August 1978

This document describes how to write application programs and define transaction processors that run under TRAX. The structure of a transaction step task, the avaiable TRAX system calls, and the use of TRAX definition utilities are discussed.

This is a new manual.

TRAX
Application
Programmer's Guide

AA-D329A-TC

OPERATING SYSTEM AND VERSION: TRAX Version 1.0

To order additional copies of this document, contact the Software Distribution Center, Digital Equipment Corporation, Maynard, Massachusetts 01754

digital equipment corporation • maynard. massachusetts

First Printing, August 1978

The postage-prepaid READER'S COMMENTS form on the last page of this document requests the user's critical evaluation to assist us in preparing future documentation.

The following are trademarks of Digital Equipment Corporation:

DIGITAL	DECsystem-10	MASSBUS
DEC	DECtape	OMNIBUS
PDP	DIBOL	OS/8
DECUS	EDUSYSTEM	PHA
UNIBUS	FLIP CHIP	RSTS
COMPUTER LABS	FOCAL	RSX
COMTEX	INDAC	TYPESET-8
DDT	LAB-8	TYPESET-11
DECCOMM	DECSYSTEM-20	TMS-11
ASSIST-11	RTS-8	ITPS-10

CONTENTS

			Page
PREFACE			**xiii**
	0.1	MANUAL OBJECTIVES	xiii
	0.2	INTENDED AUDIENCE	xiii
	0.3	STRUCTURE OF THIS DOCUMENT	xiii
	0.4	ASSOCIATED DOCUMENTS	xiii
	0.5	CONVENTIONS USED IN THIS DOCUMENT	xiv
CHAPTER	**1**	**INTRODUCTION**	**1-1**
	1.1	TRAX APPLICATION PROGRAMS – TSTs	1-1
	1.2	A TYPICAL TRANSACTION	1-5
	1.2.1	Annotated Transaction Structure Diagram – CHGCUS	1-5
CHAPTER	**2**	**CODING A TRANSACTION STEP TASK**	**2-1**
	2.1	USING TSTs IN A TRAX APPLICATION	2-1
	2.2	TST IMPLEMENTATION CONSIDERATIONS	2-1
	2.3	THE TST DEVELOPMENT PROCESS	2-2
	2.4	TST STRUCTURAL REQUIREMENTS	2-2
	2.5	THE TST ENTRY POINT	2-3
	2.5.1	COBOL – Program Name and TST Parameters	2-3
	2.5.2	Coding the BASIC-PLUS-2 TST STatement	2-4
	2.6	THE EXCHANGE MESSAGE	2-4
	2.6.1	Coding the Exchange Message in COBOL	2-6
	2.6.2	Coding the BASIC-PLUS-2 MSGMAP Statements	2-6
	2.7	THE TRANSACTION WORKSPACE	2-6
	2.7.1	Coding the Transaction Workspace in COBOL	2-8
	2.7.2	Coding the BASIC-PLUS-2 WRKMAP Statements	2-8
	2.8	EXITING FROM A TST	2-9
	2.8.1	Normal Exit from a COBOL TST	2-10
	2.8.2	Normal Exit from a BASIC-PLUS-2 TST	2-10
CHAPTER	**3**	**TST FILE INPUT/OUTPUT OPERATIONS**	**3-1**
	3.1	APPLICATION DATA FILES	3-1
	3.2	TYPES OF DATA FILES	3-2
	3.3	FILE ACCESS METHODS	3-3
	3.4	LOGICAL DATA FILE NAMES	3-3
	3.5	PREPARING TO USE FILES FROM A TST	3-4
	3.6	CLOSING DATA FILES	3-5
	3.7	RECORD LOCKING FACILITIES	3-5
	3.8	READING RECORDS FROM FILES	3-7
	3.9	INSERTING RECORDS INTO FILES	3-7
	3.10	UPDATING EXISTING RECORDS TO A FILE	3-7
	3.11	DELETING RECORDS FROM A FILE	3-7

CONTENTS (Cont.)

		Page
3.12	I/O ERROR HANDLING	3-7
3.13	STAGED FILE OPERATIONS	3-7
3.14	EXAMPLES OF COBOL TST I/O OPERATIONS	3-9
3.14.1	Preparing for I/O in a COBOL TST	3-9
3.14.2	Using COBOL to READ Records from a File	3-11
3.14.3	Using COBOL to Write Records to a File	3-13
3.14.4	Using COBOL to Update Records on a File	3-13
3.14.5	Using COBOL to Delete Records from a File	3-14
3.14.6	I/O Error Handling in COBOL TSTs	3-14
3.14.7	Closing Data Files in COBOL TSTs	3-15
3.14.8	The UNLOCK and UNLOCK ALL verbs	3-15
3.14.9	Unsupported COBOL Syntax	3-15
3.15	EXAMPLE OF BASIC-PLUS-2 TST I/O OPERATIONS	3-16
3.15.1	Preparing for I/O Operations in BASIC-PLUS-2 TSTs	3-16
3.15.2	Reading Records from BASIC-PLUS-2 TSTs	3-16
3.15.3	Using BASIC-PLUS-2 to Put Records in a File	3-17
3.15.4	Using BASIC-PLUS-2 to Update Records on a File	3-17
3.15.5	Using BASIC-PLUS-2 to Delete Records from a File	3-17
3.15.6	I/O Error Handling in BASIC-PLUS-2 TSTs	3-17
3.15.7	The UNLOCK and FREE statements	3-18
3.15.8	Non-supported BASIC-PLUS-2 file syntax	3-18
CHAPTER 4	**CALLING THE TRAX SYSTEM LIBRARY FROM A TST**	**4-1**
4.1	TRAX SYSTEM LIBRARY ROUTINES	4-1
4.2	THE REPORT ROUTINE	4-2
4.2.1	Using The REPORT Routine from COBOL TSTs	4-2
4.2.2	Using the REPORT Routine from BASIC-PLUS-2 TSTs	4-4
4.2.3	Library Routine Status Return Codes	4-5
4.3	SENDING RESPONSE MESSAGES	4-5
4.3.1	The REPLY Routine	4-6
4.3.1.1	Using The REPLY Routine from COBOL TSTs:	4-6
4.3.1.2	Using the REPLY Routine from BASIC-PLUS-2 TSTs	4-7
4.3.1.3	Library Routine Status Return Codes	4-8
4.3.2	The ABORT Routine	4-8
4.3.2.1	Using the ABORT Routine from COBOL TSTs:	4-9
4.3.2.2	Using the ABORT Routine from BASIC-PLUS-2 TSTs	4-10
4.3.2.3	Library Routine Status Return Codes	4-11
4.3.3	The PRCEED Routine	4-11
4.3.3.1	Using the PRCEED Routine from COBOL TSTs	4-11
4.3.3.2	Using the PRCEED Routine from BASIC-PLUS-2 TSTs	4-12
4.3.3.3	Library Routine Status Return Codes	4-13
4.3.4	The STPRPT Routine	4-12
4.3.4.1	Using the STPRPT Routine from COBOL TSTs	4-13
4.3.4.2	Using the STPRPT Routine from BASIC-PLUS-2 TSTs	4-14

CONTENTS (Cont.)

		Page
4.3.4.3	Library Routine Status Return Codes	4-15
4.3.5	The CLSTRN Routine	4-15
4.3.5.1	Using the CLSTRN Routine from COBOL TSTs	4-15
4.3.5.2	Using the CLSTRN Routine from BASIC-PLUS-2 TSTs	4-16
4.3.5.3	Library Routine Status Return Codes	4-16
4.3.6	The TRNSFR Routine	4-17
4.3.6.1	Using the TRNSFR Routine from COBOL TSTs	4-17
4.3.6.2	Using the TRNSFR Routine from BASIC-PLUS-2 TSTs	4-18
4.3.6.3	Library Routine Status Return Codes	4-19
4.4	THE RESTRT ROUTINE – RESTARTING AN EXCHANGE	4-19
4.4.1	Using the RESTRT Routine from COBOL TSTs	4-19
4.4.2	Using the RESTRT Routine from BASIC-PLUS-2 TSTs	4-20
4.4.3	Library Routine Status Return Codes	4-20
4.5	THE TSPAWN ROUTINE – SPAWNING A TRANSACTION	4-20
4.5.1	Spawning a Transaction Instance	4-20
4.5.1.1	Using the TSPAWN Routine from COBOL TSTs	4-20
4.5.1.2	Using the TSPAWN Routine from BASIC-PLUS-2 TSTs	4-22
4.5.1.3	Library Routine Status Return Codes	4-22
4.5.2	The TABORT Routine	4-23
4.5.2.1	Using the TABORT Routine from COBOL TSTs	4-23
4.5.2.2	Using the TABORT Routine from BASIC-PLUS-2	4-24
4.5.2.3	Library Routine Status Return Codes	4-24
4.6	ROUTINE LIST CONTROL	4-24
4.6.1	The AROUTE Routine	4-25
4.6.1.1	Using the AROUTE routine from COBOL TSTs	4-25
4.6.1.2	Using the AROUTE routine from BASIC-PLUS-2 TSTs	4-25
4.6.1.3	Library Routine Status Return Codes	4-26
4.6.2	The DROUTE Routine	4-26
4.6.2.1	Using the DROUTE routine from COBOL TSTs	4-26
4.6.2.2	Using the DROUTE routine from BASIC-PLUS-2 TSTs	4-27
4.6.2.3	Library Routine Status Return Codes	4-27
4.6.3	The DALLRT Routine	4-27
4.6.3.1	Using the DALLRT routine from COBOL TSTs	4-27
4.6.3.2	Using the DALLRT routine from BASIC-PLUS-2 TSTs	4-28
4.6.3.3	Library Routine Status Return Codes	4-28
4.7	USING MAILBOX STATIONS FROM A TST	4-28
4.7.1	The SNDMBX Routine – Sending a Message to a Mailbox	4-29
4.7.1.1	Using the SNDMBX Routine from COBOL TSTs	4-29
4.7.1.2	Using the SNDMBX Routine from BASIC-PLUS-2 TSTs	4-30
4.7.1.3	Library Routine Status Return Codes	4-31
4.7.2	The GETMBX Routine	4-31
4.7.2.1	Using the GETMBX Routine from COBOL TSTs	4-31
4.7.2.2	Using the GETMBX Routine from BASIC-PLUS-2 TSTs	4-32
4.7.2.3	Library Routine Status Return Codes	4-33

CONTENTS (Cont.)

