] procedure-name                                                 6-33

GO TO { procedure-name } ,... DEPENDING [ ON ] depend                    6-34

IF condition {statement-1  }    [ELSE {statement-2  }]                   6-34
             {NEXT SENTENCE}          {NEXT SENTENCE}

MOVE data-name-1 TO { data-name-2 } ,...                                 6-36

MOVE {CORR          } group-1 TO group-2                                 6-37
     {CORRESPONDING}

MULTIPLY value BY { multiplier } ,...                                    6-41

MULTIPLY value BY multiplier GIVING { result } ,...                      6-41

PERFORM proc-1    [{THROUGH} proc-2]                                     6-42
                  {THRU   }

PERFORM proc-1    [{THROUGH} proc-2]  count TIMES                        6-43
                  {THRU   }

PERFORM proc-1    [{THROUGH} proc-2]  UNTIL condition                    6-44
                  {THRU   }

PERFORM proc-1    [{THROUGH} proc-2]                                     6-45
                  {THRU   }

  VARYING vary-1 FROM base-1 BY step-1 UNTIL condition-1

  [ AFTER vary-2 FROM base-2 BY step-2 UNTIL condition-2 ] ...

PERFORM ONE [ OF ] {proc-1    [{THROUGH} proc-2]}    ,...                6-46
                   {          {THRU   }       }

  DEPENDING [ ON ] identifier

PRINT SCREEN [ ON ERROR imperative-statement ]                           6-46

RECONNECT MODEM                                                          6-49
```

Page

```
RESET  [TEMP     ]    [ATTR]  { screen-identifier } ,...                    6-50
       [TEMPORARY]    [DATA]

   [DEPENDING [ ON ] identifier]
   [SHADOWED                   ]

RESTART-TRANSACTION                                                         6-51

SCROLL  {UP  }  overlay-area-name                                           6-52
        {DOWN}

SEND [ identifier-1] ,... TO server-class-name                              6-52

  [ UNDER PATHWAY pathmon-name ]

  [ AT SYSTEM system-name ]

  REPLY { CODE { reply-code-value } ...

  YIELDS { identifier-2 } ... } ...

  [ ON ERROR imperative-statement ]

SET NEW-CURSOR AT { screen-identifier } ,...                                 6-59

   [DEPENDING [ ON ] identifier]
   [SHADOWED                   ]

STOP RUN                                                                    6-60

SUBTRACT { sub-1 } ,... FROM { sub-2 } ,...                                 6-60

SUBTRACT { sub-1 } ,... FROM sub-2 GIVING { result } ,...                   6-61

SUBTRACT  {CORR          }  group-1 FROM group-2                            6-61
          {CORRESPONDING}

TURN  [TEMP     ]  {mnemonic-name                                    }  IN   6-63
      [TEMPORARY]  {RECEIVE FROM  {ALTERNATE                         }
                                  {ALTERNATE OR TERMINAL             }
                                  {TERMINAL                          }
                                  {TERMINAL OR ALTERNATE             }

 { screen-identifier } ,...  [DEPENDING [ ON ] identifier]
                             [SHADOWED                   ]

USE [ FOR SCREEN ] RECOVERY [ ON { base-screen-name-n } ,... ] .            6-64
```

Page

COMPILER CONTROL COMMANDS

APPENDIX C

SCREEN COBOL RESERVED WORDS

ABORT
ABORT-INPUT
ABORT-TRANSACTION
ABSENT
ACCEPT
ACCESS
ADD
ADVANCING
ADVISORY
ALARM
AFTER
ALL
ALPHABETIC
ALSO
ALTER
ALTERNATE
AND
APPROXIMATE
ARE
AREA
AREAS
ASCENDING
ASSIGN
AT
ATTR
AUDIBLE
AUTHOR

BASE
BE
BEFORE
BEGIN-TRANSACTION
BLANK
BLOCK
BOTTOM
BY

CD
CALL
CANCEL
CF
CH
CHARACTER
CHARACTERS
CHARACTER-SET
CHECKPOINT
CLEAR
CLOCK-UNITS
CLOSE
COBOL
CODE
CODE-SET
COLLATING
COLUMN
COLUMNS
COMMA
COMMUNICATION
COMP
COMPUTATIONAL
COMPUTE
CONFIGURATION
CONTAINS
CONTROL
CONTROLS
CONVERSATIONAL
CONVERSION
COPY
CORR
CORRESPONDING
COUNT
CROSSREF
CURRENCY

DATA
DATE
DATE-COMPILED
DATE-WRITTEN
DAY
DE
DEBUG-CONTENTS
DEBUG-ITEM
DEBUG-LINE
DEBUG-NAME
DEBUG-SUB-1
DEBUG-SUB-2
DEBUG-SUB-3
DEBUGGING
DECIMAL-POINT
DECLARATIVES
DELAY
DELETE
DELIMITED
DELIMITER
DEPENDING
DESCENDING
DESTINATION
DETAIL
DIAGNOSTIC-ALLOWED
DISABLE
DISPLAY
DIVIDE
DIVISION
DOWN
DUPLICATES
DYNAMIC

EGI
ELSE

EMI
ENABLE
END
END-OF-INPUT
END-OF-PAGE
END-TRANSACTION
ENTER
ENVIRONMENT
EOP
EQUAL
ERROR
ERROR-ENHANCEMENT
ESCAPE
ESI
EVERY
EXCEPTION
EXCLUSIVE
EXIT
EXTEND

FD
FIELD-SEPARATOR
FILE
FILE-CONTROL
FILL
FILLER
FINAL
FIRST
FIXED-LENGTH
FOOTING
FOR
FROM
FULL

GENERATE
GENERIC
GIVING
GO
GREATER
GROUP
GROUP-SEPARATOR

HEADING
HIGH-VALUE
HIGH-VALUES

I-O
I-O-CONTROL
I-O-ERROR
IDENTIFICATION
IF
IN
INDEX
INDEXED
INDICATE
INITIAL
INITIATE
INPUT
INPUT-OUTPUT
INSPECT
INSTALLATION
INTO
INVALID
IS

JUST
JUSTIFIED

KEY

LABEL
LAST
LEADING
LEFT
LENGTH
LESS
LIKE
LIMIT
LIMITS
LINAGE
LINAGE-COUNTER
LINE
LINE-COUNTER
LINES
LINKAGE
LOCK
LOCKFILE
LOGICAL-TERMINAL-NAME
LOW-VALUE
LOW-VALUES

MEMORY
MERGE
MESSAGE
MODE
MODEM
MODULES
MOVE
MULTIPLE
MULTIPLY
MUST

NATIVE
NEGATIVE
NEW-CURSOR
NEW-CURSOR-COL
NEW-CURSOR-ROW
NEXT
NO
NOSHADOW
NOT
NUMBER
NUMERIC
NUMERIC-SHIFT

OBJECT-COMPUTER
OCCURS
OF
OFF
OFFSET
OLD-CURSOR
OLD-CURSOR-COL
OLD-CURSOR-ROW
OMITTED
ON
ONE
OPEN
OPTIONAL
OR
ORGANIZATION
OUTPUT
OVERFLOW
OVERLAY

PAGE
PAGE-COUNTER
PATHWAY
PERFORM
PF
PH
PIC
PICTURE
PLUS
POINTER
POSITION
POSITIVE
PRINT
PRINTING
PROCEDURE
PROCEDURES
PROCEED
PROGRAM
PROGRAM-ID
PROGRAM-STATUS
PROGRAM-STATUS-1
PROGRAM-STATUS-2
PROMPT
PROTECT

QUEUE
QUOTE
QUOTES

RANDOM
RD
READ
RECEIVE
RECEIVE-CONTROL
RECONNECT
RECORD
RECORDS
RECOVERY
REDEFINES
REDISPLAY
REEL
REFERENCES
RELATIVE
RELEASE
REMAINDER
REMOVAL
RENAMES
REPLACING
REPLY
REPORT
REPORTING
REPORTS
RERUN
RESERVE
RESET
RESTART-COUNTER
RESTART-INPUT
RESTART-TRANSACTION
RETURN
REVERSED
REWIND
REWRITE
RF
RH
RIGHT
ROUNDED
RUN

SAME
SCREEN
SCREEN-CONTROL
SCROLL
SD
SEARCH
SECTION
SECURITY
SEGMENT
SEGMENT-LIMIT
SELECT
SEND
SENTENCE
SEPARATE
SEQUENCE
SEQUENTIAL
SET
SHADOWED
SHARED
SIGN
SKIP
SKIPPING
SORT
SORT-MERGE
SOURCE
SOURCE-COMPUTER
SPACE
SPACES
SPECIAL-NAMES
STANDARD
STANDARD-1
START
STARTBACKUP
STATUS
STOP
STOP-MODE
STRING
SUB-QUEUE-1
SUB-QUEUE-2
SUB-QUEUE-3
SUBTRACT
SUM
SUPPRESS
SYMBOLIC
SYNC
SYNCDEPTH
SYNCHRONIZED
SYSTEM

TAB
TABLE
TAL
TALLYING
TAPE
TELL-ALLOWED
TEMP
TEMPORARY
TERMINAL
TERMINAL-FILENAME
TERMINAL-PRINTER
TERMINATE
TERMINATION-STATUS
TERMINATION-SUBSTATUS
TEXT
THAN
THROUGH
THRU
TIME
TIMEOUT
TIMES
TO
TOP
TRAILING
TRANSACTION-ID
TRANSPARENT
TURN

UNDER
UNIT
UNLOCK
UNLOCKFILE
UNLOCKRECORD
UNSTRING
UNTIL
UP
UPON
UPSHIFT
USAGE
USE
USER
USING

VALUE
VALUES
VARYING

WHEN
WITH
WORDS
WORKING-STORAGE
WRITE

YIELDS

ZERO
ZEROES
ZEROS

APPENDIX D

USER CONVERSION PROCEDURES

User-defined checking and conversion can be implemented by writing one or more procedures in the Tandem Transaction Language (TAL). Use the BINDER development tool to store the procedures in the Tandem-supplied TCP object file. (The UPDATE program provides this operation for software versions prior to GUARDIAN E05 for the NonStop System and GUARDIAN A04 for the NonStop II System.)

The following are the basic commands to create a user TCP with BINDER:

```
BIND
ADD * FROM $SYSTEM.SYSTEM.PATHTCP
REPLACE * FROM user-conversion-object
BUILD user-tcp-name
EXIT
```

An example of the above commands:

```
BIND
ADD * FROM $SYSTEM.SYSTEM.PATHTCP
REPLACE * FROM $uvol.usubvol.urobj
BUILD $uvol.usubvol.urtcp
EXIT
```

For information about using BINDER to manage object files, see the *BINDER User's Manual.*

If you wish to continue using UPDATE to store conversion procedures, the symbol table and BINDER table must be removed from the PATHTCP object file. The BINDER command, STRIP, will delete the two tables from the TCP code.

Four procedures are available: two for input conversion and checking, and two for output conversion. These procedures exist in the TCP and are called only if the USER CONVERSION screen field characteristic clause is declared for the field. Any or all of the procedures can be replaced.

INPUT PROCEDURES

Two procedures can be invoked during input. One procedure is for input of numeric data items, and the other is for nonnumeric items. When the USER CONVERSION clause is declared for the field, the appropriate procedure is called as follows:

before value checks are applied

after the input has been stripped of fill characters

after standard conversion is attempted

The procedure is called even if an error occurs during the standard conversion attempt; if a length error occurs, however, the procedure is not called.

Most of the parameters of the two procedures are the same; they differ only for the internal data item. Declarations for the input procedures are shown in Figures D-1 and D-2.