		Page
4.7.3	The MBXNUM Routine	4-34
4.7.3.1	Using the MBXNUM Routine from COBOL TSTs	4-34
4.7.3.2	Using the MBXNUM Routine from BASIC-PLUS-2 TSTs	4-35
4.7.3.3	Library Routine Status Return Codes	4-35
4.8	SYSTEM INFORMATION ROUTINES	4-36
4.8.1	The GETIME Routine	4-36
4.8.1.1	Using the GETIME routine from COBOL TSTs	4-36
4.8.1.2	Using the GETIME routine from BASIC-PLUS-2 TSTs	4-37
4.8.1.3	Library Routine Status Return Codes	4-38
4.8.2	Determining the Current TST Station ID	4-38
4.8.2.1	Using the GETSTN Routine from COBOL TSTs	4-38
4.8.2.2	Using the GETSTN routine from BASIC-PLUS-2 TSTs	4-39
4.8.2.3	Library Routine Status Return Codes	4-39
4.8.3	Determine the Initiating Station ID	4-40
4.8.3.1	Using the GETSRC Routine from COBOL TSTs	4-40
4.8.3.2	Using the GETSRC routine from BASIC-PLUS-2 TSTs	4-40
4.8.3.3	Library Routine Status Return Codes	4-40
4.8.4	Determine the Transaction Type	4-41
4.8.4.1	Using the GETRAN Routine from COBOL TSTs	4-41
4.8.4.2	Using the GETRAN routine from BASIC-PLUS-2 TSTs	4-42
4.8.4.3	Library Routine Status Return Codes	4-42
4.8.5	Determine a Physical File Specification	4-42
4.8.5.1	Using the GETFIL Routine from COBOL TSTs	4-43
4.8.5.2	Using the GETFIL routine from BASIC-PLUS-2 TSTs	4-44
4.8.5.3	Library Routine Status Return Codes	4-44
4.9	LOGGING INFORMATION TO THE JOURNAL FILE	4-45
4.9.1	The LOGTRN Routine – Log Specified Data	4-45
4.9.1.1	Using the LOGTRN Routine from COBOL TSTs;	4-45
4.9.1.2	Using the LOGTRN Routine from BASIC-PLUS-2 TSTs	4-46
4.9.1.3	Library Routine Status Return Codes	4-47
CHAPTER 5	**USING BATCH FACILITIES WITH A TRANSACTION PROCESSOR**	**5-1**
5.1	SUBMITTING A BATCH JOB FROM A TRANSACTION INSTANCE	5-1
5.2	INITIATING A TRANSACTION FROM A BATCH JOB	5-2
5.3	INITIATE TRANSACTION – THE STTRAN ROUTINE	5-2
5.4	USING THE STTRAN ROUTINE FROM COBOL PROGRAMS	5-3
5.5	USING THE STTRAN ROUTINE FROM BASIC-PLUS-2	5-4
5.6	LIBRARY ROUTINE STATUS RETURN CODES	5-5
CHAPTER 6	**COMMUNICATION BETWEEN TRANSACTION PROCESSORS**	**6-1**
6.1	TRAX/TL	6-1
6.1.1	Operations from a Master Link Stations	6-2
6.1.2	Preparing the Exchange Message for the Master Link	6-4
6.1.3	COBOL Master to Slave Message Format	6-4

CONTENTS (Cont.)

		Page
6.1.4	BASIC-PLUS-2 Master to Slave Message Format	6-5
6.1.5	Slave Link Station	6-6
6.1.5.1	Response messages Sent to the Slave Link Station	6-6
6.2	TRAX/3271-TL	6-9
6.2.1	Master Link Stations	6-10
6.2.2	Preparing the Exchange Message for the Master Link	6-10
6.2.3	COBOL Master to IBM Message Format	6-11
6.2.4	BASIC-PLUS-2 Master to IBM Message Format	6-12
6.2.4.1	Handling Responses from IBM Systems	6-13
CHAPTER 7	**TST DEBUGGING AND TESTING FACILITIES**	**7-1**
7.1	COMPILING TSTs	7-1
7.1.1	Compiling a COBOL TST	7-1
7.1.2	Compiling a BASIC-PLUS-2 TST	7-1
7.2	LINKING TSTs – THE TSTBLD UTILITY	7-2
7.2.1	Examples of TSTBLD Usage	7-5
7.3	TST DEBUGGING IN THE SUPPORT ENVIRONMENT	7-7
7.3.1	DEBUG – The TST Debugging Utility	7-7
7.3.2	Using the DEBUG Utility	7-7
CHAPTER 8	**USING THE DEFINITION UTILITIES**	**8-1**
8.1	UTILITY DESCRIPTIONS	8-1
8.2	TRAX UTILITY DIALOG CONVENTIONS	8-2
CHAPTER 9	**TRANSACTION PROCESSOR DEFINITION: THE TPDEF UTILITY**	**9-1**
9.1	TRANSACTION PROCESSOR DATA STRUCTURES	9-1
9.1.1	The Transaction Processor File Record	9-1
9.1.2	The Terminal Management Task and Common Data Area	9-1
9.1.3	The Definition Data Files	9-2
9.2	THE TPDEF UTILITY	9-3
9.2.1	Invoking the TPDEF Utility	9-3
9.2.2	Creating or Editing a TP Definition	9-4
9.2.3	Listing the INDEX of Defined TP	9-8
9.2.4	Printing or Showing a TP Definition	9-11
9.2.5	Copying or Renaming an Existing TP	9-12
9.2.6	Deleting A TP Definition	9-14
CHAPTER 10	**STATION DEFINITION**	**10-1**
10.1	STATION CONCEPTS	10-1
10.2	THE STADEF UTILITY	10-1
10.2.1	Invoking the STADEF Utility	10-1
10.2.2	Adding or Editing a Station Definition	10-2
10.2.2.1	Terminal Stations	10-3

CONTENTS (Cont.)

		Page
10.2.2.2	TST Stations	10-7
10.2.2.3	Master Link Stations	10-11
10.2.2.4	Slave Link Stations	10-13
10.2.2.5	Submit Batch Station	10-13
10.2.2.6	Slave Batch Stations	10-13
10.2.2.7	Mailbox Stations	10-13
10.2.3	Listing the INDEX of Defined Stations	10-15
10.2.4	Printing or Showing a Station Definition	10-16
10.2.5	Deleting a Station Definition	10-18
CHAPTER 11	**TRANSACTION DEFINITION**	**11-1**
11.1	TRANSACTION CONCEPTS	11-1
11.1.1	The TRADEF Utility	11-1
11.1.2	Invoking the TRADEF Utility	11-1
11.1.3	Adding or Editing a Transaction Definition	11-2
11.1.3.1	Exchange Definition Parameters	11-4
11.1.3.2	Subsequent Action Parameters	11-7
11.1.4	Listing the INDEX of Defined Transactions	11-11
11.1.5	Printing or Showing a Transaction Definition	11-11
11.1.6	Deleting a Transaction Definition	11-12
CHAPTER 12	**APPLICATION DATA FILE DEFINITION**	**12-1**
12.1	FILE CONCEPTS	12-1
12.2	THE FILDEF UTILITY	12-1
12.3	INVOKING THE FILDEF UTILITY	12-2
12.3.1	Adding or Modifying a File Definition Record	12-3
12.3.2	Listing the INDEX of File Definitions	12-8
12.3.3	Printing or Showing a File Definition Record	12-9
12.3.4	Deleting a File Definition Record	12-10
CHAPTER 13	**APPLICATION SECURITY TOOLS**	**13-1**
13.1	WORK CLASSES	13-1
13.1.1	The WORDEF Utility	13-1
13.1.1.1	Invoking the WORDEF Utility	13-1
13.1.1.2	Adding or Modifying a Work Class Definition	13-2
13.1.1.3	Listing the INDEX of Work Class Definitions	13-4
13.1.1.4	Printing or Displaying Work Class Definitions	13-5
13.1.1.5	Deleting a Work Class Definition	13-5
13.2	USER AUTHORIZATIONS	13-7
13.2.1	The AUTDEF Utility	13-7
13.2.1.1	Invoking the AUTDEF Utility	13-7
13.2.1.2	Adding or Editing a User Authorization	13-8
13.2.1.3	Listing the INDEX of User Authorizations	13-11
13.2.1.4	Printing or Showing a User Authorization	13-11

CONTENTS (Cont.)

		Page
13-2	WORDEF Terminal Dialog Listing for INDEX Command	13-5
13-3	WORDEF Terminal Dialog Listing for SHOW Command	13-6
13-4	WORDEF Terminal Dialog Listing for DELETE Command	13-6
13-5a	User Authorization Specification Sheet for "SAMPLE"	13-9
13-5b	AUTDEF Terminal Dialog Listing for ADD Command	13-10
13-6	AUTDEF Utility Listing of INDEX Command Terminal Dialog	13-11
13-7	AUTDEF Utility Listing of PRINT Command Terminal Dialog	13-12
13-8	AUTDEF Utility Listing of DELETE Command Terminal Dialog	13-12
13-9	Transaction Processor Specification Sheet	13-13
13-10a	Transaction Specification for "SIGNOF"	13-14
13-10a	Transaction Specification for "SIGNON"	13-15
13-12a	Work Class File Definition Specification	13-16
13-11a	TST station Specification for "SIGNON" and "SIGNOF"	13-17
13-11b	Terminal Station Specification with SIGNON Work Class	13-17
13-12b	User Authorization File Definition Specification	13-18
13-13	ATL Utility Dialog to Add SIGNON and SIGNOF forms	13-19
13-14a	Initial Display of SIGNON Form	13-20
13-14b	SIGNON Form after ID and password are typed	13-21
13-15	Initial Display of SIGNOF Form	13-22
14-1	SERCTL Utility Dialog to Enable Error Logging	14-2
14-2	TPCTRL Terminal Dialog for the INSTALL Command	14-3
14-3	TPCTRL Terminal Dialog to START "SAMPLE"	14-4
14-4	SERLOG Output Listing	14-5
14-5	TPCTRL Terminal Dialog to STOP and REMOVE "SAMPLE"	14-6
14-6	TPTRAC Dialog for SAMPLE	14-6
14-7	TPTRAC Annotated Output Listing	14-9
A-1	Parameter List for "REPLY" Library routine	A-2

CONTENTS (Cont.)

			Page
FIGURE	1-1	TST Data Flow	1-3
	1-2	Transaction Structure Diagram for "CHGCUS"	1-4
	2-1	Exchange Message Specification Sheet	2-5
	2-2	Transaction Workspace Specification Sheet	2-7
	3-1	Customer File Definition Sheet	3-8
	3-2	Record Layout Sheet Describing Customer Record	3-12
	7-1a	TST Specification Sheet for COBOL TST RECUST	7-5
	7-1b	TSTBLD Dialog to Build "RDCUST" for Debugging	7-5
	7-2a	TST Specification Sheet for BASIC TST RDCUST	7-6
	7-2b	TSTBLD Dialog to Build "RDCUST" for Debugging	7-6
	7-3a	DEBUG Utility Output for COBOL "RDCUST" TST	7-11
	9-1a	Transaction Processor Specification Sheet for "SAMPLE"	9-9
	9-1b	Terminal Listing of CREATE Command Dialog	9-10
	9-2	Terminal Listing of INDEX Command Dialog	9-11
	9-3	Terminal Listing of SHOW Command Dialog	9-12
	9-4	TPDEF Utility COPY Dialog Listing	9-13
	9-5	Terminal Listing of DELETE Command Dialog	9-14
	10-1a	Terminal Station Specification Sheet – "SAMPLE"	10-6
	10-1b	Listing of Terminal Dialog to Add a Terminal Station	10-7
	10-2a	TST Station Specification Sheet – "SAMPLE"	10-9
	10-2b	Listing of Terminal Dialog to Add a TST Station	10-10
	10-3a	Master Link Station Specification Sheet – "SAMPLE"	10-12
	10-3b	Listing of Terminal Dialog to Add a Master Link Station	10-12
	10-4a	Special Purpose Station Specification Sheet – "SAMPLE"	10-14
	10-4b	Listing of Terminal Dialog to Add a Mailbox Station	10-14
	10-5	STADEF Utility Listing of INDEX Command Terminal Dialog	10-15
	10-6	STADEF Utility Listing of PRINT Command Terminal Dialog	10-16
	10-7	STADEF Utility Listing of DELETE Command Terminal Dialog	10-18
	11-1	System Workspace Worksheet for "CHGCUS"	11-5
	11-2a	Transaction Specification Sheet for "CHGCUS"	11-9
	11-2b	TRADEF Utility Listing of Add Command Terminal Dialog	11-10
	11-3	TRADEF Utility Listing of INDEX Command Terminal Dialog	11-12
	11-4	TRADEF Utility Listing of Print Command Terminal Dialog	11-13
	11-5	TRADEF Utility Listing of DELETE Command Terminal Dialog	11-13
	12-1a	File Definition Specification for "CUSTOM"	12-7
	12-2	FILDEF Utility Listing of INDEX Command Dialog	12-8
	12-1b	FILDEF Utility Listing ADD Command Terminal Dialog	12-9
	12-3	FILDEF Utility Listing of SHOW Command Dialog	12-10
	12-4	FILDEF Utility Listing of DELETE Command Dialog	12-11
	13-1a	Work Class Specification Sheet for "SAMPLE"	13-3
	13-1b	WORDEF Terminal Dialog Listing for ADD Command	13-4

CONTENTS (Cont.)

		Page
13.2.1.5	Deleting a User Authorization	13-12
13.2.2	The SIGNON and SIGNOF Transactions	13-12
13.2.2.1	Incorporating the SIGNON/SIGNOF Transactions	13-13
13.2.2.2	Using the SIGNON and SIGNOF Transactions	13-20
CHAPTER 14	**TRANSACTION PROCESSOR TESTING ENVIRONMENT**	**14-1**
14.1	INSTALLING AND TESTING A TRANSACTION PROCESSOR	14-1
14.2	DEBUGGING IN THE TRANSACTION PROCESSING ENVIRONMENT	14-1
14.3	THE SOFTWARE ERROR LOGGING TASK – SERLOG	14-1
14.4	USING THE TPCTRL UTILITY	14-2
14.5	TESTING THE TRANSACTION	14-4
14.6	USING THE SECONDARY ERROR LOG LISTINGS	14-5
14.7	STOPPING A TRANSACTION PROCESSOR	14-7
14.8	TRANSACTION PROCESSOR TRACE UTILITY – TPTRAC	14-7
14.9	ANNOTATED TPTRAC OUTPUT LISTING	14-9
APPENDIX A	**MACRO PROGRAMMING NOTES**	**A-1**
A.1	WRITING A MACRO TST	A-1
A.1.1	MACRO Entry Point	A-1
A.1.2	Using the TRAX system library from a MACRO TST	A-2
APPENDIX B	**COBOL TST EXAMPLE – TDCUST**	**B-1**
APPENDIX C	**BASIC TST EXAMPLE – RDCUST**	**C-1**

PREFACE

0.1 MANUAL OBJECTIVES

This manual is both a tutorial and a reference document. Useful techniques for writing TRAX application programs are described. Debugging and testing techniques for TRAX applications are discussed, with examples. This manual also gives detailed reference information about how to use the TRAX system library routines, and the TRAX transaction processor definition utilities.

0.2 INTENDED AUDIENCE

This document is written for use by the TRAX application programmer. It assumes prior knowledge of either BASIC-PLUS-2 or COBOL. To use the material in this manual, you must also have an understanding of TRAX application design considerations.

0.3 STRUCTURE OF THIS DOCUMENT

This manual is divided into chapters and appendices. The following list gives a general description of each section:

Chapter

1. Introduces key concepts and facilities.
2. Describes structural requirements for TSTs.
3. Discusses file I/O operations from TSTs.
4. Describes how to use the TRAX system library routines.
5. Describes how to use batch processing with a transaction processor.
6. Describes the link facilities used for inter-processor communication.
7. Describes how to compile, build, and debug a TST.
8. Introduces the Transaction Processor Definition Utilities.
9. Describes how to use the TPDEF utility to define a transaction processor.
10. Describes how to use the STADEF utility to define stations.
11. Describes how to use the TRADEF utility to define transaction types.
12. Describes how to use the FILDEF utility to define application data files.
13. Describes TRAX application security facilities.
14. Describes the transaction testing and debugging environment.

A. Gives you information about writing TSTs in MACRO.

B. Shows the COBOL TST RDCUST from the TRAX Sample Application.

C. Shows the BASIC TST's RDCUST.

0.4 ASSOCIATED DOCUMENTS

Before reading this document, you must read the *Introduction to TRAX.*

If you are not familiar with the program development facilities supported under TRAX, you should read the *TRAX Support Environment User's Guide* and the *DEC Editor Reference Manual.*

If you are unfamiliar with TRAX application design techniques, you may find it helpful to read the *TRAX Application Designer's Guide.*

Several programming reference documents are supplied with the TRAX documentation set. You will find detailed information on how to use the entire programming language in these manuals:

TRAX COBOL Language Reference Manual
TRAX COBOL User's Guide
TRAX BASIC-PLUS-2 Language Reference Manual
TRAX BASIC-PLUS-2 User's Guide

Finally the TRAX System Manager's Guide has useful information concerning system backup and maintenance, as well as an extensive listing of system error messages.

0.5 CONVENTION USED IN THIS DOCUMENT

Throughout this manual, the term TST is used to refer to a transaction step task, which is a TRAX application program. TP is sometimes used in place of the term transaction processor. The following conventions and used in examples:

CTRL/Z	The CTRL key and another key pressed simultaneously (e.g., CTRL/Z)
RET	The RETURN key (carriage-return/line feed).
ESC	The ESCAPE key.
Red text	Where examples contain both user input and computer output, the characters you type are in red; the characters the computer prints are in black

CHAPTER 1
INTRODUCTION

Application programming at a TRAX installation consists of three primary functions:

- Writing Transaction Step Tasks (TST).
- Running the transaction processor definition utilities.
- Debugging, integrating, and testing TSTs as part of the transaction processor.

This manual contains specific information to assist you in performing these tasks. This chapter contains a description of what a TST can do, and introduces a transaction design to acquaint you with the type of processing performed in a TRAX system.

Before you begin to use this manual, you should be comfortable with the following terminology used to describe transaction processing at a TRAX installation:

Transaction processor
Transaction
Transaction instance
Exchange
Exchange message
Transaction workspace
Transaction slot
Response message
Station

If any of these terms are unfamiliar, you should refer to the *Introduction to TRAX* or the *TRAX Application Designer's Guide.*

1.1 TRAX APPLICATION PROGRAMS – TSTs

A Transaction Step Task (TST) is a TRAX application program. You write a TST in COBOL, BASIC-PLUS-2, or MACRO. The TST is responsible for the application related processing in a transaction processor. It typically performs functions such as data base inquiry and update, input validation, and mathematical calculations.

A TST is incorporated into a transaction processor through a TST station. The TST station is the system software module that serves as the interface between the TRAX executive and the task image constructed from your TST source statements. Each TST task image has a defined TST station specified through the STADEF utility dialog.

The TST closely resembles a subroutine. The transaction processor invokes the TST in the same fashion that a mainline program calls a subroutine. During the call, the transaction processor passes two parameters to the TST. These parameters are the exchange message, which contains the user

input, and the transaction workspace, which provides context for TSTs in the same transaction instance.

The order in which TSTs process an exchange message is dependent upon the transaction definition. Each TST applies a specified set of processing steps to an arriving exchange message.

A TST can perform any or all of the following actions:

- Validate user input contained in the exchange message.
- Alter data in the exchange message or transaction workspace.
- Add records, update records, or retrieve records in permanent data files.
- Create, update, and read from work (temporary) files.
- Store and retrieve messages at mailbox stations.
- Perform logical and arithmetic operation on data supplied to the TST.
- Send data (report messages) to be printed on a hard-copy output-only terminal.
- Send data (response messages) to the initiating station at the conclusion of an exchange.
- Control the processing sequence by altering the exchange routing list, or the subsequent action of the current exchange.
- Call a system routine to restart the current exchange.
- Construct an exchange message and cause a batch job to be submitted.
- Initiate (spawn) new transaction instances. These spawned transactions are independent and asynchronous from the processing of the current exchange.
- Abort the current, or a spawned transaction instance.
- Call system library routines to access system information such as time and data, station, file, and transaction ID's.
- Log selected data to the system journalling device.

Chapter 2 describes how to set up a TST, and access the data in the exchange message and transaction workspace.

Chapter 3 describes file I/O operations as they apply to TSTs.

Chapter 4 describes the set of TRAX system library routines available to TSTs.

Figure 1-1 is a graphic description of the data that is accessed, modified, and created by a TST.