```
PROC USER^NUMERIC^INPUT^CONVERSION( USERCODE, ERROR, INPUT,
          INPUT^LEN, INTERNAL, INTERNAL^SCALE );
INT      USERCODE;
INT     .ERROR;
STRING  .INPUT;
INT      INPUT^LEN;
FIXED   .INTERNAL;
INT      INTERNAL^SCALE;
```

Figure D-1. Input Procedure Declaration for Numeric Data Items

```
PROC USER^ALPHA^INPUT^CONVERSION( USERCODE, ERROR, INPUT,
          INPUT^LEN, INTERNAL, INTERNAL^LEN );
INT      USERCODE;
INT     .ERROR;
STRING  .INPUT;
INT      INPUT^LEN;
STRING  .INTERNAL;
INT      INTERNAL^LEN;
```

Figure D-2. Input Procedure Declaration for Nonnumeric Data Items

The *USERCODE* parameter is the value given in the USER CONVERSION field characteristic clause. This parameter can be used to select a particular type of conversion.

The *ERROR* parameter is both an input and an output parameter. When the procedure is called, the parameter contains either zero (indicating no error) or the number of a conversion error detected during the attempted standard conversion. Refer to Table A-1 of Appendix A for a listing of error codes.

The value of the *ERROR* parameter after the call determines whether or not an error for the field will be reported back to the terminal. If the value is nonzero, that value is used to select the error message to be displayed. Processing depends on the purpose of the procedure as follows:

- If the procedure simply performs additional checking on the input, the routine should return immediately if ERROR is nonzero. If ERROR is zero, the routine should proceed with its own checking and set ERROR according to the results.

- If the procedure performs conversion, the routine can generally ignore the value of ERROR as it is passed and set the parameter according to the results of the processing.

The *INPUT* parameter is the string of characters input from the terminal. Nonnumeric input is stripped of fill characters from the right; numeric input is stripped of fill characters from both the right and the left.

The *INPUT*^LEN parameter gives the number of bytes in the input string after the string is stripped of fill characters. The byte before and the byte after the input string will be set to null values.

The *INTERNAL* parameter contains the result of the standard conversion (if no error occurred), and should contain the result of the user conversion (unless ERROR is nonzero upon return).

- For the numeric procedure, *INTERNAL* is a FIXED parameter; if necessary, this value is later converted to the final data type by the TCP. The *INTERNAL*^*SCALE* parameter gives the scale that *INTERNAL* should have.

- For the nonnumeric procedure, *INTERNAL* is a string parameter. *INTERNAL*^*LEN* gives the exact number of bytes the result should occupy.

OUTPUT PROCEDURES

Two procedures can be invoked during output. One procedure is for output of numeric data items, and the other is for nonnumeric items. When the USER CONVERSION clause is declared for the field, the appropriate procedure is called after standard conversion has completed.

Most of the parameters of the two procedures are the same; they differ only for the internal data item. Declarations for the output procedures are shown in Figures D-3 and D-4.

```
PROC USER^NUMERIC^OUTPUT^CONVERSION( USERCODE, OUTPUT,
           OUTPUT^LEN, MAX^OUTPUT^LEN,
           INTERNAL, INTERNAL^SCALE );
INT      USERCODE;
STRING  .OUTPUT;
INT     .OUTPUT^LEN;
INT      MAX^OUTPUT^LEN;
FIXED   .INTERNAL;
INT      INTERNAL^SCALE;
```

Figure D-3. Output Procedure Declaration for Numeric Data Items

```
PROC USER^ALPHA^OUTPUT^CONVERSION( USERCODE, OUTPUT,
          OUTPUT^LEN, MAX^OUTPUT^LEN,
          INTERNAL, INTERNAL^LEN );
INT      USERCODE;
STRING .OUTPUT;
INT     .OUTPUT^LEN;
INT      MAX^OUTPUT^LEN;
STRING .INTERNAL;
INT      INTERNAL^LEN;
```

Figure D-4. Output Procedure Declaration for Nonnumeric Data Items

The *USERCODE* parameter is the value given in the USER CONVERSION field characteristic clause. This parameter can be used to select a particular type of conversion.

The *OUTPUT* parameter indicates where the string of characters for output to the terminal is to be placed. When the procedure is called, the location designated by this parameter will contain the result of the standard conversion.

The *OUTPUT^LEN* parameter contains the length of the output string. If the procedure changes the output string, the procedure should set *OUTPUT^LEN* to the associated length; in no case should *OUTPUT^LEN* be greater than *MAX^OUTPUT^LEN*. If *OUTPUT^LEN* is less than the field length, the fill character is used to pad the field.

The *INTERNAL* parameter contains the data to be converted.

- For the numeric procedure, *INTERNAL* is a FIXED parameter. The *INTERNAL^SCALE* parameter contains the number of decimal places.
- For the nonnumeric procedure, *INTERNAL* is a string. The *INTERNAL^LEN* parameter contains the number of bytes in the string.

3270 KEY MAPPING

A user-replaceable procedure called USER^3270^KEY^MAPPING is provided to support program attention keys PA4 through PA10. These keys are used on terminals analogous to the IBM-3270 terminal. Declarations for the key mapping procedure are shown in Figure D-5.

```
PROC USER^3270^KEY^MAPPING ( AID, KEYNUM );

  INT  AID;       ! 3270 AID BYTE     — CONTAINS THE PATHWAY KEY NUMBER
  INT .KEYNUM;    ! ON THE CALL       — ASSOCIATED WITH THE AID BYTE
                                        ACCORDING TO THE FOLLOWING TABLE
                                        OR -1 IF KEY IS UNDEFINED.
                  ! ON THE RETURN     — PATHWAY KEY NUMBER OR -1 IF
                                        UNDEFINED.
!
! PATHWAY 3270 KEY TO INTERNAL KEY NUMBER MAPPING.
!
!
! SCREEN COBOL              3270 AID
! SPECIAL-NAME                BYTE          PATHWAY KEY NUMBER
!
! ENTER                      %047           0
! PA1                        %045           1
! PA2                        %076           2
! PA3                        %054           3
! CLEAR                      %137           4
! PF1                        %061           5
! PF2                        %062           6
! PF3                        %063           7
! PF4                        %064           8
! PF5                        %065           9
! PF6                        %066          10
! PF7                        %067          11
! PF8                        %070          12
! PF9                        %071          13
! PF10                       %072          14
! PF11                       %043          15
! PF12                       %100          16
! PF13                       %101          17
! PF14                       %102          18
! PF15                       %103          19
! PF16                       %104          20
! PF17                       %105          21
! PF18                       %106          22
! PF19                       %107          23
! PF20                       %110          24
! PF21                       %111          25
! PF22                       %133          26
! PF23                       %056          27
! PF24                       %074          28
! undefined                  %060          29 (Test Request)
! undefined                  %127          30 (Op ID Card Reader)
! PA5                      undefined       32
```

Figure D-5. Procedure Declaration for 3270 Key Mapping

```
! PA6          undefined      33
! PA7          undefined      34
! PA8          undefined      35
! PA9          undefined      36
! PA10         undefined      37
! undefined    %075           -1 (Selector Pen Attn)
! undefined    %055           -1 (no aid--Display)
! undefined    %131           -1 (no aid--Printer)
! other        —              -1
```

Figure D-5. Procedure Declaration for 3270 Key Mapping (continued)

APPENDIX E

PATHWAY PROGRAMMING FOR TMF

The purpose of this appendix is to provide information related to PATHWAY and TMF interaction. This information is for SCREEN COBOL programmers who are designing PATHWAY application programs and for system managers who are controlling PATHWAY applications in a system that uses TMF.

The general environment for PATHWAY applications using TMF is a requester/server environment where the requester (TCP) accepts input from a terminal operator and transforms the input into a request for data base services from the servers. The servers, in turn, satisfy the request by reading, locking, and changing (or adding or deleting) records in audited data base files. The application requester is a SCREEN COBOL program. The server can be written in COBOL, TAL, or FORTRAN, and it must follow the record-locking rules imposed by TMF.

To write application requesters and servers that use TMF, you should know:

- the recommended structure for applications that use TMF
- how to use the SCREEN COBOL verbs that support TMF
- how to access audited data base files
- the general guidelines for coding servers
- the record-locking rules that must be followed by processes that change records in audited data base files
- how to avoid deadlock
- the anomalies that can occur during transaction backout
- the considerations involved in converting existing applications for TMF.

This appendix discusses each of these topics in detail.

TASK OVERVIEW

Figure E-1 illustrates the basic tasks involved in programming PATHWAY applications that use TMF.

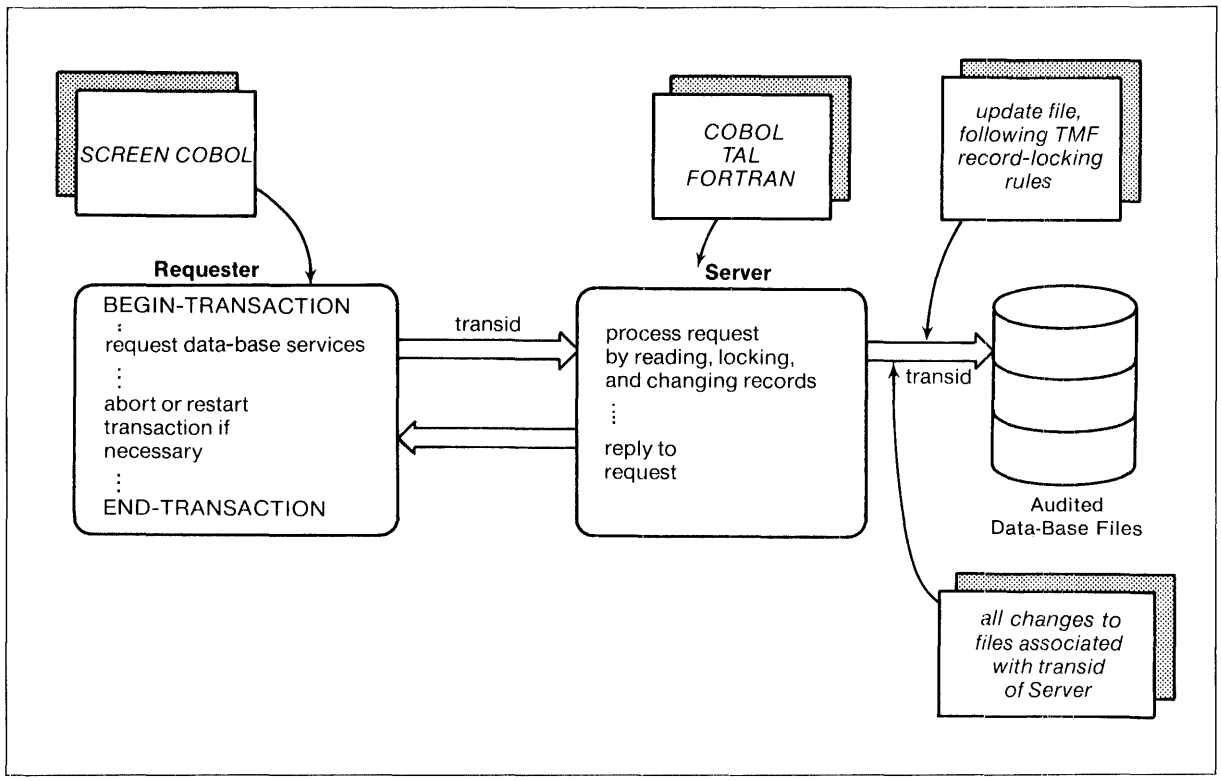


Figure E-1. PATHWAY Programming for TMF

TMF APPLICATION STRUCTURE

The recommended structure for applications that use TMF has the characteristics described below. If your current or planned application has these characteristics, programming the application to use TMF will be relatively straightforward. If not, refer to Application Conversion Considerations in this appendix for a detailed discussion of converting an application to use TMF.

TMF and PATHWAY Application Characteristics

The recommended characteristics for PATHWAY applications that use TMF are discussed in the following paragraphs.

One process (the TCP requester generally) coordinates all of the work required to do a single transaction. This process identifies the beginning and ending points of each TMF transaction. Additionally, if the server replies to a request message by indicating that it failed to complete all of the changes, this process can either abort and abandon the transaction, or abort and retry the transaction according to the SCREEN COBOL application.

The communication between requesters and servers is by standard interprocess I/O. The SCREEN COBOL requester does the SEND and the server does the READUPDATE $RECEIVE and REPLY. Each request message and the server reply to this message is for a single transaction.

Any disc I/O request is for a single transaction. TMF appends the process's current-transaction identifier to each disc-request message so the audit trails can include the identity of the transaction responsible for each data base change.

Any concurrency control is done by using the ENSCRIBE record-locking facilities and all servers should follow the record-locking rules imposed by TMF. ENSCRIBE record-locking gives TMF the control required to ensure that transactions are presented with a consistent view of the data base.

Servers do not reply to request messages until all work for the request has been completed; the contents of the reply message should indicate whether the work for the request is completed successfully or abandoned in a partial state. This characteristic lets the requester decide if the transaction should be committed or aborted.

Servers should not be NonStop; this is unnecessary overhead with TMF and requires additional programming effort.

Servers always perform all of the I/O for the request message most recently read from $RECEIVE and always reply to that message before reading another message; therefore, servers should not do $RECEIVE-queuing.

The rest of this appendix contains detailed information related to writing SCREEN COBOL requesters and servers that use TMF. Refer to Programming Considerations and Application Conversion Considerations in this appendix for information specifically related to the server considerations listed previously.

TMF Restrictions

The following restrictions apply to applications that use TMF:

- A MUMPS program should not be used as a server for a SCREEN COBOL requester under TMF.
- The COBOL feature that provides record-blocking on an unstructured disc file should not be used on audited files.
- A server that provides its own record-blocking, record-caching, or its own manner of record-locking in any form, should not be used to change audited files.

PATHWAY PROGRAMMING USING TMF

PATHWAY applications can be programmed to use TMF by using the following:

- SCREEN COBOL verbs that start and end a transaction, abort a transaction, and restart a transaction
- the special registers TRANSACTION-ID, TERMINATION-STATUS, and RESTART-COUNTER.

Transaction Mode Use

A terminal program unit (that is, a SCREEN COBOL program executing on behalf of a terminal) configured for TMF enters *transaction mode* when the BEGIN-TRANSACTION verb is executed and leaves transaction mode when END-TRANSACTION or ABORT-TRANSACTION is executed. When BEGIN-TRANSACTION is executed, the transaction is assigned a unique transaction identifier (called a *transid*) that distinguishes one transaction from all other transactions. If the program unit is configured with *TMF OFF*, the PATHWAY TCP does not allow that program unit to enter transaction mode, but causes BEGIN-TRANSACTION to issue a null transaction identifier.

For the PATHCOM SUSPEND, STOP, or FREEZE commands, the effect of transaction mode is like setting the STOP-MODE special register to a nonzero value: none of these commands can take effect until the terminal leaves transaction mode and the terminal STOP-MODE register is zero.

The SUSPEND! and FREEZE! commands take effect immediately and cause transaction backout.

The ABORT command takes effect immediately. If the terminal is in transaction mode when this command is executed, the transaction is aborted.

For more details regarding SUSPEND, FREEZE, STOP, and ABORT, refer to the *PATHWAY Operating Manual.*

TMF and SCREEN COBOL Verbs

The following SCREEN COBOL verbs enable PATHWAY programmers to use TMF:

ABORT-TRANSACTION	aborts and backs out a transaction.
BEGIN-TRANSACTION	starts a transaction.
END-TRANSACTION	ends a transaction.
RESTART-TRANSACTION	backs out, then restarts a transaction from the BEGIN-TRANSACTION point.

ABORT-TRANSACTION USE. Generally, this verb is used when the SCREEN COBOL program detects an irrecoverable error and decides to abandon the transaction. When this verb is executed, the transaction is aborted; all updates made by the transaction to audited data files are backed out. The aborted transaction is not restarted automatically.

The form of the ABORT-TRANSACTION verb is:

```
ABORT-TRANSACTION
```

with no options.

Execution of ABORT-TRANSACTION causes the terminal to leave transaction mode and sets the special register TRANSACTION-ID to SPACES.

If the terminal is not in transaction mode when ABORT-TRANSACTION is executed, the terminal is suspended for pending abort and terminal execution cannot be restarted with a RESUME command.

If a fatal error occurs while the transaction is being aborted, and the current BEGIN-TRANSACTION verb does not have an *ON ERROR* clause, then the terminal is *suspended for pending abort*; the current transaction is backed out and terminal execution cannot be resumed with a RESUME command. If the BEGIN-TRANSACTION verb does have an *ON ERROR* clause, that clause is executed and the terminal is not suspended.

BEGIN-TRANSACTION USE. BEGIN-TRANSACTION starts a new transaction; this verb identifies the beginning of a sequence of operations that are treated by TMF as a single transaction. When this verb is executed, the following occurs:

- the terminal enters transaction mode
- TMF is requested to start a new transaction
- the transaction identifier for the new transaction is assigned to the TRANSACTION-ID special register
- the RESTART-COUNTER and TERMINATION-STATUS special registers are reset to zero for the first occurrence of the transaction.

The form of the BEGIN-TRANSACTION verb is:

```
BEGIN-TRANSACTION [ ON ERROR statement ]
```

where

ON ERROR statement

specifies the statement to be executed if the transaction is being restarted or if an error occurs.

The BEGIN-TRANSACTION verb indicates the restarting point to be used if a failure occurs while the terminal is in transaction mode. If the transaction fails for any reason, its data base changes are backed out and (with the exception of the SCREEN COBOL program issuing ABORT-TRANSACTION) execution of the SCREEN COBOL program can be restarted at that point if these conditions are met:

- If *ON ERROR* is absent, the number of times that the transaction has been restarted is compared with the global restart limit specified via the *MAXTMFRESTARTS* option of the SET PATHWAY command in PATHWAY. If the number of restarts is less than that limit, the transaction is restarted with a new transaction identifer, the RESTART-COUNTER special register is incremented by 1, and the TERMINATION-STATUS special register remains set to 1. If the number of restarts equals the transaction restart limit, the terminal is suspended but its execution can be resumed.

- If *ON ERROR* is present, the transaction is restarted, RESTART-COUNTER is incremented by 1, TERMINATION-STATUS remains set to 1, and the *ON ERROR* branch is executed. You can then determine whether or not the transaction should be restarted in the *ON ERROR* branch of the SCREEN COBOL program; for example, RESTART-COUNTER can be compared to a local restart limit established within the program.

If the terminal is already in transaction mode when BEGIN-TRANSACTION is issued, the terminal is *suspended for pending abort*; the current transaction is backed out and terminal execution cannot be resumed with a RESUME command.

The following code sequence accepts input data from the operator and starts a new transaction. In the event of an error, TMF checks to determine if this transaction has been restarted more than two times. If the transaction has been started more that two times, the transaction is aborted and the operator is asked to enter the data again. If the transaction has not been restarted more than two times, another attempt is made to process the transaction.

```
enter-data
.
.
ACCEPT screen
BEGIN-TRANSACTION
   ON ERROR PERFORM check-error.
IF abort-flag NOT = 0
   GO TO enter-data.
.
.
SEND .....
END-TRANSACTION
.
.
stop-trans.
  GO TO enter-data
.
.
check-error.
  MOVE 0 TO abort-flag.
IF TERMINATION-STATUS = 1
   IF RESTART-COUNTER > 2
        ABORT-TRANSACTION
        DISPLAY "Nope" IN MSG
        MOVE 1 TO abort-flag.
```

END-TRANSACTION USE. END-TRANSACTION indicates that the transaction is complete. When this verb is successfully executed, the data base updates made by the transaction become permanent, the terminal leaves transaction mode, and the special register TRANSACTION-ID is set to SPACES.

If TMF rejects END-TRANSACTION, the SCREEN COBOL program is restarted at the BEGIN-TRANSACTION point.

The form of the END-TRANSACTION verb is simply

```
END-TRANSACTION

with no options.
```

If the terminal is not in transaction mode when END-TRANSACTION is executed, the terminal is suspended for pending abort; terminal execution cannot be resumed with a RESUME command.

RESTART-TRANSACTION USE. RESTART-TRANSACTION is used when the SCREEN COBOL program detects an error that might be temporary, abandons the current attempt, and retries the transaction. When this verb is executed, the following occurs:

- the current execution of the transaction is backed out
- the transaction is restarted at the BEGIN-TRANSACTION point with a new transaction identifier
- the special register RESTART-COUNTER is incremented by 1.

The form of the RESTART-TRANSACTION verb is:

```
RESTART-TRANSACTION

with no options.
```

The restart due to executing RESTART-TRANSACTION counts as a restart for purposes of the global transaction restart limit.

If the terminal is not in transaction mode when RESTART-TRANSACTION is executed, the terminal is suspended for pending abort; terminal execution cannot be resumed with a RESUME command.

TMF and Special Registers

Special registers are data items defined automatically by the SCREEN COBOL compiler, not by the programmer. Three special registers have been provided for TMF users:

- TRANSACTION-ID
- TERMINATION-STATUS
- RESTART-COUNTER.

The special registers are described in the following paragraphs.

TRANSACTION-ID. Executing BEGIN-TRANSACTION sets TRANSACTION-ID to the value of the transaction identifier. Executing END-TRANSACTION or ABORT-TRANSACTION sets this register to SPACES.

TRANSACTION-ID has this implicit declaration:

```
01 TRANSACTION-ID PIC X(8).
```

TERMINATION-STATUS. Executing BEGIN-TRANSACTION sets the value of TERMINATION-STATUS to indicate the outcome of BEGIN-TRANSACTION. The following values are possible:

1 The transaction is started or restarted.

2 TMF is not installed. If there is no *ON ERROR* phrase, the default system action is to suspend the terminal for pending abort.

3 TMF is not started. If there is no *ON ERROR* phrase, the default system action is to suspend the terminal, but the terminal can be restarted by the PATHCOM command RESUME.

4 A fatal error occurred. If there is no *ON ERROR* phrase, the default system action is to suspend the terminal for pending abort.

TERMINATION-STATUS has this implicit declaration:

```
01 TERMINATION-STATUS PIC 9999 COMP.
```

RESTART-COUNTER. Executing BEGIN-TRANSACTION sets RESTART-COUNTER to the number of times the transaction has been restarted. RESTART-COUNTER is reset to zero when BEGIN-TRANSACTION is first executed for a particular transaction.

RESTART-COUNTER has this implicit declaration:

```
01 RESTART-COUNTER PIC 9999 COMP.
```

Refer to the BEGIN-TRANSACTION verb, in this appendix, for an example of how to use RESTART-COUNTER to selectively limit the number of times a transaction is retried.

TMF PROGRAMMING CONSIDERATIONS

The following topics are general considerations related to programming for TMF:

- accessing audited data base files
- record locking
- coding servers
- avoiding deadlock
- TMF backout anomalies.

Accessing Audited Data-Base Files

Audited data base files are files that reside on audited volumes and have been designated as audited by use of FUP or the CREATE procedure. Before and after images of all changes to these files are written to the audit trails that have been configured for the audited volumes. Figure E-2 illustrates the differences between processes that change audited files and those that can change only non-audited files.

A server that has a transaction identifier can read, lock, insert, delete, and change records in audited files. A transaction identifier is created when a SCREEN COBOL program issues the BEGIN-TRANSACTION verb. A server acquires a transaction when reading $RECEIVE to pick up a request message generated by a SCREEN COBOL SEND statement.

In PATHWAY, servers can lock and change records in an audited file only if they are members of a server class that is defined as a TMF server class. Refer to the *PATHWAY Operating Manual* for an explanation of how to configure server classes.

A process that does not have a current-transaction identifier can read records in audited files, but cannot lock or change them.

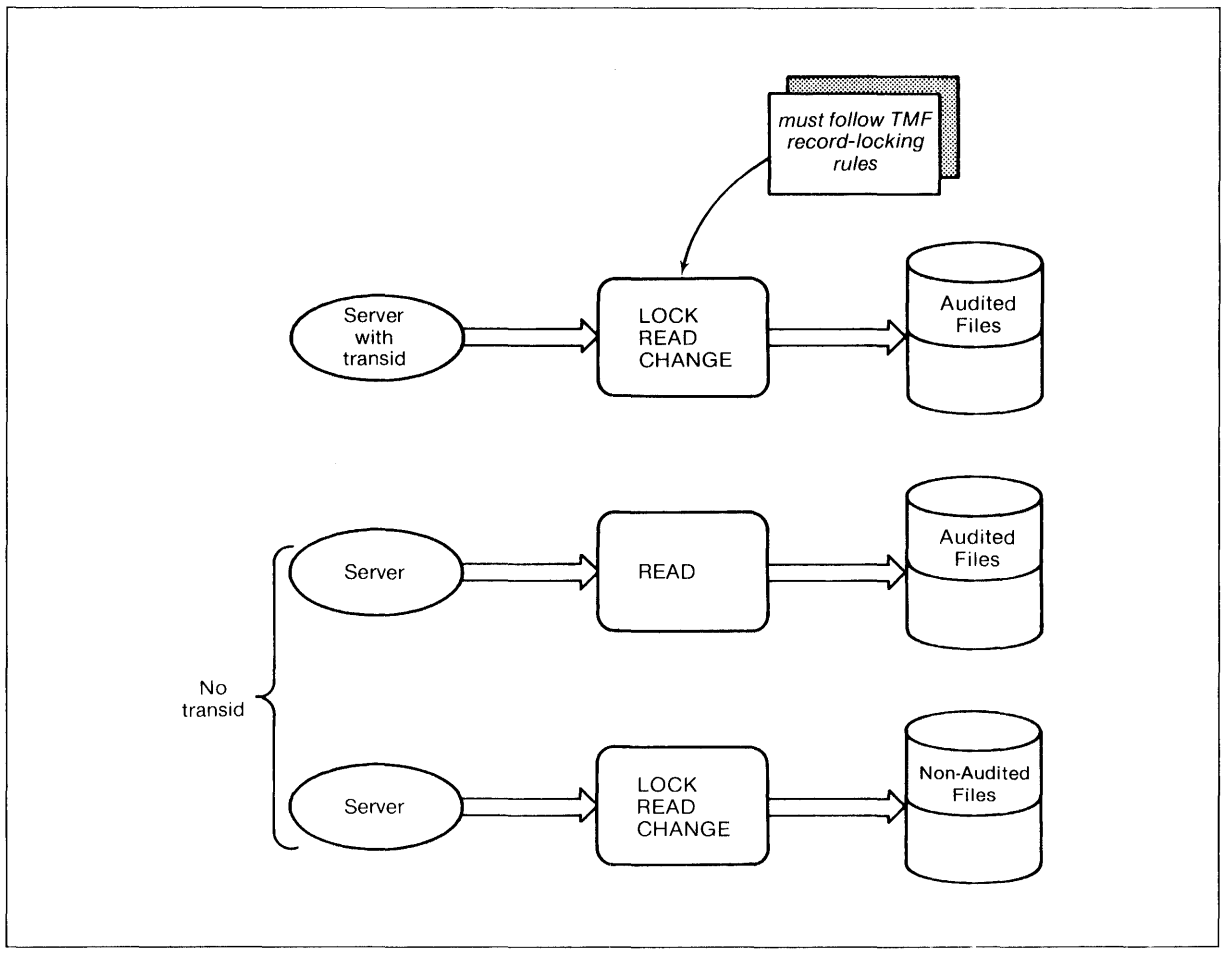


Figure E-2. Accessing and Changing Audited and Non-Audited Files

Record Locking

For all changes to audited files, TMF enforces the following record-locking protocol:

- An existing record must be locked by a transaction before the record can be changed or deleted by a transaction.
- TMF locks all records inserted by a transaction.
- TMF locks the primary keys of all records deleted by a transaction.
- TMF will not release the locks for any record changed, inserted, or deleted by a transaction until the transaction either is committed or aborts and is backed out.

Locks can be acquired individually on a record-by-record basis or a lock can be acquired for an entire file by using the GUARDIAN LOCKFILE procedure. Figure E-3 illustrates how processes can acquire locks, update audited files, and when TMF will release the locks. Mixing record locks and file locks in the same file is not supported. Record locks cannot be granted while a file is locked.

If the entire set of current active transactions tries to acquire more than 922 key locks or 1808 record locks per file, error 32 (CONTROL BLOCK SPACE UNAVAILABLE) is returned as a file error.

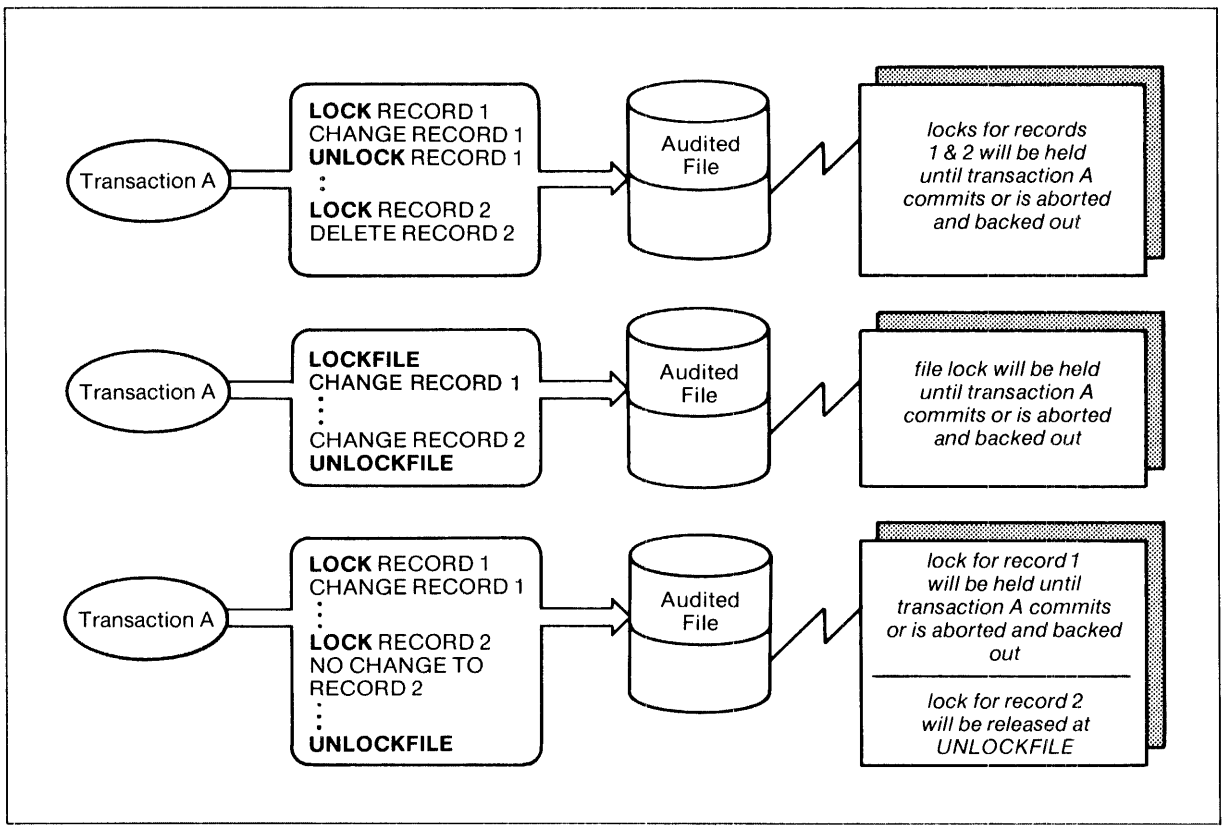


Figure E-3. Record Locking for TMF

The file lock or record locks are owned by the current-transaction identifier of the process that issued the lock request. In PATHWAY, a single transaction can send requests to several servers or multiple requests to the same server class. In the situation where several processes share a common transaction identifier and the locks are held by the same transaction identifier, the locks do not cause conflict among the processes participating in the transaction if all are record locks or all are file locks. Record locking by transaction identifier is illustrated in Figure E-4.

Figure E-4 illustrates the following principles:

- The TCP interprets BEGIN-TRANSACTION and obtains the transaction identifier before requesting data base activity from the servers.
- The transaction identifier is transmitted to the servers in the request message and any disc activity performed by the servers is associated with the transaction identifier.
- The transaction identifier owns the locks; all servers that acquired the same transaction identifier can read, lock, add, delete, and change records in the audited files. For example: server A can read and lock a record and server B can read or change the same record, if both servers A and B have the same transaction identifier.

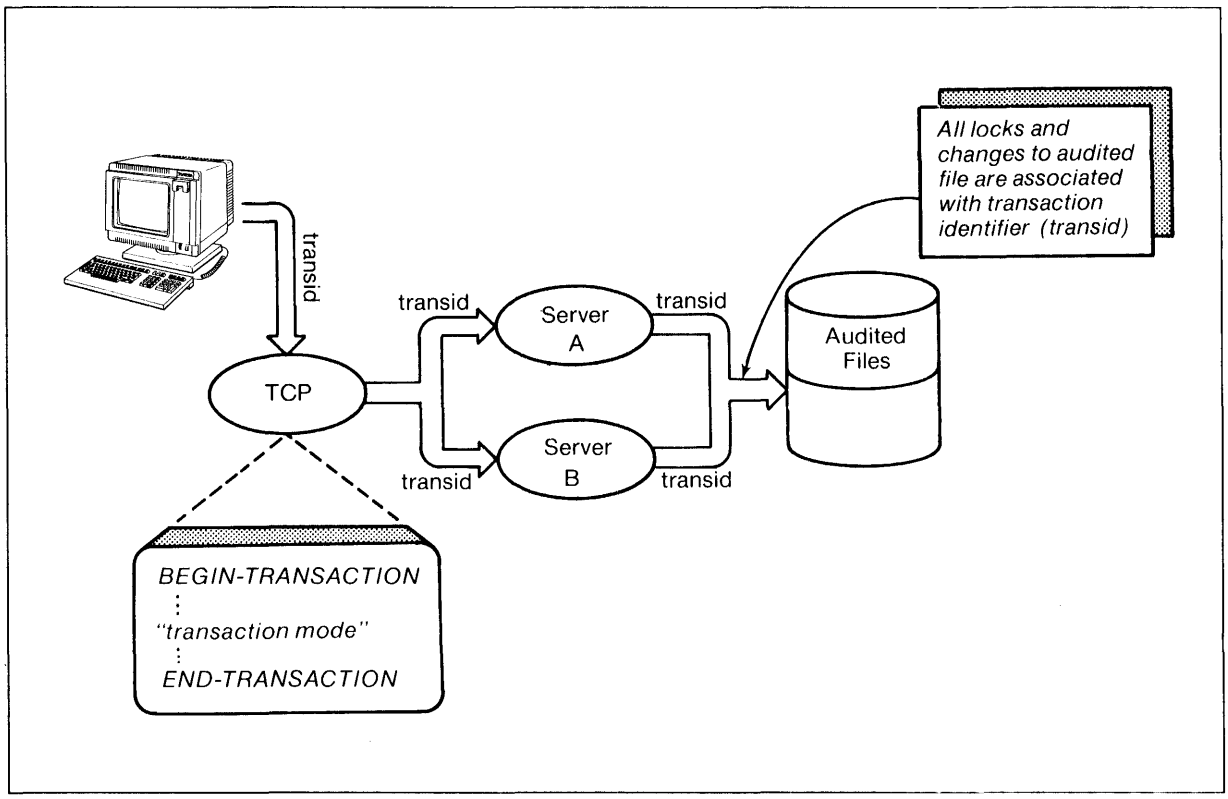


Figure E-4. Record Locking by Transaction Identifier

REPEATABLE READS. Generally, a transaction should lock any data read during the transaction and used in producing output, regardless of whether the data is modified. Following this rule guarantees that all the transaction read operations are repeatable and that data on which the transaction depends does not change before the transaction is committed.