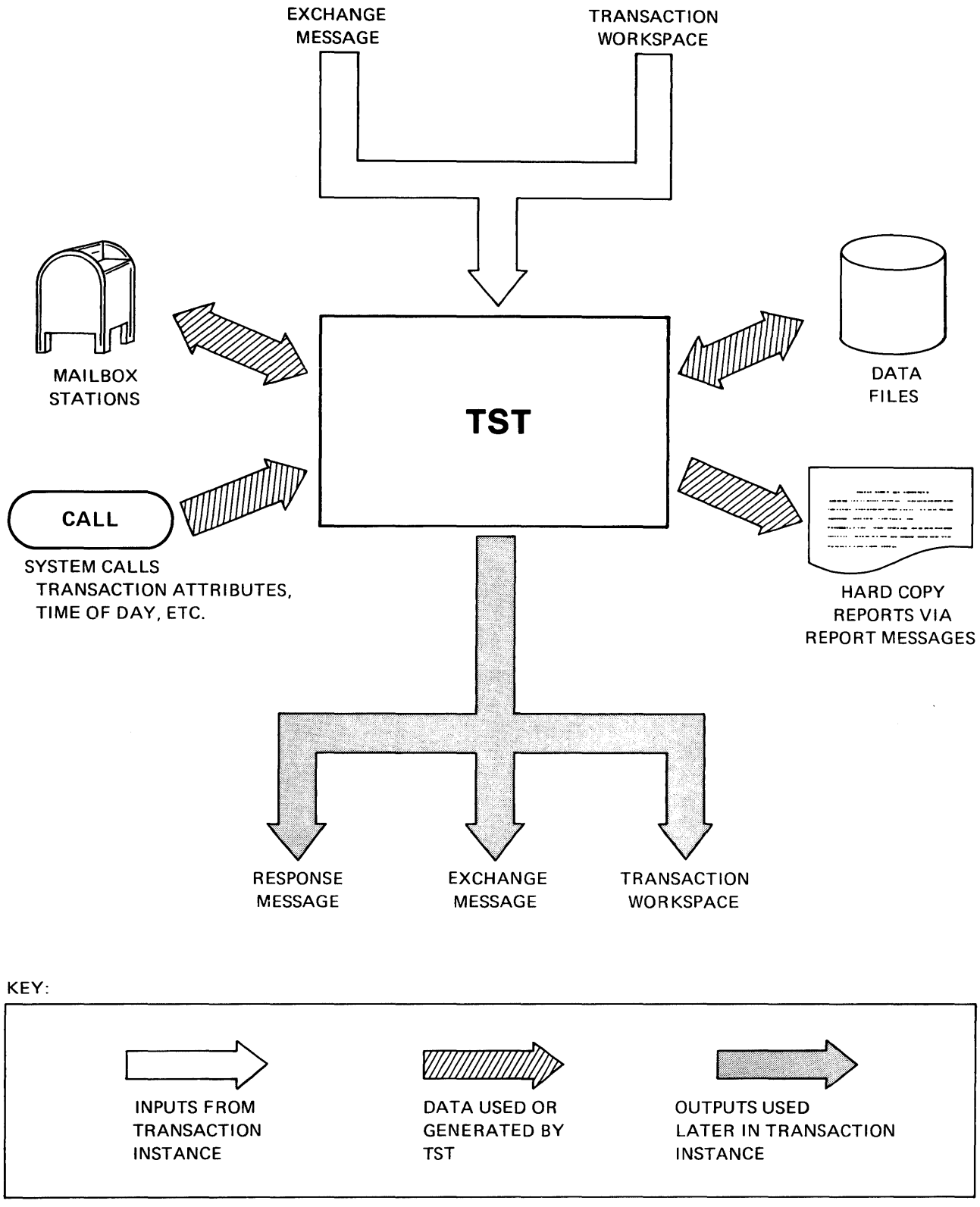


Figure 1-1 TST Data Flow

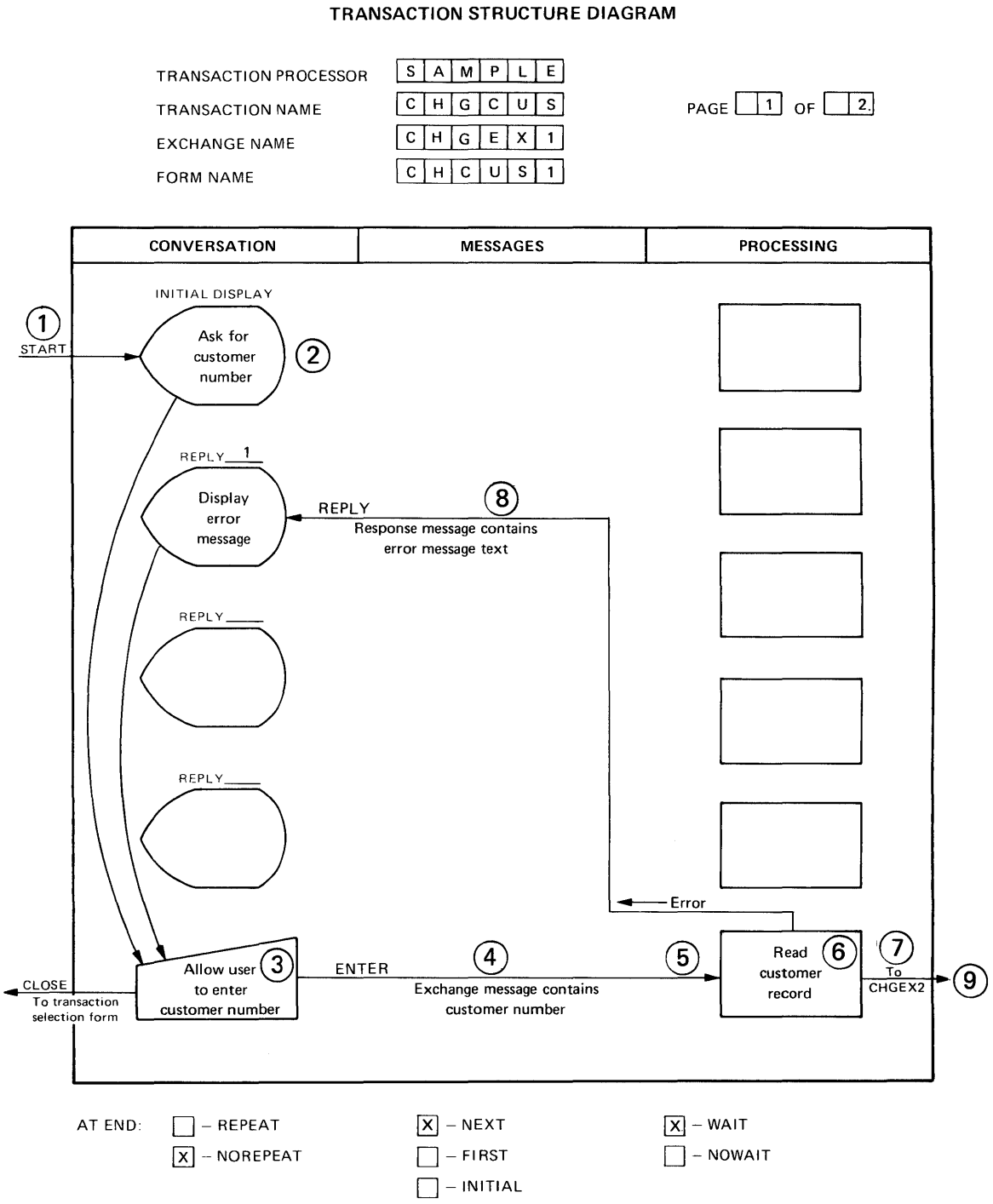


Figure 1-2 Transaction Structure Diagram for "CHGCUS"

1.2 A TYPICAL TRANSACTION

A set of transactions have been designed to illustrate the examples in this manual. To more clearly understand the application programming process, look first at the following transaction design used to change customer data. Figure 1-2 is a transaction structure diagram prepared by the application designer. It shows the sequence of processing and flow of data through this transaction. The numbers on the diagram correspond to the numbered notes in the following section.

1.2.1 Notes on the Transaction Structure Diagram for CHGCUS

1. The Transaction Begins. The user has selected this transaction on a transaction selection form.
2. Displaying the Form for the First Exchange. The first exchange begins with the display of a form. This form is specified in the transaction definition and is found in the form definition file.
3. First User Input. The user enters the customer ID on the form, and transmits the data to the system by pressing the ENTER key.
4. Exchange Message Constructed. When the data from the terminal arrives at the transaction processor, it is used to construct an exchange message according to the specifications in the form definition.
5. Exchange Message Routing. The transaction definition includes a routing list for each exchange. This list specifies in order the TST stations where the exchange message routed. In this example, only one TST, RDCUST, is on the routing list.
6. Processing by the RDCUST TST. The exchange message is routed to the TST station RDCUST. The RDCUST TST is invoked and the exchange message and transaction workspace are passed to it. RDCUST reads the customer record specified by the user input contained in the exchange message.
7. Response Message Sent. When the record is read successfully, the TST calls a system library routine to send a response message containing customer record data. The response message is sent to the initiating station and is a "PRCEED" type directing the station to go on to the next exchange.
8. Error Reply Sent. If the TST cannot read the record, it sends a reply type response message to the initiating station. In some cases, this reply allows the user to reenter the first exchange. In the case of severe errors, the reply causes the transaction to be aborted. The application programmer is responsible for selecting the type of response message.
9. The Second Exchange. When the terminal station receives the PRCEED messages from RDCUST, it enters the second exchange of CHGCUS.

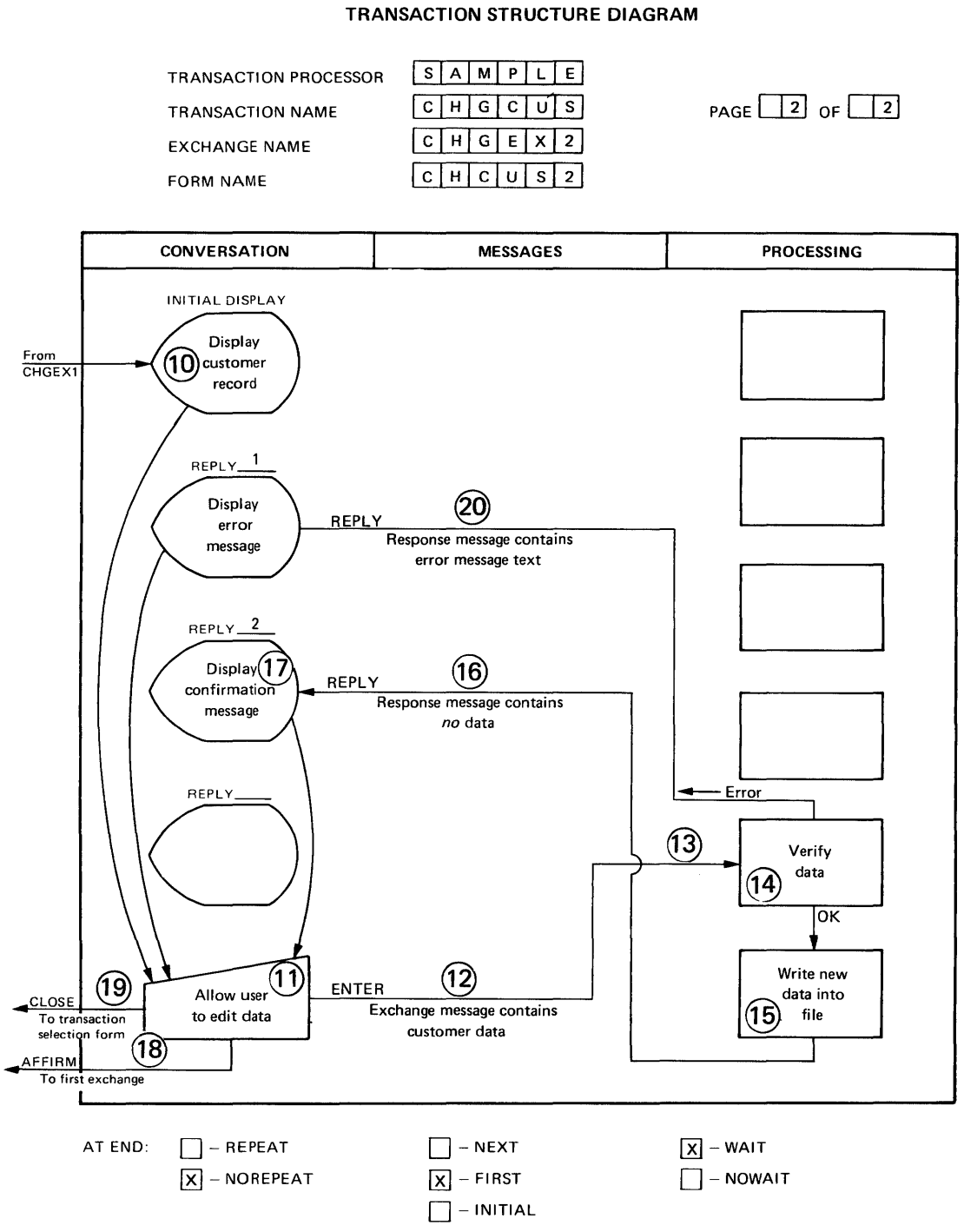


Figure 1-2 (cont.) Transaction Structure Diagram for "CHGCUS"

10. Display Response Message Data. The response message data is displayed on the form for the second exchange according to the specifications in the form definition. The user is asked to edit the customer data.
11. Second Exchange User Input. The user reads the information on the form, and edits the data as required. If the data is correct, the user need not enter any data.
12. Exchange Message Constructed. When the user presses the ENTER key, the data is formatted into an exchange message according to the specifications on the form definition. All customer data is included in the form definition, not just the data changed by the user.
13. Exchange Message Routing. The second exchange has two TSTs on the routing list. The first TST, VALIDC, validates the user input. The second TST, REWRIT, updates the record on the customer file.
14. Processing by VALIDC TST. This TST examines the user input data, and sends error replies if information is missing or incomplete.
15. Processing by REWRIT TST. If VALIDC terminates normally, the exchange message is passed to the REWRIT TST station. REWRIT updates the record content on the customer file. If errors are encountered, a reply response message is sent to the initiating station. If successful, a success reply is sent to the initiating station.
16. Generation of Response Message. This response message contains no data. It does contain a parameter value instructing the terminal station to display reply screen 1.
17. Response Message Arrives at Terminal Station. The response message causes a reply screen containing the notation ★★TRANSACTION COMPLETE★★ to be displayed on the terminal screen.
18. User presses AFFIRM Key. When the confirmation message is received, pressing the AFFIRM key ends the transaction instance and returns the terminal screen to the first form of the CHGCUS transaction.
19. User return to Transaction Selection Screen. If the user wants to return to a transaction selection menu, he presses the CLOSE key.
20. Error Replies. If VALIDC or RDCUST has sent an error reply, the cause of the error is displayed on the screen. If the error is recoverable, the user can modify the customer data as required and reenter the second exchange. Non-recoverable errors force the user to abort the transaction instance.

In addition to a transaction structure diagram, the application programmer should normally receive the following information from the application designer:

1. A copy of the transaction definition specification sheet.
2. A transaction workspace specification sheet for the transaction.
3. A copy of the record layout for each file accessed by the transaction.
4. A file definition specification sheet for each file.
5. An exchange message specification sheet for each exchange.
6. Response message specification sheets for each response message and error reply sent in the transition.
7. A TST specification sheet, and TST station specification sheet for each TST used in the transaction.
8. Mailbox and Report message specifications required by the transaction.

The specification sheets used to illustrate Chapters 2, 3, and 4 are drawn from the implementation of CHGCUS used in the TRAX Sample Application. These specifications are developed in the *TRAX Application Guide.*

CHAPTER 2
CODING A TRANSACTION STEP TASK

2.1 USING TSTs IN A TRAX APPLICATION

A TST is an application program that performs a single data processing operation in the execution of a transaction instance. For programming purposes, a TST is a special type of subroutine that is called by a transaction processor. Writing TSTs is the major procedural programming effort involved in the development of a transaction processing application.

Every TST has an associated TST station. When a terminal user generates data that needs to be processed by a specific TST, the transaction processor sends that data (called an exchange message) to the TSTs station. The presence of the exchange message at the station causes a copy of the TST to be brought into memory and executed. The TST then processes the exchange message and exits. The TST is not run again until another exchange message arrives at its station.

The exchange routing list directs the processing of the exchange message. This list is defined for each exchange in a transaction as part of the transaction definition.

Exchange messages are sent to stations in the order that the stations are specified in the exchange's routing list. When one TST terminates, the processor sends the exchange message to the next station in the routing list of the current exchange. A TST may alter the contents of the exchange message.

2.2 TST IMPLEMENTATION CONSIDERATIONS

In a transaction processing environment, many data processing functions are repeated in different places within the overall framework of an application. Careful design allows the application programmer to use source language libraries to reduce the number of individual statements that must be coded.

For example, a data structure for a customer record remains the same whether it is used for adding, deleting, updating or displaying the information. Storing this data structure in a separate file will allow you to include in the TSTs you are coding without having to retype each of the source statements.

A recommended technique to improve productivity is to use skeleton source files for your TSTs. A skeleton file contains general headings pertinent to TSTs. The skeleton source files can be used to construct TST files by using the DEC Editor, or by using the COBOL COPY verb or the BASIC-PLUS-2 APPEND command.

NOTE

While using the same form or TSTs for different applications may appear attractive during the design phase, experience has shown that coding each form and TST for a specific transaction type makes the transaction easier to debug and checkout, and results in a more efficient custom-tailored system.

2.3 THE TST DEVELOPMENT PROCESS

The stages in the development of a TST are:

1. Analyze the TST specification received from the application designer. The specification is a descriptive plan that you can convert into a set of source language statements.
2. Code the TST in the programming language you have selected, and create a source file using the DEC Editor (see the *DEC Editor Reference Manual*).
3. Compile your source file using the appropriate language compiler. To compile a TST source statement file, you must specify the /TST command switch at the time you invoke the compiler. (Compiler specific information is available in the *TRAX Language Reference Manual and Users Guide* for each of the supported source languages.)
4. Using the TSTBLD utility, link the resulting object module into a task image file suitable for use with the DEBUG utility.
5. Debug the TST using the DEBUG utility in the support environment.
6. Using the TSTBLD utility, link the object module into a task image file suitable for transaction processing debugging. This task image should include support for a debugging terminal.
7. Install a transaction processor that uses the TST. Debug the TST using the debugging terminal, the software error log, and the transaction processor trace facility.
8. When debugging and testing is complete, run the TSTBLD utility to relink the object module into a production version of the TST.

2.4 TST STRUCTURAL REQUIREMENTS

This section describes the structural elements of a TST, and suggests how you should code them in both COBOL and BASIC-PLUS-2.

A TST closely resembles a subroutine. If you think of the transaction processor as a main program, a TST is simply a subroutine called and executed by the transaction processor after an exchange message has been received by the station associated with the TST.

A TST uses two arguments supplied to it by the transaction processor, the exchange message, and the transaction workspace.

The exchange message is created by the initiator of an exchange. The exchange message contains the external data that is processed by an exchange defined in a transaction definition. Each exchange has its own unique exchange message, which is routed to the TST stations in the order specified by the routing list for that exchange. An exchange message must exist in every transaction instance.

The TST workspace is an area of storage provided by the system where a TST can store data for processing by subsequent exchanges in the same transaction instance. The transaction workspace is optional. If a transaction workspace is not present, the TST should not refer to a transaction workspace argument.

2.5 THE TST ENTRY POINT

When the transaction processor sends an exchange message to a TST station, the TST associated with that station begins execution. The transaction processor supplies the TST, as a subroutine, with two arguments, the exchange message and the transaction workspace. The method of accessing these arguments is different for each language.

2.5.1 Specifying the Program Name and TST Parameters in COBOL

A COBOL TST requires that you specify TSTEP as the PROGRAM-ID in the IDENTIFICATION DIVISION. Since a TST references two common data structures, the exchange message and transaction workspace, you must define these areas in a LINKAGE SECTION in the DATA DIVISION. The TST accesses these structures through the USING clause of the PROCEDURE DIVISION header where you specify the data names of the exchange message and transaction workspace.

The following COBOL examples are taken from the TST "RDCUST":

In the IDENTIFICATION DIVISION:

PROGRAM-ID. TSTEP.	TRAX requires that a TST always have the program name TSTEP.

In the DATA DIVISION:

LINKAGE SECTION.	The LINKAGE SECTION denotes storage that is not physically in the program, but in a common area that is referenced by subroutines.
01 EXCHANGE-MESSAGE.	The group data item name of the data structure describing the contents of the exchange message. The exchange message structure must be defined in the Linkage Section. See Section 2.6.1 for further information on coding the exchange message.
01 TRANSACTION-WORKSPACE.	The group data item name of the data structure describing the contents of the transaction workspace. If a workspace has been defined for the transaction type, you must define the workspace data structure in the Linkage Section. If no workspace parameter is specified in the PROCEDURE DIVISION header, you do not need to define a workspace data structure in the LINKAGE SECTION. See Section 2.7.1 for further information on coding the transaction workspace.

In the PROCEDURE DIVISION, the first statement must be the following:

```
PROCEDURE DIVISION USING EXCHANGE-MESSAGE, TRANSACTION-WORKSPACE.
```

The parameters specified in the USING clause of the PROCEDURE DIVISION header are the date names defined for the exchange message and transaction workspace in the LINKAGE SECTION of the DATA DIVISION. The data structures following those data names reference data stored in the system common data areas.

2.5.2 Coding the BASIC-PLUS-2 TST Statement

In BASIC-PLUS-2, the entry point must be the first statement in the TST. This statement must be a TST statement. The suggested form of a TST statement used in the entry point is:

```
1 TST TSTEP (argument1, argument2)
```

In this example:

1	is the statement number.
TST	is a BASIC-PLUS-2 keyword used to identify this program as a TST.
TSTEP	is the program name required for all TSTs.
argument1	is a dummy parameter used by the BASIC-PLUS-2 TST to map the data structure specified in the MSGMAP statement onto the exchange message supplied by the transaction processor. This data structure may have any name you choose, but must be a string type variable.
argument2	is a dummy parameter used by the TST to map the data structure specified in the WRKMAP statement on to the transaction workspace supplied by the transaction processor. This data structure may have any name you choose, but must be a string-type variable. If a transaction is defined without a transaction workspace, this parameter should be omitted.

2.6 THE EXCHANGE MESSAGE

The exchange message is a TRAX data structure that allows TSTs to pass information from the initiator of a transaction instance to other stations defined as part of the same exchange in a transaction instance. In your TST source statements, you must define a data structure that describes the exchange message. The exact method of defining the exchange message varies according to the source language that you select for your TSTs.

The next two sections describe language specific considerations to be followed in coding the exchange message data structure.

When you specify the exchange message size in the transaction definition, the value you supply is rounded up to the next highest multiple of 64. For example, if you specify the exchange message area as 96 bytes, the transaction definition allocates 128 bytes for the exchange message area.

The combined space allocations of the exchange message and transaction workspace cannot exceed 8064 bytes.

The form in Figure 2-1 is a specification sheet prepared by the application designer for the exchange message in the first exchange of the Change Customer Transaction.

EXCHANGE MESSAGE SPECIFICATION SHEET

Transaction Processor [S][A][M][P][L][E]
Transaction Name [C][H][G][C][U][S]
Exchange Label [C][H][G][E][X][1]

Field No.	Starting Byte	Length (Bytes)	Contents
1	1	6	Customer Number
2			
3			
4			
5			
6			
7			
8			
9			
10			
11			
12			

Figure 2-1

2.6.1 Coding the Exchange Message in COBOL

The exchange message is the first of the two parameters passed to the TST by the transaction processor. It must be defined in the LINKAGE SECTION and its group data item name must be the first parameter in the USING clause of the PROCEDURE DIVISION header. For example:

```
LINKAGE SECTION.
01   EXCHANGE-MESSAGE.
           03   EM-CUSTOMER-NUMBER          PIC X(6).
```

2.6.2 Coding the BASIC-PLUS-2 MSGMAP Statements

(Exchange Message Map)

You use the BASIC-PLUS-2 MSGMAP statement to describe the exchange message data structure. The MSGMAP statement is a TRAX extension to BASIC-PLUS-2, and is a special type of MAP statement for use with exchange messages. The following example shows the BASIC-PLUS-2 MSGMAP statements used to describe the exchange message specified in Figure 2-1.

```
600 \     MSGMAP   EM.CUSTOMER.NUMBER$   =    6                    &
```

2.7 THE TRANSACTION WORKSPACE

The transaction workspace is the second data structure the transaction processor supplies to your TST. The transaction workspace provides context for subsequent TST processing. It should be noted that the system does not initialize the transaction workspace when a transaction instance begins. The transaction workspace is available to every TST in every exchange performed by a transaction instance.

If the transaction definition specifies a workspace, then you must specify a workspace parameter even if your TST does not reference it. If the transaction definition specifies a workspace with a length of zero, you must not specify a workspace parameter. The method of referencing the transaction workspace area differs according to the programming language you select.