OPENING AUDITED FILES. Because locks are owned by the transaction identifier instead of the process identifier and the identifier of the file opener, they can persist longer than the opener process. This means that even if a file has been closed by all its openers, the disc process keeps that file effectively open until all transactions owning locks in the file have ended or have been aborted and backed out.

The following types of errors are possible for files that have pending transaction locks:

- Attempting to open an audited file with exclusive access will fail with file error 12 (FILE IN USE), regardless of whether openers of the file exist.
- FUP operations requiring exclusive access such as PURGE and PURGEDATA will fail. PURGE fails with file error 12 and PURGEDATA fails with file error 80.

Additionally, error 80 (INVALID OPERATION ON AUDITED FILE OR NON-AUDITED DISC VOLUME) is returned for the following open situations:

- attempting to open an audited file having an automatically updated key file that cannot be opened or is not audited
- attempting to open an audited file that does not reside on an audited volume
- attempting to open a structured audited file with unstructured access
- attempting to open an audited, partitioned file having a non-audited secondary partition.

READING DELETED RECORDS. If transaction T1 *deletes* a record and another transaction T2 attempts to read the same record while T1 is still active, the following occurs:

- If the read request is the GUARDIAN procedure READ after exact positioning, file error 1 (END-OF-FILE) is returned.
- If the read request is the GUARDIAN procedure READUPDATE, file error 73 (FILE/RECORD LOCKED) is returned in alternate locking mode and the request waits for T1 to complete in default locking mode.

BATCH UPDATES. When programming for batch updating of audited files, you should either have the transaction lock an entire file at a time by using the LOCKFILE procedure or carefully keep track of the number of locks held. If you do not use LOCKFILE, TMF sets two implicit locks:

- When a new record is inserted in an audited file, TMF implicitly locks that record.
- When a record is deleted from an audited file, TMF implicitly locks the key of that record.

These locks are not released until the transaction is committed or is aborted and backed out. This means that transactions doing batch updates to audited files (if they involve deleting or inserting a large number of records) can obtain too many locks. The maximum number of locks that can be acquired for each file is 922 key locks and 1808 record locks. In this situation, error 32 (CONTROL BLOCK SPACE UNAVAILABLE) is returned as a file error.

If a transaction calls LOCKFILE for a primary-key file, LOCKFILE is also applied to any associated alternate-key files. This prevents primary-file updates from causing the alternate-key files to obtain record locks (for insertions) or key locks (for deletions).

Coding Servers

Figure E-5 illustrates the typical sequence of actions performed by a single-threaded (not $RECEIVE-queuing) server.

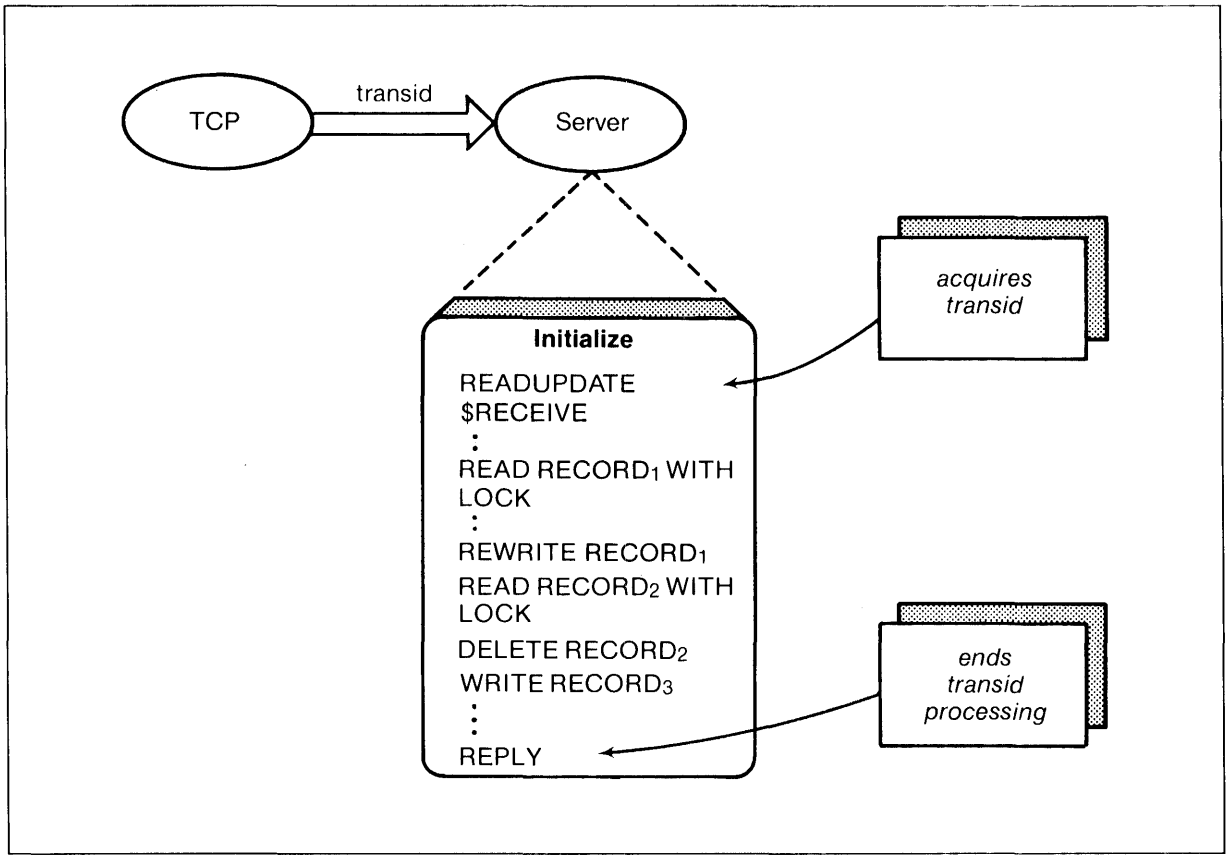


Figure E-5. Nonqueuing Server

When you write servers of the type illustrated in Figure E-5, consider the following:

- When the server reads $RECEIVE to pick up a request message, the server automatically acquires the transaction identifier of the process that sent the message. All data base operations performed from the point of the read on $RECEIVE until the server replies are associated with the transaction identifier.
- Existing servers that are not NonStop servers (do not do $RECEIVE-queuing) and lock all records before making a change generally do not have to be modified for TMF.
- The server must follow the record-locking rules imposed by TMF. This means the server must lock all records to be deleted or changed.

Refer to the *Transaction Monitoring Facility (TMF) System Management and Operations Guide* for a description of actions performed by $RECEIVE-queuing servers.

Deadlock

The following example of a sequence of record locking operations results in a deadlock situation:

1. Transaction 1 locks record A.
2. Transaction 2 locks record B.
3. Transaction 1 attempts to lock record B and has to wait.
4. Transaction 2 attempts to lock record A and has to wait.

Neither transaction can proceed and the situation is a deadlock.

Some deadlock situations that can occur because of the TMF record locking protocol are:

Deleting a record implicitly locks the key of the record and can cause the deadlock situation illustrated in Figure E-6.

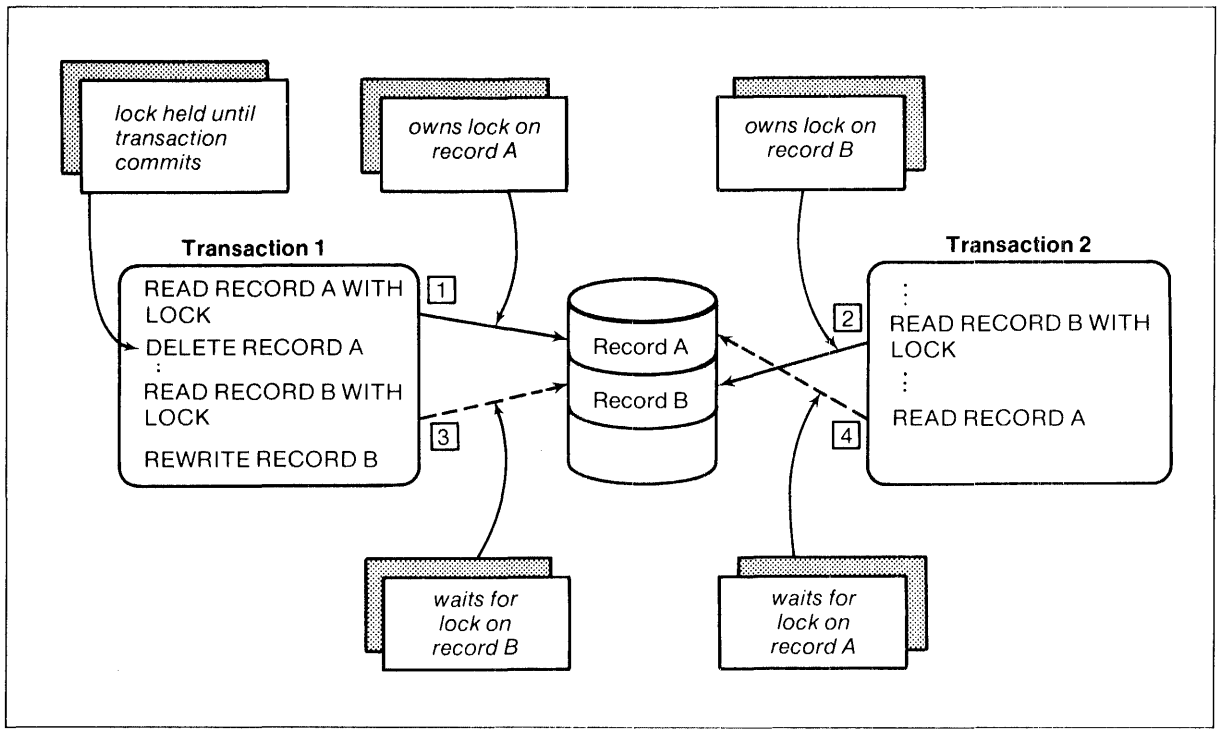


Figure E-6. Deadlock Caused by Deleting a Record

A record inserted by a transaction is automatically locked and can cause the deadlock situation as illustrated in Figure E-7.

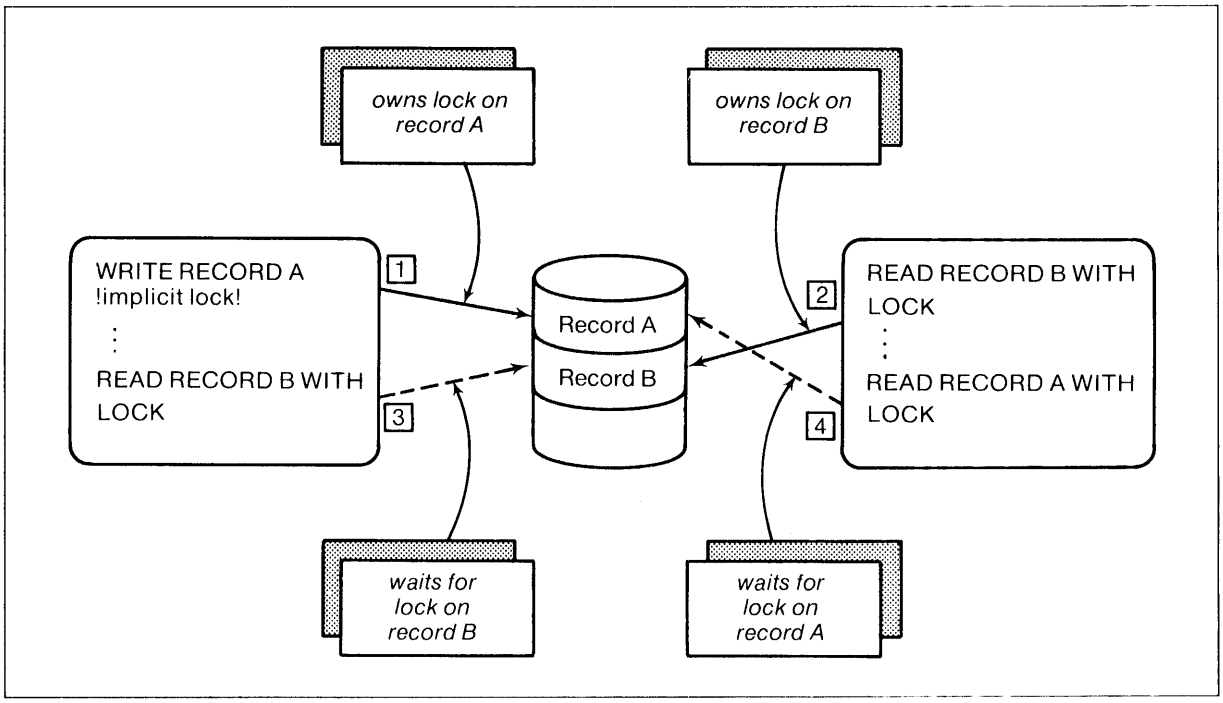


Figure E-7. Deadlock Caused by Inserting a Record