When you specify the transaction workspace size in the transaction definition, the value you supply is rounded up to the next highest multiple of 64. For example, if you specify the transaction workspace area as 205 bytes, the transaction definition allocates 256 bytes for the transaction workspace area.

For any transaction, the combined space allocations of the exchange message and transaction workspace cannot exceed 8064 bytes.

Looking at the example transaction shown in Figure 1-2, you can see the transaction workspace in use.

The transaction CHGCUS illustrated in Figure 1-2, asks the terminal operator for a customer ID number, and then lists the corresponding customer record on the terminal.

The transaction workspace is first used to transmit the customer record from the first exchange (the process of getting the customer ID and retrieving the corresponding customer record), to the

TRANSACTION WORKSPACE SPECIFICATION SHEET

Transaction Processor S A M P L E

Transaction Name C H G C U S

Field No.	Starting Byte	Length (Bytes)	Contents
1	1	6	Customer Number
2	7	30	Customer Name
3	37	30	Address Line One
4	67	30	Address Line Two
5	97	30	Address Line Three
6	127	5	Zip Code
7	132	10	Telephone Number
8	142	20	Attention – Of
9	162	12	Credit Limit (9(10)V99)
10	174	12	Current Balance (9(10)V99)
11	186	12	Purchases Y-T-D (9(10)V99)
12	198	4	Next Order Sequence Number
13	202	4	Next Payment Sequence Number
14			
15			
16			
17			
18			
19			
20			
21			
22			
23			
24			
25			

Figure 2-2

second exchange. The second exchange is initiated by a response message that sends the information from the transaction workspace to the initiating terminal station where it is inserted into the form specified for the second exchange.

Figure 2-2 shows a transaction workspace specification prepared for this transaction type.

2.7.1 Coding the Transaction Workspace in COBOL

The transaction workspace is the second of the two parameters passed to the TST by the transaction processor. It must be defined in the LINKAGE SECTION and the group data item name must be specified as the second USING clause parameter in the PROCEDURE DIVISION header. The following example shows how the transaction workspace data structure described in Figure 2-2 is defined in the LINKAGE SECTION.

```
DATA DIVISION.

LINKAGE SECTION.
          •
          •
          •

01 TRANSACTION-WORKSPACE.
    02 WS-CUSTOMER-FILE-RECORD.
        03 WS-CUSTOMER-NUMBER          PIC X(6).
        03 WS-CUSTOMER-NAME            PIC X(30).
        03 WS-ADDRESS-LINE-1           PIC X(30).
        03 WS-ADDRESS-LINE-2           PIC X(30).
        03 WS-ADDRESS-LINE-3           PIC X(30).
        03 WS-ADDRESS-ZIP-CODE         PIC 9(5).
        03 WS-TELEPHONE-NUMBER         PIC 9(10).
        03 WS-ATTENTION-LINE           PIC X(20).
        03 WS-CREDIT-LIMIT-AMOUNT      PIC 9(12).
        03 WS-CURRENT-BALANCE          PIC 9(12).
        03 WS-PURCHASES-YTD            PIC 9 (12).
        03 WS-NEXT-ORDER-DATE          PIC 9(4).
        03 WS-NEXT-PAYMENT-DATE        PIC 9(4).
```

The level 01 data name describes the entire workspace. The level 02 data item describes a record data structure. The level 03 data items describe the individual fields in that record.

2.7.2 Coding the BASIC-PLUS-2 WRKMAP Statements

(Transaction Workspace Map)

The WRKMAP statement is used to describe the structure of the transaction workspace in BASIC-PLUS-2 TSTs. The WRKMAP statement can specify several levels of data structure mapping. WRKMAP is a TRAX extension to the BASIC-PLUS-2 language. It has all the capabilities of the MAP statement, however, a map name need not be specified.

The following example shows the use of the WRKMAP statement to describe the transaction workspace data structure shown in Figure 2-2.

```
600     WRKMAP                                                  &
                WS.CUSTOMER.FILE.RECORD$          = 205         &
\       WRKMAP                                                  &
                WS.CUSTOMER.NAME$           = 6                 &
        ,       WS.CUSTOMER.NAME$           = 30                &
        ,       WS.ADDRESS.1$               = 30                &
        ,       WS.ADDRESS.2$               = 30                &
        ,       WS.ADDRESS.3$               = 30                &
        ,       WS.ZIP.CODE$                =  5                &
        ,       WS.TELEPHONE.NUMBER$        = 10                &
        ,       WS.ATTENTION.LINE$          = 20                &
        ,       WS.CREDIT.LIMIT$            = 12                &
        ,       WS.CURRENT.BALANCE$         = 12                &
        ,       WS.PURCHASES.YTD$           = 12                &
        ,       WS.NEXT.ORDER.NUMBER$       =  4                &
        ,       WS.NEXT.PAYMENT.NUMBER$     =  4                &
```

In this example, the first WRKMAP statement defines a workspace of 205 characters, and assigns it to a string variable.

The second WRKMAP statement subdivides the record into individual fields, and assigns each field to a string variable.

2.8 EXITING FROM A TST

When writing TSTs, you must provide for an orderly exit.

In the case of a transaction instance initiated from an application terminal, an orderly exit depends on the type and status of processing in that TST.

- If your TST has completed processing successfully, and additional TSTs remain on the exchange routing list, then a normal exit point must be provided to allow processing to continue at the next station on the list.
- If your TST has detected an error condition that requires user intervention or termination of the transaction instance, the TST should delete any remaining stations from the routing list and send a response message back to the initiating terminal.
- Depending on the severity of the error, the response message may allow the user to edit and reenter the exchange, or may cause the transaction instance to be terminated.
- If your TST is the last station in an exchange routing list, you must send a response message back to the initiating terminal, and provide a normal exit point to release system resources.

In the last two cases, if the call to the response message library routine is unsuccessful, you must provide a facility for an orderly termination of the transaction instance.

The normal exit point should be coded according to the conventions of the programming language. The following sections describe the normal exit points for each source language.

2.8.1 Normal Exit from a COBOL TST

The EXIT-PROGRAM statement is used to perform a normal exit from a TST. This statement marks the logical end of the TST, and causes control to be returned to the transaction processor. An example of the EXIT-PROGRAM Statement is:

```
END-PROGRAM.
        EXIT PROGRAM.
```

2.8.2 Normal Exit from a BASIC-PLUS-2 TST

The BASIC-PLUS-2 language provides a TSTEND statement. This statement must be placed as the last statement in a TST. The TSTEND statement is the last statement processed by the BASIC-PLUS-2 compiler, and it causes the TST to exit when it is executed. (It is suggested that the TSTEND statement appear at line number 32767.) For example:

```
32000           ! PREPARE FOR EXIT !
32767           TSTEND
```

CHAPTER 3
TST FILE INPUT/OUTPUT OPERATIONS

TRAX operates on files using the standard file I/O syntax of COBOL and BASIC-PLUS-2. In addition, TRAX supports several types of record locking operations designed to improve data base integrity.

This chapter introduces important TRAX file processing concepts and explains how to perform I/O operations from TSTs. The permitted language syntax for COBOL and BASIC-PLUS-2 is documented in the respective Language Reference Manuals. The discussion in this chapter centers on the specific implications of file I/O operations in TSTs.

3.1 APPLICATION DATA FILES

Every transaction processor has a set of application data files associated with it. These files consist of two types, permanent files and work files. The association between a transaction processor and a specific data file is made by the FILDEF utility program (See Chapter 12) which matches the physical file specification with a logical file name assigned by the application designer. Various file attributes are made known to the transaction processor during the execution of the FILDEF utility.

When a transaction processor is started, all permanent data files defined for that transaction processor are opened by the TRAX data management routines.

Data files are organized in one of three ways:

- *Sequential* In the sequential file organization, records appear in physical sequence. Each record, except the first, has another record preceding it, and each record, except the last, has another record following it. The records appear in the physical order they were originally written to the file.

- *Relative* A relative file is structured as a series of fixed-size record cells. Cell size is based on the size you specify as the maximum length permitted for a record in the file. These cells are numbered in succession from 1 (the first) to n (the last). The cell's number represents its location relative to the beginning of the file.

  Each cell in a relative file can contain a single record. The file can have empty cells interspersed with full cells.

  The cell numbers in a relative file are unique. You can use them to identify both the cell, and the record occupying that cell. When you use a cell number to identify a record, the cell number is the relative record number.

- *Indexed* Indexed files are organized according to keys in the individual records. A key is a character string present in every record of an indexed file. The location and length of a key is identical in all records.

  Every indexed file must have at least one key specified. This primary key must be a unique identifier on the record. Duplicate primary key values are not permitted. You may also specify alternate keys for indexed files.

  Alternate key values can be duplicated in a file and may be changed by record updates. As you write records into an indexed file, RMS builds a tree-structured table known as an index. Every key defined for a file has a separate index.

3.2 TYPES OF DATA FILES

TRAX supports two classes of application data files; permanent files, and work files.

Permanent files must be created and defined before a transaction processor is started. Permanent files make use of full RMS file sharing capabilities and permit individual record locking. The data in permanent files can be accessed by many concurrent transaction instances. Permanent files can be organized as sequential, relative, or indexed files.

Work files are generally temporary files. A work file is created by an executing transaction instance, and may be subsequently modified by that same instance. Once that transaction instance is closed, only programs in the support environment may process that data. Since work files are not shared, record locking is not supported for work files.

Typically work files are used to:

- Hold temporary data specific to a transaction instance.
- Build command files for use by a support environment program.

Work files may be organized as sequential or relative files.

Work files are opened and connected by the OPEN statement and disconnected and closed by the CLOSE statement in the TST.

3.3 FILE ACCESS METHODS

TRAX supports two file access methods; sequential and random. *Sequential access* means that records are retrieved or written in the a sequence implied by the file organization. Sequential access to sequential files implies the records are accessed one-by-one starting with the first physical record in the file. To read the fifth record in a sequentially organized file, the first four records must be read first.

Records preceding the current record location cannot be read unless the sequential file is first closed, then reopened and accessed from the beginning of the file. TRAX does not support rewinding of sequential files.

Sequential access to relative files is based on the cell numbers in the file. The records are accessed in cell number order beginning with the first cell and continuing in ascending cell number order. Empty record cells are ignored. Sequential write operations on relative files are permitted only to empty cells in the file.

Sequential access to indexed files is based upon the key specified in the operation. If a file is organized with two keys, account number, and name, a sequential read operation using the account number key begins with the specified account number and accesses the next higher number each time. Similarly, a sequential read on name begins with the specified name and accesses the next alphabetically sequenced name each time.

Random Access. In random access mode, the program, rather than the file organization determines the order in which records are processed. Each program request for a record operates independently of the preceding record access. In random mode, the I/O operation must identify the desired record when it issues the call to the I/O routines. Random access is not permitted on sequential files.

Random access on relative files is based on the relative record number specified in the I/O statement. A program can read relative records at random by specifying cell numbers. In random access mode, you can operate on record number 47 then record number 13 followed by record number 31.

Random access on indexed files is based on the key value specified in the I/O statement. If you want to read the record with the alternate key value JONES, then the key ABRAMS, and then SMITH, you use the random access mode, and specify the exact key value when you call the I/O routine.