A process can deadlock itself as illustrated in Figure E-8 if the process acquires different current-transaction identifiers.

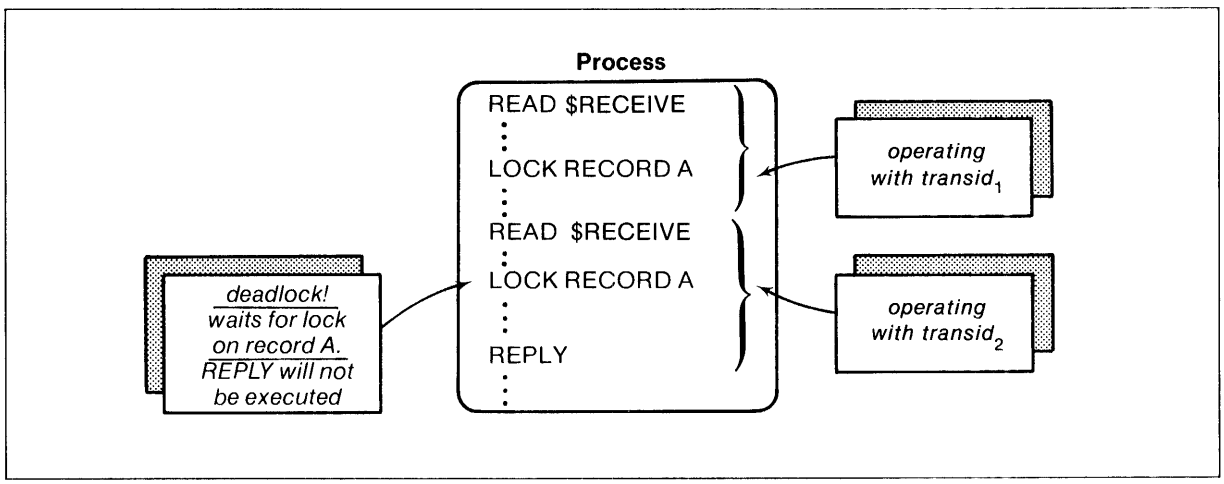


Figure E-8. Deadlock Caused by a Process Switching Transaction Identifiers

Multiple SEND statements to one PATHWAY server (if they cause the server to access the same record under a different transaction identifier) can cause the server to participate in a deadlock as illustrated in Figure E-9. This situation only occurs if different TCPs are involved in the send operations.

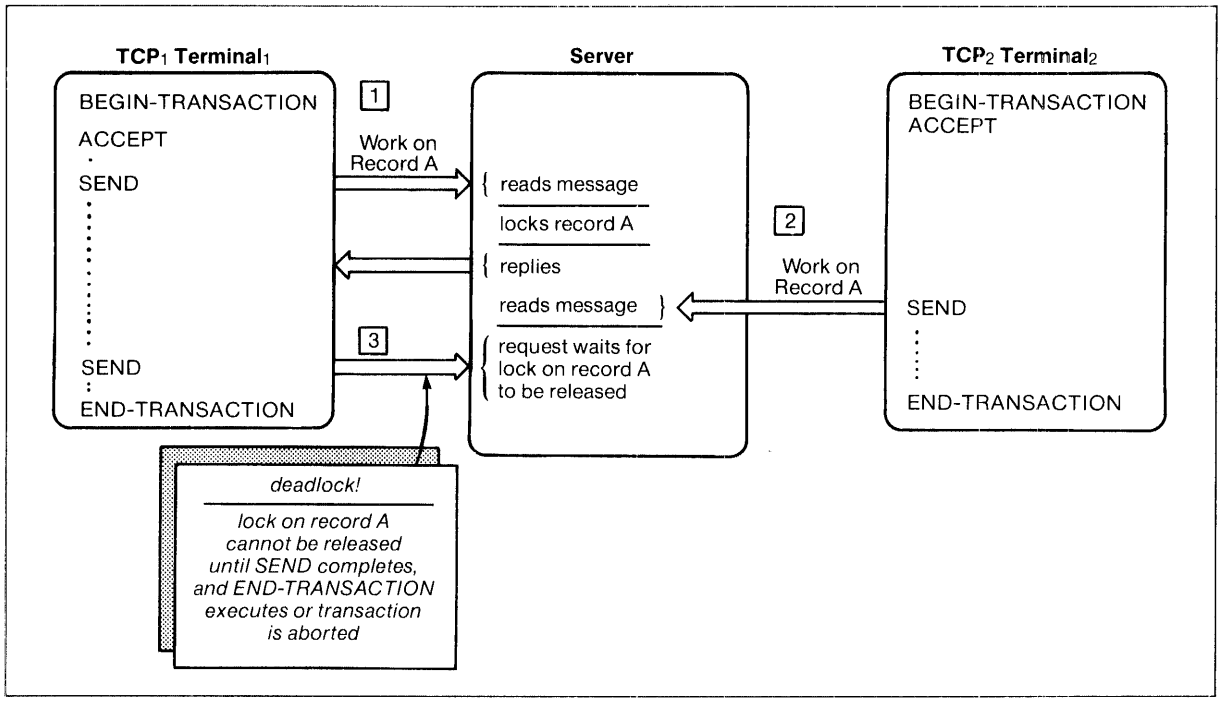


Figure E-9. Deadlock Caused by Multiple SEND Statements

There is no way of detecting if a transaction becomes involved in a deadlock. However, the following situations can be detected:

- A transaction is attempting to read or lock a record that is already locked.
- A transaction read or lock request is waiting too long before completion.

Each of the above situations is explained in the following paragraphs and illustrated in Figure E-10. In either situation, it is safe to assume (although it might not be true) that the transaction is in a deadlock. Code the transaction to either abort or restart. The locks held for the transaction will be released, avoiding the possibility of the transaction participating in or prolonging a deadlock. A PATHWAY server can return a message to the SCREEN COBOL requester indicating the deadlock possibility and the requester can then use the ABORT-TRANSACTION or RESTART-TRANSACTION verb.

TAL programmers can determine if a record is already locked by using the GUARDIAN SETMODE procedure to select alternate locking mode. In this mode, file error 73 can be returned to the server when that server attempts to access a locked record.

In default locking mode, TAL programmers can determine if an I/O request has waited too long before completion. In this mode, a server process will be suspended when the server attempts to access a locked record. To avoid deadlock, open the file using no-wait I/O and specify a nonzero time limit in the call to AWAITIO. If AWAITIO returns GUARDIAN error 40 (indicating timeout), the transaction might be in a deadlock situation.

COBOL programmers can open files using the *WITH TIME LIMITS* parameter. *WITH TIME LIMITS* indicates that further I/O requests will be timed by a value that is specified in the *TIME LIMIT* parameter of the request. If the I/O request times out, GUARDIAN error 40 will be returned to the request.

FORTRAN programmers can open files with the *TIMED* specifier and use the *TIMEOUT* specifier in their I/O requests to specify a timeout value. If the I/O request times out, GUARDIAN error 40 will be returned to the request.

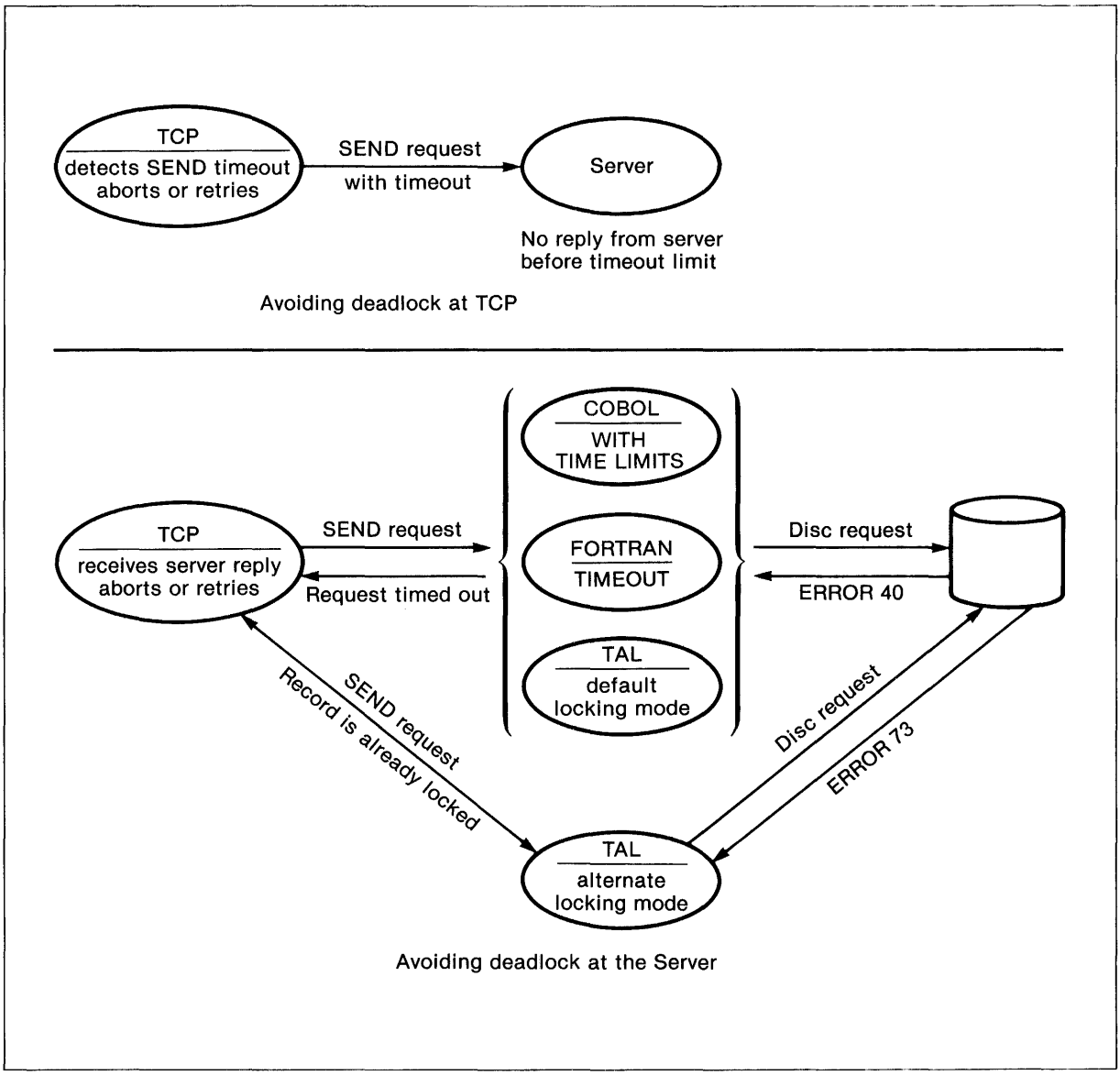


Figure E-10. Avoiding Deadlock

Backout Anomalies

When a transaction aborts, TMF backs the transaction out in the following sequence:

- Records updated by the transaction are backed out by a WRITEUPDATE of the before-images for each of the updated records.
- Records deleted by the transaction are backed out by a WRITE (insert) of the before-images for each of the deleted records.
- Records inserted by the transaction are backed out by a *write count* zero WRITEUPDATE (delete).

Because of this sequence, certain types of anomalies can occur during backout:

- Insertions at end-of-file (EOF) to an *unstructured file* cannot be backed out. This means that EOF will not be restored to its previous value, because another transaction might have written to EOF after the insert but before the backout.
- If a record A is inserted at EOF of an *entry-sequenced file* and no other record has been inserted after record A, backout of record A involves the disc process moving the EOF pointer to the previous record. If another record already follows record A, however, record A is backed out by rewriting record A with a length of zero bytes; that is, making it an empty record. A READ at record A address then returns a null record with a length of zero bytes.
- An EOF (-1D POSITION) insertion to a *relative file* is backed out by deleting the record. However, EOF is not restored to some previous value, because another transaction might have written to EOF after the insert but before the backout.
- Backout can fail for a transaction that deletes records from a *key-sequenced file* that is near a file full condition. This occurs if other transactions, concurrent with the transaction that deleted the record, insert enough records to fill the file. If the file is full, the transaction that deleted the records cannot be backed out because there is insufficient space to insert the records that it deleted. If this happens, the console receives a message like this:

```
89  11:43 21APR81 FROM 004,01,018 LDEV 0022 CU % 420 BACKOUT ERROR
#0045 TRANSACTION SEQ #00000238
```

  and the transaction to delete the record remains in an aborting state. Note that the message reflects GUARDIAN error 45, FILE IS FULL. Here, you could use DELETE TRANSACTION to unlock affected files. However, remember that this could cause an inconsistency in your data base, as the deleted record might not be restored. If your application programs maintain a log to tie the transaction identifier to the user-entered transaction, you might be able to correct the problem manually. It could also be possible to run a separate transaction to delete one or more unneeded records, then use the ABORT TRANSACTION command to complete the previously unsuccessful backout. (A better solution for this problem is to use the FUP INFO command periodically and make sure your files never get that full.)

- A request to write to a *relative file* using -2D positioning is changed to -1D positioning if there are active key locks on the file when the disc process handles the request. A -2D request specifies *insert at any available position* and a -1D request specifies *insert at EOF*. Active key locks imply the existence of uncommitted transactions that have deleted records. And, if it is necessary to abort and back out the uncommitted transaction, the deleted records need to be reinserted at the same record addresses. This means that if the -2D request were honored, one of the reserved record addresses might be used and transaction backout would be impossible for the transaction that deleted the record at the reserved address. Refer to Record Locking in this appendix for a discussion of key locks.

APPLICATION CONVERSION CONSIDERATIONS

Converting an existing application to use TMF requires that you do the following:

1. Decide which files in the data base should be audited.

2. Determine what (if any) modifications are necessary to convert the application to follow the TMF record-locking rules.

3. Decide how to group sequences of the application operations into TMF transactions (that is, units of recovery).

4. Add the necessary transaction control statements to the SCREEN COBOL requester to enable the program to begin and end the transaction, and to enable the program to abort or restart the transaction if necessary.

5. Ensure that any NonStop servers respond correctly to error 75 (REQUESTING PROCESS HAS NO CURRENT-TRANSACTION IDENTIFIER). A backup server that takes over in mid-transaction does not have a current-transaction identifier to send to the disc process; therefore, the disc process returns error 75 to the server, which passes the error along to the requester. If the requester aborts and retries the transaction, the new request has a current-transaction identifier. The preferred solution is to change the servers so they are not NonStop servers.

6. Modify any $RECEIVE-queuing servers to assume a new request transaction identifier whenever such a server begins work on a new queued message.

7. Determine if any new deadlock situations are introduced as a result of TMF implicit record locking and modify the application to avoid the deadlock. One way to avoid deadlock is to use timeout.

Audited Files

TMF recovery strategy involves backing out the aborted transaction changes, which enables the transaction to be reexecuted from the beginning (with a new transaction identifier). This means that if you decide to have a mixture of audited and non-audited files in the data base, you must be careful: only changes to audited files will be backed out. And if a transaction works on a mix of audited and non-audited files, the operations on the non-audited files must be *retryable*.

A retryable operation is an operation that can be interrupted and repeated an indefinite number of times without affecting the consistency of the data base; for example, all reading operations are retryable. Whether or not a writing operation (on a non-audited file) is retryable depends on your criteria for consistency of the data in the data base. If the transaction changes both audited and non-audited files, you should analyze it to determine whether backing out and reexecuting the transaction affects consistency.

For example, consider a transaction that extracts records from a data base, computes some aggregates like averages or means, and then uses the aggregates to extract a subset of the extracted records from the data base for summary reporting. This transaction can be implemented by doing the extraction twice, the first to compute the aggregates and the second to extract the subset. You can place the extracted records in a non-audited scratch file (each server can have its own scratch file to avoid conflict). If the transaction is aborted and restarted, the transaction starts writing the scratch file from the beginning and there is no real need for the scratch file to be audited.

Another example is logging all input messages to a server, which allows examination of them after a failure. It is self-defeating to designate the log file as an audited file; the message that caused the failure would be backed out.

Record Locking Conversion

An application that does not follow the TMF record locking rules must be modified to adhere to them. Refer to Record Locking for a complete discussion of the rules and for a discussion of the factors that could change your application conversion strategy with respect to locking. These factors include:

- repeatable reads
- the errors that result from locks being held by the transaction identifier instead of the process identifier and *openid* of the file opener
- the errors that result from reading deleted records
- batch updates by a transaction that acquires a large number of locks. (Batch updating should use file locks instead.)

Grouping Transaction Operations

Your application can view the transaction as a logical unit of work; for example, entering the order header along with all of the detail items in a purchase order. However, TMF treats the transaction as a physical unit of recovery. When you convert applications to use TMF, this difference must be considered. Basically, this means deciding how to answer certain questions. What is the logical unit of work that you want to accomplish within an application? How can the work be divided into a number of transactions that can be recovered by TMF? Factors that influence the answers to these questions are:

- Concurrency: How long will record locks be held by a transaction?
- Performance: How much extra disc I/O, server activity, and TCP paging is involved in the choice of one conversion strategy over another?
- Consistency: Are the units of recovery large enough to ensure that your criteria for consistency will be maintained?

In view of these factors, two guidelines will generally help you decide how to group the data base accesses made by an application into a single transaction. The first is: any group of accesses that together modify the data base from one consistent state to another consistent state should be a single TMF transaction. The second is: any group of accesses that require a consistent view of the data base should be a single TMF transaction.

The following examples demonstrate application of the previous guidelines:

EXAMPLE 1: Some logical transactions do not have to be identified as TMF transactions. For example, a logical transaction locates a single record and displays the record contents. Since this transaction changes nothing in the data base, it does not affect consistency and does not have to be a TMF transaction.

EXAMPLE 2: A data entry transaction with a group of accesses that insert new data into the data base should be a TMF transaction. For example, a logical transaction records receipt of some items for a stockroom by accepting the stock codes and quantity received from a data entry operator, and then updates the records (in an audited file) for the items. The first guideline applies, and you should arrange to begin a TMF transaction after the data is accepted, and to end the transaction after the last record is updated. TMF ensures that all changes resulting from the one operator entry are either permanent or backed out in case the transaction aborts. Note that since any change to an audited file requires a transaction identifier, this example is also true if the transaction only inserts one record in the file.

EXAMPLE 3: An update transaction should be a TMF transaction. For example, assume a logical transaction does the following:

1. accepts a specification from the operator
2. performs the equivalent of an inquiry operation to find the data that will be updated
3. releases the locks obtained for the inquiry
4. displays the data for the operator
5. accepts modifications to the displayed data (saving a copy of the original displayed data)
6. performs the inquiry a second time
7. verifies that the results of the first inquiry and the second inquiry are the same
8. writes the modified record to the data base.

The transaction should be implemented as two TMF transactions. The first should begin after the data is accepted and should end (in place of releasing the locks) after the last record is read. The second should begin after the modifications to the displayed data have been accepted and should end after the last modified record is written to the data base. However, if the inquiry part of the transaction is just a single read, there is no need for the first inquiry to be part of a TMF transaction.

Transaction Control

The transaction control verbs for SCREEN COBOL requestors are BEGIN-TRANSACTION, END-TRANSACTION, ABORT-TRANSACTION, and RESTART-TRANSACTION.

For PATHWAY applications, transaction control simply involves adding BEGIN-TRANSACTION and END-TRANSACTION verbs to the SCREEN COBOL units and using ABORT-TRANSACTION or RESTART-TRANSACTION in the places where the SCREEN COBOL unit itself handles transaction failure. PATHWAY handles a number of failure cases itself by automatically aborting the transaction and restarting it at the BEGIN-TRANSACTION point. The TCP performs the following:

- takes care of all details involved in handling concurrent active transactions
- keeps track of the transaction identifiers for multiple transactions
- checkpoints the transaction identifier
- generally operates as a NonStop process
- handles the TMF-related programming involved when the backup process takes over.

NonStop Servers

NonStop operation of servers is unnecessary overhead with TMF. However, if your applications use NonStop servers, it is not usually necessary to change them from NonStop to ordinary servers. Nevertheless, you should note that if the primary server process fails, the backup process (on takeover) does not have a current-transaction identifier. This means that the server process receives error 75 (NO CURRENT-TRANSACTION IDENTIFIER) on the first I/O request to an audited file. If the server is coded to recognize this error and report it as a failure to the requester, or if the server is coded to terminate (both processes if the backup creates a new backup immediately) when it receives this error, then it can be a NonStop server.

Since the COBOL runtime library recognizes the PARAM named NONSTOP, you can disable NonStop operation of COBOL servers by having a PARAM NONSTOP OFF in effect when the server is started. The runtime library will ignore the STARTBACKUP and CHECKPOINT verbs and store the *successful completion* code in the PROGRAM-STATUS special register. For PATHWAY, PARAM NONSTOP OFF can be included with the parameters in the definition of the server class during PATHWAY configuration.

There is no equivalent mechanism for disabling NonStop servers coded in TAL or FORTRAN.

Deadlocks and Conversion

An application that uses TMF might hold more record locks and hold them longer than without TMF. This occurs because:

- Implicit locks are held on the keys of deleted records.
- Implicit locks are held for inserted records.
- Locks are held until the transaction is either committed or aborted and backed out.

The increased locking might cause new deadlock possibilities for the application and should be studied to determine if the possibilities exist. If they do and deadlock can become a problem, consider implementing the deadlock avoidance schemes discussed in the Deadlock paragraph.

PATHWAY INTERACTION WITH TMF

The rest of this appendix discusses information about three basic questions related to PATHWAY interaction with TMF.

1. How do the settings you specify for the TMF parameter of the SET SERVER, SET TERM, and SET PROGRAM commands affect SCREEN COBOL SEND statements?

2. How is TCP checkpointing strategy affected by the settings you specify for the TMF parameter of the SET SERVER command?

3. What problems are caused by using the TMF OFF option of the SET TERM and SET PROGRAM commands as a *switch* to turn TMF *off* for a PATHWAY system that is configured for TMF?

Understanding the answers to the preceding questions could help to ensure the consistency of the data base and help you to improve the reliability and performance of the applications that use the data base.

SET SERVER Command and TMF

The SET SERVER command contains a TMF parameter with an ON or OFF option. By setting this parameter you control how a TCP allows access to a server class, that is, the types of operations a server class can perform.

If you specify ON for the TMF parameter, the TCP allows a SEND to members of this server class whether or not the SCREEN COBOL program is in transaction mode.

If you specify OFF for the TMF parameter, the TCP allows a SEND to the members of this server class only if the SCREEN COBOL program is not in transaction mode. OFF is the default setting.

In addition, the TCP makes checkpointing decisions based upon the option specified for the TMF parameter. You must match the TMF parameter setting to the application environment. Refer to the TCP Checkpointing Strategy paragraph later in this appendix.

SET TERM and SET PROGRAM Commands and TMF

The commands SET TERM and SET PROGRAM each contain a TMF parameter with an ON or OFF option.

If you specify ON for the TMF parameter, the TCP will invoke the corresponding GUARDIAN operating system procedure for any TMF verb issued from a SCREEN COBOL program. ON is the default setting whether or not TMF is running with PATHWAY.

If you specify OFF for the TMF parameter, the TCP will not invoke the corresponding GUARDIAN operating system procedure for any TMF verb issued from a SCREEN COBOL program. Instead, the verb will appear (to the SCREEN COBOL program) to complete successfully and the program can continue to execute. Under special circumstances (discussed later in this appendix), specifying the OFF option is a convenient way to partially test programs on a PATHWAY system that does not have TMF running.

For most PATHWAY applications, whether or not TMF is running, you should use the default parameter settings and ignore the OFF options.

EFFECTS OF THE TMF PARAMETER ON PATHWAY SEND OPERATIONS

Table E-1 illustrates what happens to a SCREEN COBOL SEND statement for the various settings of the TMF parameter in the SET TERM, SET PROGRAM, and SET SERVER commands; PATHWAY and TMF are both assumed to be running on the system. Depending on the type of file access attempted, PATHWAY either allows the SEND statement to execute or issues the appropriate error message.

In Table E-1, *transaction mode* (Trans. Mode) indicates that the SEND is executed after a SCREEN COBOL program has issued a BEGIN-TRANSACTION statement, but before the program has issued an END-TRANSACTION or an ABORT-TRANSACTION statement.

In a PATHWAY system that normally runs with TMF, the SET SERVER TMF ON and SET TERM or SET PROGRAM TMF OFF combination (shown in Table E-1) should not be viewed as a way to temporarily turn off TMF. This condition will appear to allow normal PATHWAY operation because the BEGIN-TRANSACTION statement that would have failed with TMF stopped now appears to work; the TCP allows a SEND to a server that can access and update nonaudited files. In this condition, you should note that both the normal TMF consistency for files accessed by the server and the correct PATHWAY NonStop operations will *not* be maintained.

Table E-1. SEND Operations With TMF

PATHWAY COMMAND	Audited Files		Non-Audited Files	
	Trans. Mode	**Non-Trans. Mode**	**Trans. Mode**	**Non-Trans. Mode**
SET SERVER TMF ON	(1) OK	(2) GUARD. Error 75	OK	OK
SET SERVER TMF OFF	PATHWAY SEND Error 13	GUARD. Error 75	PATHWAY SEND Error 13	(3) OK
SET TERM TMF ON SET PROGRAM TMF ON				

	Audited Files		Non-Audited Files	
	Trans. Mode	**Non-Trans. Mode**	**Trans. Mode**	**Non-Trans. Mode**
SET SERVER TMF ON	GUARD. Error 75	(1) GUARD. Error 75	(4) OK	OK
SET SERVER TMF OFF	PATHWAY SEND Error 13	GUARD. Error 75	PATHWAY SEND Error 13	OK
SET TERM OFF SET PROGRAM OFF				

GUARDIAN File Management Error 75 —

Indicates that no TRANSACTION-ID was present; PATHWAY has allowed the SEND and the server receives the error.

A GUARDIAN error indicates the SEND was allowed by the TCP, but the GUARDIAN operating system did not permit a lock or update operation on the disc file.

PATHWAY SEND Error 13 —

Indicates a TMF mode violation; the error is returned by the TCP in the TERMINATION-STATUS special register.

NOTES:

(1) SET SERVER TMF ON, SET TERM and SET PROGRAM TMF ON are the parameter settings for normal TMF and PATHWAY operation. The SET SERVER TMF ON must be set with PATHCOM.

(2) This SEND is allowed because a server is assumed to have read-only access to the files; attempts to lock or update a record in an audited file will result in a GUARDIAN error 75.

(3) SET SERVER TMF OFF, SET TERM and SET PROGRAM TMF ON are the parameter settings for normal non-TMF PATHWAY operation. These are the TMF parameter defaults.

(4) SET SERVER TMF ON, SET TERM and SET PROGRAM TMF OFF are the parameter settings for special program testing.