TRAX also supports access by both methods in the same program. This allows you to access a record randomly, then access all following records in a sequential fashion. The implementation of this feature is language dependent.

3.4 LOGICAL DATA FILE NAMES

Any file that a TST intends to access, including work files, must be defined in the framework of the transaction processor.

You use the FILDEF utility to assign a 1- to 6-character logical file name to each data file. After you answer a set of questions, FILDEF creates a set of internal file definitions which the processor utilizes each time a TST attempts to access a file.

The logical names assigned to permanent data and work files must be used by every TST that references these files. The file definition record relates the file's logical name to its corresponding file specification.

3.5 PREPARING TO USE FILES FROM A TST

Each application data file that you reference in a TST must have a Input/Output channel number assigned to it in the TST. This channel number allows the data management routines of the transaction processor to connect defined application data files to your TST. Selection of I/O channel numbers is made by the application designer at the time the TST is specified. There are two rules governing channel assignments:

1. The same channel number cannot be used for two different files in the same transaction instance.
2. If a file is opened on a specific channel number, in order to maintain context, subsequent TSTs in that transaction instance must use the same channel number to reference that file.

The range of channel numbers is dependent upon the source language you are using. Before you can perform operations on application data files, you must logically connect the files to your TST. Preparing for file operations differs in each supported source language. The following general rules apply to TRAX applications:

- The transaction processor opens all defined application data files at the time it is installed.
- The file open operation in the TST is actually a logical channel connection to the application data file.
- You must properly specify the file organization, access method, and file attributes in your TST. This data affects the type of calls your TST makes to TRAX data management services routines.

TSTs must execute an OPEN statement to connect a stream to a file and to establish record context between TSTs in a transaction instance. Normally OPEN performs two distinct functions; opening a file and connecting an I/O channel from the program to the file. In a TST, the file is opened by the transaction processor. The OPEN statement performs only the channel connect operation. The OPEN is not performed by the OPEN statement in the TST.

The following is an example of maintaining record context. TST1 connects to a file using channel 1. Suppose TST1 sequentially reads the first five records on the file and exits. Later in the same transaction instance, TST2 connects to the same file on channel 1. A sequential read by TST2 retrieves the sixth record from the file. This is possible since TST1 didn't CLOSE the file, and record context was maintained for subsequent TSTs in the transaction instance. If TST1 issued a CLOSE, then the sequential read by TST2 would have retrieved the first record instead of the sixth. When the OPEN statement in a TST is executed, the channel specified in the OPEN statment is connected to the specified data file. A channel is disconnected and all locks on the channel are released by a CLOSE statement, or when the transaction instance terminates. Permanent files are closed when the transaction processor is stopped.

In the case where a sequential file is opened for output, only one copy of the TST performing the update operations should be defined in the station definition file. This allows the transaction processor executive to queue the write requests to the sequential file in the order they are issued by the TST. Only one TST in a transaction processor can access a sequential file with WRITE access.

NOTE

Only one channel can be connected to a sequential file. Sequential files may not be shared among transaction instances.

3.6 CLOSING DATA FILES

Permanent data files are normally closed when the transaction processor is stopped. Work files are closed:

1. By the explicit action (a CLOSE statement) of some TST.
2. When a transaction instance terminates, and a work file is open, the work file is closed by the transaction processor.

If your TST contains a close statement that acts on a permanent data files, the channel connecting your TST to that file is disconnected. The file is not actually closed by the TST.

NOTE

You cannot rewind a sequential work file. You can rewind a sequential permanent file by first closing the file, then reopening it.

3.7 RECORD LOCKING FACILITIES

TRAX allows TSTs to lock records during processing. This feature provides you with a valuable tool to insure data base integrity and proper sequencing of updates to the data base.

All updates to data files SHOULD be preceded by a READ with LOCK to the specified record.

A record is locked by either of the following actions:

1. Executing a statement that specifies the LOCK keyword explicitly locks the record.
2. Modifying a record on a staged file (ADD, UPDATE, DELETE), implicitly locks the record.

There are two different types of lock conditions, the soft lock and the hard lock. The two lock types return different error codes to the TST when a locked record is retrieved.

The *soft lock* is any lock applied to a record in a file that allows read access to other transaction instances. Applying a soft lock to a record permits other transactions to read the locked record but not to modify it. Your TST can successfully read a soft locked record from a file.

The *hard lock* is any lock applied to a record in a file that does not permit access to locked records.

The following examples illustrate the difference between a soft and a hard lock.

If the customer file CUSTOM is defined with read access allowed to locked records, then any read operation attempted on a record locked by another transaction instance returns the soft lock error.

If, on the other hand, CUSTOM did not permit read access to locked records, then a read operation attempted on a record locked by another transaction instance returns the hard lock error.

If a hard lock is encountered during a read operation, the request for the records is placed on the lock wait queue. The request remains in the lock wait queue for the number of seconds defined as the interval between attempts to access locked records for the file that contains the record. (See the FILDEF utility description in Chapter 12.) At the end of this time interval, if the lock for the record is not released, an error is returned to your TST. If the lock is removed before the lock wait time expires then the first request (for that record) is given the record. In this case, the TST is unaware of the fact the record was locked.

NOTE

When coding your TSTs, a recommended practice is to lock all the needed records for the requested operation in the same exchange. This technique provides you with the greatest number of recoverable alternatives when a lock error is encountered.

If a record operation (with or without LOCK) fails because the record is locked by another task, you may recover using three possible alternatives:

1. Have the TST perform an exchange restart. Restarting an exchange should be attempted only if all the locks to this point are set in the current exchange and the transaction is defined with exchange recovery enabled.

NOTE

Attempting to restart an exchange where locks have been applied to records in preceding exchanges may result in deadly embrace. You should code your application so that all locks in a transaction instance are applied in the same exchange.

2. If all locks are set in the current exchange, your TST must release all locks, notify the initiating station of the lock condition with a REPLY response message, and request the user to retry the exchange.
3. Send an ABORT or CLOSE response message to the initiating station, notifying the user that a lock error has occurred. This alternative forces the initiating station to return to the initial state.

3.8 READING RECORDS FROM FILES

Sequential files may be read using sequential access mode. Relative and indexed files can be read using either sequential or dynamic access mode. Indexed files also support exact, approximate and generic key matching in random access mode. In this case, your TST specifies that the key data field of the record retrieved is equal to, equal to or greater than, or greater than the program supplied key value.

3.9 INSERTING RECORDS INTO FILES

Records written to sequential files are added after the last record currently in the file. Records written to relative and indexed files are added to the file according to the key data specified in the record being written.

3.10 UPDATING EXISTING RECORDS TO A FILE

On indexed files, TRAX does not allow you to update the primary record key. Secondary key values can be modified by any TST accessing the file for output.

A suggested practice is to always lock a record when you retrieve it for an update operation. You can release the lock as part of the statement that updates the file.

3.11 DELETING RECORDS FROM A FILE

The type of deletion that is physically executed on a record depends upon the deletions parameter specified in the application data file definition. If "fast" deletions are enabled, then TRAX marks the intended record for deletion. Otherwise, the record is physically deleted from the file.

NOTE

You cannot delete records from a sequential file.

3.12 I/O ERROR HANDLING

TSTs should include error handling routines. A typical error handler consists of a routine that checks the I/O error return, formats an error message, and calls the REPLY or ABORT routine to send a response message to the initiating station.

I/O errors that are not trapped by a TST are trapped by the transaction processor. The transaction instance is immediately aborted, and a error is logged to the Software Error Log.

3.13 STAGED FILE OPERATIONS

File staging implies that updates to the file are performed at the conclusion of a transaction instance. The updating of the file, called unstaging, occurs asynchronously after the conclusion of the transaction instance. If the transaction instance fails for any reason, any updates from that transaction instance are ignored.

FILE DEFINITION

Part One

Transaction Processor Name: [S][A][M][P][L][E]

Logical Filename: [C][U][S][T][O][M]

RMS File Specification: [ ][ ][ ]:[[3][5][0].[2][2][7]][C][U][S][T][O][M].[D][A][T];[ ][ ]

Work File?
- [ ] – Yes (Go to Part Two)
- [X] – No (Continue with next question)

Is This an Indexed File?
- [X] – Yes: No. of Keys [ ][2]
  Maximum Key Length [ ][3][Ø]
- [ ] – No: Sequential or Relative File

Maximum Concurrent File Accesses? [ ][4]

Read-Only?
- [ ] – Yes
- [X] – No

Fast Deletions?
- [ ] – Yes
- [X] – No

Lock Interval [ ][2] seconds

Read Access to Locked Records?
- [ ] – Yes
- [X] – No

Journal?
- [ ] – Yes (Go to Part Two)
- [X] – No (Continue with next question)

Staged File Updates?
- [ ] – Yes
- [X] – No

Part Two

File Channel Assignment

Description of File Contents: CUSTOMER FILE RECORD

Assigned I/O Channel Number [ ][3]

Figure 3-1 Customer File Definition Sheet

A file is defined as staged during the file definition procedure (See chapter 12).

Any updates to a staged file (adding, updating, or deleting a record) apply a lock to that record. This lock remains in effect until unstaging takes place at the close of the transaction instance.

Adding a record to a staged file causes that record to be immediately written to the file. Should the transaction instance fail, the added record is deleted during unstaging.

If you update a record on a staged file, the updated record is written to the staging area in the system workspace. If the transaction instance terminates normally, the record is written to the file from the staging area. If the transaction instance is aborted, the staged record is ignored.

If you delete a record in a staged file, the deleted record is saved in the staging area. The deletion is performed after the transaction instance is successfully closed. If the transaction instance is aborted, the deletion is not performed.

Since the actual update takes place after the transaction instance is closed, an I/O error can occur during unstaging. For example, you must insure that primary key values are unique, and have not been changed during an update. If a file does not permit duplicate alternate keys, you must check to make sure that both the primary and alternate key values are uniques. This validation must be performed as part of a TST if the transaction instance references a staged file.

You may not close and reopen a staged sequential file during the same transaction instance. Closing a file disconnects the channel. This channel is required for unstaging, hence the restriction.

If you attempt to read a record from a staged file using the alternate key value, and that alternate key has been modified during the same transaction instance, the record is retrieved as though the update had never occurred.

3.14 EXAMPLES OF COBOL TST I/O OPERATIONS

The following sections use examples taken from a set of TSTs written for the TRAX Sample Application. In addition, each section discusses the valid language syntax elements allowed for file operations from TSTs. The examples are based on the customer file definition in Figure 3-1 and the record layout in Figure 3-2.

3.14.1 Preparing for I/O in a COBOL TST

Before you can access files from a COBOL TST, you must first set up the file access parameters and channel. This requires a channel assignment, a file control paragraph, a file descriptor entry, a data record definition, and an OPEN statement.