The SET SERVER TMF ON, SET TERM and SET PROGRAM TMF OFF, and nonaudited files combination is a convenient way to partially test or debug a SCREEN COBOL program on a system that does not yet have TMF configured. The program will execute, and all SEND requests to audited files will receive error replies.

TCP CHECKPOINTING STRATEGY

For PATHWAY systems with TMF running, the TCP uses the following checkpointing strategy:

- At BEGIN-TRANSACTION — A full context write to the swap file and a checkpoint to the backup is performed.
- At END-TRANSACTION — A full context checkpoint is performed.
- For a SEND to a non-TMF server (SET SERVER TMF OFF) outside of transaction mode — A checkpoint is performed before and after the SEND. Note that a SEND to a non-TMF server that operates on nonaudited files indicates PATHWAY operation without TMF; the TCP checkpoints the SEND context. If the non-TMF server attempts to lock or update a record in an audited file, an error is returned.
- For a SEND to a TMF server (SET SERVER ON) — No checkpoints are performed whether or not the SCREEN COBOL program is in transaction mode. This strategy means that SEND requests to TMF servers that operate on audited files will require fewer checkpoints than SEND requests to non-TMF servers. Note that if the SEND request is outside of transaction mode to a TMF server that operates on nonaudited files, data might be lost because TMF is not invoked and the TCP performs fewer checkpoints.

TCP checkpointing requirements can be significantly reduced if PATHWAY applications running with TMF have TMF servers read outside of transaction mode before updating the data base.

The performance of a PATHWAY application can be improved by taking advantage of the TCP checkpointing strategy for TMF servers as follows:

- not using transaction mode for a server with read-only access to a data base (the access is retryable) where the requester displays the data before any attempt is made to change the data. In the event of a failure, the read operations are retryable and NonStop operation is maintained. (If you know that no other server has write access to the same data base, you can make the read-only operations nonretryable. Refer to the *Transaction Monitoring Facilitiy (TMF) System Management and Operations Guide*.)
- not using transaction mode for a server that writes to an entry-sequenced logging file in which possible duplicates are acceptable. In the event of a failure, the write operations can be rewritten.

PRECAUTIONS FOR USING TMF PARAMETERS

If a TMF error occurs and makes normal PATHWAY/TMF operation impossible, setting the TMF parameter options OFF is not the solution for continuing normal PATHWAY operations. The following can result:

1. The server intended for operation with TMF probably does not make the checkpoints necessary to function as a NonStop server when TMF is not invoked.
2. A SCREEN COBOL program that uses ABORT-TRANSACTION or RESTART-TRANSACTION to handle exceptions to normal program operation only appears to execute but the TMF verbs have no effect.
3. With SET SERVER TMF ON and SET TERM or SET PROGRAM TMF OFF, the TCP makes checkpoint, retry, and syncdepth decisions as if TMF were running. For example, the TCP performs fewer checkpoints and opens servers with a syncdepth of 0 instead of 1. The TCP will not be running in normal NonStop mode, and a single cpu failure can cause the application to fail.

APPENDIX F

GLOSSARY

Accept operation — An operation in which the program waits for response from the terminal and allows data to be input into the program data area from the terminal.

Advisory message — A message displayed in the terminal advisory field to inform the terminal operator of errors detected during input checking.

Alphabetic character — A character that belongs to the set of letters A through Z and the space character.

Alphanumeric character — Any character in the character set.

Application — A complete sequence of machine instructions and routines necessary to solve a problem.

Arithmetic expression — A combination of numeric elementary items and numeric literals connected by arithmetic operators to form an expression that reduces to a single value.

Arithmetic operator — A character that denotes an arithmetic operation: + for addition, – for subtraction, * for multiplication, and / for division.

Assignment — A convention in which an ASSIGN command is issued to make logical file assignments for programs. A logical assignment equates a Tandem file name with a logical file of a program and optionally attributes characteristics to that file.

Audited file — A data file that is flagged for auditing by TMF; auditing is the monitoring of transactions in preparation for recovery efforts.

Base screen — A screen that can be initially displayed on the terminal. In contrast to an overlay screen that is displayed in the overlay area of a base screen, the base screen can be displayed independently.

Block mode — A terminal operating mode in which data is read from the terminal and displayed on the terminal a screen at a time.

Character string — A series of contiguous characters.

Clause — An ordered set of characters that specify the characteristics of a field.

Command Interpreter — An interactive program used to run programs, check system status, create and delete disc files, and alter hardware states.

Comment line — A line consisting of any combination of characters from the character set for documentation purposes. Comment lines are indicated by an * or / in the indicator field of the reference format; a / causes a page eject before printing.

Compiler command line — An instruction for the SCREEN COBOL compiler; the line is indicated by a question mark in the indicator field.

Complex condition — A condition that has a truth value resulting from the interaction of all logical operators on the individual truth values of simple conditions, or on the intermediate truth values of conditions connected or negated.

Conditional expression — An expression that identifies a condition that is to be tested by the program for selection between alternate paths of control.

Condition-name — A level 88 data item with a value or range of values for testing purposes.

Condition variable — An item that immediately precedes a condition-name entry.

Conversational mode — A terminal operating mode in which data is read from the terminal and displayed on the terminal one line at a time.

Copy library — A library of SCREEN COBOL text that can be inserted into the source program by a COPY statement.

Data base — A collection of data that is described and controlled within a computer system.

Data base rollforward — A TMF activity in which the data base is restored to a consistent state after a catastrophic failure.

Data Division — The SCREEN COBOL source program division that defines the program data structures in terms of their formats and usage.

DDL — The Data Definition Language that is used to describe the records and files comprising a data base.

Declarative procedures — Screen recovery procedures specified by USE statements; procedures are declared in the Procedure Division immediately following the division header.

Default value — The value that is used by the system when a value has not been supplied by the user.

Diagnostic screen — A screen of information that is displayed to inform the terminal operator of error conditions and termination status.

Display attribute — A terminal display feature that is given a screen data name; the screen data name can be associated with a predefined system name in the SPECIAL-NAMES paragraph and thus be manipulated by a SCREEN COBOL program.

EDIT file — A source text file that can be augmented and modified by the user through the Tandem text editor.

Edited Item — An item whose PICTURE clause contains editing symbols.

Editing characters — The symbols that can be used in PICTURE clauses to format screen data.

ENCOMPASS — The Tandem distributed data base management system.

Environment Division — The SCREEN COBOL source program division that specifies the program execution environment.

External process — A PATHMON, TCP or server class that is running in a different PATHWAY system from the process with which it is communicating. For example, a TCP requests a link from an external PATHMON to an external server. The PATHMON and the server are in a different PATHWAY system from the TCP.

Field characteristic clause — An ordered set of characters that specify the characteristics of a screen field.

Figurative constant — A constant that has been prenamed and predefined by the SCREEN COBOL compiler.

Fill character — A character selected as the padding character of a field.

FILLER — A keyword that takes the place of a data name; FILLER items cannot be referenced.

Floating insertion characters — A string of at least two symbols, only one of which appears in an appropriate position in the final edited item. A single floating insertion character is placed in the character position immediately preceding the first nonblank character. The characters preceding the placement of the floating insertion character are replaced by spaces.

GUARDIAN — The Tandem operating system.

Identification Division — The SCREEN COBOL source program division that identifies the program.

Identifier — A data name made unique by qualification or subscripting.

INSPECT — An interactive program debugging tool that uses a table of symbolic names to access code and data locations in the executing program.

Integer numeric literal — A numeric literal that does not have a decimal point.

Interactive mode — An operating mode in which commands are entered from a terminal keyboard.

Keyword — A word in a command string that must be spelled and positioned in a prescribed way, usually to indicate the meaning of an adjacent parameter.

Level — A system of numbers that indicate either hierarchy or special properties of data items. Levels 01-49 describe hierarchy; level 66 specifies items introduced by a RENAMES clause; level 77 describes an independent data item that cannot be subdivided; level 88 defines a condition-name.

Linkage section — A SCREEN COBOL source program section that describes the structure of parameter data passed to a subprogam by a CALL statement.

Literal — A character string whose value is implied either by a set of characters or by a reserved word that represents a figurative constant.

Log files — Files that are used for reporting errors and changes in status.

Modified Data Tag (MDT) — A bit that is set or reset to indicate whether or not data in an associated field is to be sent to the computer from the terminal.

MUMPS — The Massachusetts General Hospital Utility Multi-Programming System interpretive programming language.

Noninteger numeric literal — A numeric literal that has a decimal point.

Noninteractive mode — An operating mode in which commands are entered through a command file.

Nonnumeric literal — An ASCII character string enclosed in quotation marks; a maximum of 120 characters, not including the surrounding quotation marks, is allowed.

Numeric character — A character that belongs to the set of digits 0 through 9.

Numeric literal — A string of one or more digits (0-9), a plus or minus sign, and an optional decimal point; a maximum of 18 digits is allowed.

Obey file — A file that serves as an alternate source for command input.

Overlay area — An area that defines an area of a base screen within which an overlay screen can be displayed.

Overlay screen — A screen that appears in the overlay area of a base screen.

Paragraph — A group of related sentences and statements.

PATHAID — A group of utility programs that are used to create and modify screen definitions.

PATHCOM command file — A file of commands that define PATHWAY entities required to execute an application.

PATHCOM process — The command interface to PATHMON.

PATHCTL — A disc file in which PATHMON maintains status information and the application configuration.

PATHMON — The central controlling process in a PATHWAY system.

PATHTCP — The TCP object module.

PATHWAY — A transaction processing system that supplies the programs, procedures, and structures necessary to produce user-written applications.

PATHWAY Monitor process — See PATHMON.

Phrase — An optional portion of a clause or statement.

POBJ — The default name of the SCREEN COBOL object program.

Procedure — A named grouping that can consist of a paragraph, a group of successive paragraphs, a section, or a group of successive sections.

Procedure Division — The SCREEN COBOL source program division that specifies the processing steps of the program.

Pseudo code — Code that is interpreted by software instead of being executed by the hardware.

Punctuation characters — Characters that are used to separate words, sentences, or special clauses, and to group arithmetic relationships.

Qualification — A convention that is used to make a name unique.

Reference format — The columnar positioning of source code.

Requester process — A process that interprets application program object code and sends replies to a server; synonymous with requester.

Reserved word — A word that can only be used as a keyword.

SCREEN COBOL — A procedural language that is used to define and control terminal displays.

SCREEN COBOL Utility Program (SCUP) — A utility that provides control and manipulation of SCREEN COBOL object files.

SCREEN COBOL word — A character string that forms a reserved word, user-defined word, or system name; a maximum of 30 characters is allowed.

Screen overlay area — Refer to overlay area.

Screen section — A SCREEN COBOL source program section that describes the types and locations of fields in screens that can be displayed on the terminal.

SCUP — See SCREEN COBOL Utility Program.

Section — A group of related paragraphs.

Send operation — An operation in which a transaction request message is sent to a server process and a reply is received back from the server process.

Sentence — A string of one or more statements.

Separator — A punctuation character that separates language elements.

Server class — A grouping of duplicate copies of a single server program; server processes within the class have identical attributes.

Server process — A process that implements application requests and sends replies to the requester; synonymous with server.

Simple condition — A condition that has a truth value of true of false; simple conditions are categorized as class, condition-name, relation, and sign conditions.

Special character — Any character in the character set except the letters A through Z, space, and digits 0 through 9.

SPECIAL-NAMES paragraph — An optional paragraph in the Environment Division of a SCREEN COBOL program; the paragraph allows user-selected names to be assigned to system names.

Special registers — Data items that are defined automatically by the SCREEN COBOL compiler.

Statement — A combination of words and symbols beginning with a SCREEN COBOL verb.

Subscripting — A convention that is used to reference individual elements in a table.

Symbol table — A table of symbols that identify program code and data locations. The table is built during compilation by the SYMSERV process and used by INSPECT for program debugging.

System name — 1) A SCREEN COBOL word that identifies part of the Tandem operating environment; a name can be associated with function keys or terminal display attributes.

2) A name given in the SCREEN COBOL SEND statement to identify the Tandem system on which a PATHWAY system is running.

Table — A set of repeated data items defined by an OCCURS clause.

TAL — The Tandem Transaction Application Language that is used to write systems software and routines that support transaction-oriented applications.

TCP — A Tandem-supplied program that interprets SCREEN COBOL object code and send messages to server processes; synonymous with requester process.

Terminal — A device capable of sending and receiving information over communication lines.

Terminal context — Data maintained by a TCP for each active terminal under its control.

Terminal Control Process — See TCP.

TMF — See Transaction Monitoring Facility.

Transaction — A basic unit of work that originates at a computer terminal and accesses data base files.

Transaction backout — A TMF activity in which the effects of a partially completed transaction are cancelled.

Transaction ID (TRANSID) — A unique transaction identifier that allows TMF to distinguish transactions when concurrent terminal programs are in transaction mode.

Transaction mode — The operational mode of a terminal between the execution of a BEGIN-TRANSACTION statement and the execution of the associated END-TRANSACTION statement or an ABORT-TRANSACTION statement.

Transaction Monitoring Facility (TMF) — A data management product that maintains the consistency of a data base and provides the tools for data base recovery.

TRANSID — See Transaction ID.

User conversion procedures — Procedures that are written by an installation to perform additional checking and conversion; procedures are called when the USER CONVERSION clause is declared for a field.

User-defined word — A word consisting of the letters A through Z, digits 0 through 9, and the hyphen character; the word must have at least one alphabetic character, must not begin or end with a hyphen, and must not contain embedded spaces.

Variable occurrence data item — A data item described with an OCCURS clause that includes a DEPENDING ON phrase.

Verb — A SCREEN COBOL leading keyword in a statement that identifies the purpose of that statement.

Working storage section — A SCREEN COBOL source program section that describes the structure of local data developed within the program.

INDEX

YOUR COMMENTS PLEASE

Tandem NonStop™ & NonStop II™ Systems
PATHWAY™ Programming Manual
82059 E00

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.

.D

.D

FROM:

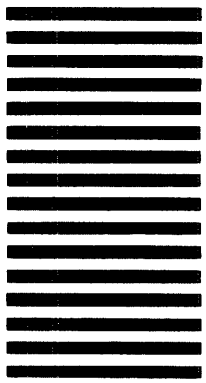
Name _____ Date _____

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—Technical Communications

TAPE