To specify a COBOL TST I/O channel, you should create a level 01 data name for each application file. The data item must be defined within the TSTs working storage section. The value of the data item is formed by concatenating the logical file name to the string /CH:n. (where n represents a channel number in the range 1-64) For example, the COBOL statement creating the elementary data item that identifies the Customer File (whose logical file name for this transaction is CUSTOM) as the file assigned to I/O channel 3 is:

```
01   CUSFILNAME PIC X(11)
     VALUE IS "CUSTOM/CH:3".
```

Before you reference using a file in a COBOL TST, you must first identify (using the COBOL SELECT verb) the file, and declare its attributes and data structure in the program. This process begins in the ENVIRONMENT DIVISION of a COBOL TST. In the INPUT-OUTPUT SECTION, any file that you reference in the program must be specified in a FILE-CONTROL paragraph, using the SELECT statement. For example:

```
ENVIRONMENT DIVISION.
INPUT-OUTPUT SECTION.
FILE-CONTROL.
  SELECT  CUSTOM ASSIGN TO "CUSTOM.DAT"
          ORGANIZATION IS INDEXED
          ACCESS MODE IS RANDOM
          RECORD KEY IS CUSTOMER-NUMBER
          ALTERNATE RECORD KEY IS CUSTOMER-NAME
          FILE STATUS IS CUSTOMER-FILE-STATUS.
```

This example shows the FILE-CONTROL paragraph for the customer master file. In the SELECT statement, you identify the application data file, assign a logical file name, and specify the organization and access characteristics of the application data file. You also specify a data item where the system is to return status information to the TST.

The next step in making the file known to the TST occurs in the DATA DIVISION. In a file definition group item, you must specify the label records and the file identifier, and identify the COBOL level 01 group name describing the data structure of the records in the file.

```
DATA DIVISION.
FILE SECTION.
FD CUSTOM
  LABEL RECORDS ARE STANDARD
  VALUE OF ID IS CUSFILNAME
  DATA RECORD IS CUSTOMER-FILE-RECORD.
```

In this example, the logical file name, "CUSTOM" identifies the FD item. TRAX always assumes standard labels for application data files. The data item specified in the VALUE of ID clause contains the channel number specification that was discussed in Section 3.5.

The data structure of the customer record must also be described in the DATA DIVISION, following the FD group item. The following data structure corresponds to the design specification of the customer file shown in Figure 3-2.

```
01  CUSTOMER-FILE-RECORD.
    03  CUSTOMER-NUMBER                        PIC X(6).
    03  CUSTOMER-NAME                          PIC X(30).
    03  ADDRESS-LINE-1                         PIC X(30).
    03  ADDRESS-LINE-2                         PIC X(30).
    03  ADDRESS-LINE-3                         PIC X(30).
    03  ADDRESS-ZIP-CODE                       PIC 9(5).
    03  TELEPHONE-NUMBER                       PIC 9(10).
    03  ATTENTION-LINE                         PIC X(20).
    03  CREDIT-LIMIT-AMOUNT                    PIC 9(10)V99.
    03  ACCOUNT-POINTER REDEFINES CREDIT-LIMIT-AMOUNT.
            05  FILLER                         PIC 9(6).
            05  NEXT-ACCOUNT-NUMBER            PIC 9(6).
    03  CURRENT-BALANCE                        PIC 9(10)V99.
    03  PURCHASES-YTD                          PIC 9(10)V99.
    03  NEXT-ORDER-DATE                        PIC 9(4).
    03  NEXT-PAYMENT-DATE                      PIC 9(4).
```

All file operations are performed in the PROCEDURE DIVISION of a COBOL TST. Prior to performing any I/O operation, you must first open the files you wish to access. You may choose to open a file for Input, Output, or both. The OPEN statement for a data file must appear in the Procedure Division before any I/O operations can be performed on that file.

To create a work file in COBOL, use the OPEN statement with the OUTPUT keyword.

The following example shows a file opened for both input and output in the COBOL TST "REWRIT", which is used to insert a new customer record into the customer file.

```
OPEN-FILES.
  OPEN I-O CUSTOM.
  IF CUSTOMER-FILE-STATUS IS GREATER THAN "09"
  GO TO END PROGRAM.
```

In this example, the file "CUSTOM" is opened for I-O. At run-time, if the return code in the file status word of the customer file indicates the open operation failed, then processing branches to a routine that terminates the TST. This TST is written with a USE procedure to handle I/O errors. Error handling is discussed in Section 3.14.6.

3.14.2 Using COBOL to READ Records from a File

TRAX conforms with standard COBOL syntax when performing read operations on a file. In addition, TRAX allows you to lock records during a read operation by specifying the WITH LOCK clause following the READ verb.

RECORD LAYOUT SHEET

Transaction Processor: SAMPLE

File Description: Customer Master File ([350,227]Custom,DAT)

Logical Filename: CUSTOM

This is Record Format: 1 of 1

Logical Record Length: 205

Physical Record Length: ____ (Tape Only)

Field No.	Starting Byte	Length (Bytes)	Contents	Data Type
1	1	6	Customer Number	
2	7	30	Customer Name	
3	37	30	Address Line One	
4	67	30	Address Line Two	
5	97	30	Address Line Three	
6	127	5	Zip Code	
7	132	10	Telephone Number	
8	142	20	Attention-Of	
9	162	12	Credit Limit(9(10)V99)	
10	174	12	Current Balance(9(10)V99)	
11	186	12	Purchases Y-T-D(9(10)V99)	
12	198	4	Next Order Sequence Number	
13	202	4	Next Payment Sequence Number	
14				
15				
16				
17				
18				
19				
20				
21				
22				
23				
24				
25				

Figure 3-2 Record Layout Sheet Describing Customer Record.

For example, the following statement in a COBOL TST performs a random read operation on an indexed customer file according to the primary key value, and locks the record at the completion of the read operation. Following the READ statement, two IF statements test for the presence of I/O errors, and for the "soft lock" condition.

```
READ WITH LOCK CUSTOM RECORD.
IF FILE-STATUS-WORD IS NOT EQUAL TO ZERO GO TO END-PROGRAM.
IF CUSTOMER-FILE-STATUS IS EQUAL TO "92" GO TO LOCKED-RECORD.
```

If the statement is successful, the file status word contains "01" after the record is returned to your program. If the record was read successfully, but locked by another task, the file status word contains "92".

If the requested record cannot be read because of a hard lock applied by another task, an error is trapped to the USE procedure in the COBOL TST, and the file status word contains "92".

A second form of read operation is performed when suing the COBOL dynamic access mode. The START verb allows you to specify a key value and randomly position a pointer in a relative or indexed file. Once the START verb is executed, you can execute sequential read operations on the file from that point. The following example shows how the START verb and the READ verb are used to set up a secondary key search of the customer file.

```
START CUSTOM KEY IS NOT < CUSTOMER-NAME
  IF FILE-STATUS-WORD IS NOT EQUAL TO ZERO GO TO END-PROGRAM.

READ CUSTOM NEXT RECORD
  IF FILE-STATUS-WORD IS NOT EQUAL TO ZERO GO TO END-PROGRAM.
  IF CUSTOMER-FILE-STATUS IS EQUAL TO "92" GO TO LOCKED-RECORD.
```

The error handling in this example consists of a USE procedure in the DECLARATIVES section. If the error path has been activated, FILE-STATUS-WORD is greater than zero. The second IF statement checks for the "soft lock" error.

The syntax of the START verb does not recognize the WITH LOCK clause.

3.14.3 Using COBOL to Write Records to a File

TRAX conforms to standard COBOL syntax when writing records to a file. In addition, TRAX allows you to apply a lock to a record after a write operation, by specifying the WITH LOCK CLAUSE following the WRITE verb.

The following COBOL statement writes a customer record to the file, and retains a lock on the record until processing is completed. Following the WRITE statement, an IF statement checks to see if an error has been detected, and the USE procedure has altered the value of FILE-STATUS-WORD.

```
WRITE-CUSTOMER-FILE-RECORD.
IF FILE-STATUS-WORD IS < 0 GO TO END-PROGRAM.
```

3.14.4 Using COBOL to Update Records on a File

TRAX permits standard COBOL synatax for update operations. In addition, you may specify the clauses with LOCK and WITH UNLOCK as part of the REWRITE verb syntax.

In the following example, the customer record is being rewritten to the file.

```
REWRITE WITH UNLOCK CUSTOMER-FILE-RECORD.
IF FILE-STATUS-WORD IS < 0 GO TO END-PROGRAM
```

Status information is returned to the TST in the file status word specified in the SELECT statement.

If you attempt to update a record that is locked by another transaction instance, the COBOL TST traps to the USE procedure, and returns a file status code of "92".

3.14.5 Using COBOL to Delete Records from a File

To delete a record from a file in COBOL, you use the DELETE verb. COBOL requires that your transaction instance read the record, and store the primary key in the data structure referenced in the file descriptor entry, prior to deleting it.

The following example shows how to use the COBOL DELETE verb:

```
DELETE CUSTOM RECORD.
IF FILE-STATUS-WORD IS < 0 GO TO END-PROGRAM.
```

3.14.6 I/O Error Handling in COBOL TSTs

The COBOL USE procedure is an efficient way of trapping errors in COBOL TSTs. When an error is detected by the system, it alters the file status word for that COBOL program and branches to a defined USE procedure in the DECLARATIVES section of the PROCEDURE DIVISION.

The following example shows a USE procedure that formats REPLY messages for file status values of "10", "23" and "92". All other errors are trapped to a section that formats an ABORT message containing the file status value.

```
PROCEDURE DIVISION USING EXCHANGE-MESSAGE, TRANSACTION-WORKSPACE.
DECLARATIVES.

I-O-ERROR SECTION.

  USE AFTER STANDARD ERROR PROCEDURE ON CUSTOM.

  IF CUSTOMER-FILE-STATUS IS < 0
  MOVE "CUSTOM" TO RMB-FILE-NAME
  MOVE CUSTOMER-FILE-STATUS TO FILE-STATUS-WORD

  IF FILE-STATUS-WORD IS EQUAL TO "10"
  MOVE "Reached End of File" TO REPLY-MESSAGE-BUFFER
          GO TO SEND-REPLY-MESSAGE.

  IF FILE-STATUS-WORD IS EQUAL TO "23"
  MOVE "No Record for that ID" TO REPLY-MESSAGE-BUFFER
          GO TO SEND-REPLY-MESSAGE

  IF FILE-STATUS-WORD IS EQUAL TO "92"
  MOVE "The Record you wanted is
-       "locked by another user. You may press CLOSE to exit,
-       "or you may wait and press ENTER to try again."
  TO REPLY-MESSAGE-BUFFER
          GO TO SEND-REPLY-MESSAGE.
```

```
    MOVE " I-O ERROR.  STATUS WORD IS: " TO REPLY-MESSAGE-TEXT
    MOVE FILE-STATUS-WORD TO REPLY-FILE-STATUS.

SEND-ABORT-MESSAGE.

    MOVE 160 TO BUFFER-SIZE
    MOVE 2 TO REPLY-NUMBER
    CALL "ABORT" USING
                        REPLY-MESSAGE-BUFFER
                        BUFFER-SIZE
                        REPLY-NUMBER
                        STATUS-WORDS.
    GO TO END-ERROR-SECTION.

SEND-REPLY-MESSAGE.

    MOVE 160 TO BUFFER-SIZE
    MOVE 2 TO REPLY-NUMBER
    CALL "REPLY" USING
                        REPLY-MESSAGE-BUFFER,
                        BUFFER-SIZE,
                        REPLY-NUMBER,
                        STATUS-WORDS.

END-ERROR-SECTION.
END DECLARATIVES.
```

NOTE

If you wish to use record locking from COBOL TSTs, you must trap I/O errors through a USE procedure in the DECLARATIVES section of the COBOL TST. This is the only way you can detect the difference between a hard and a soft lock error.

3.14.7 Closing Data Files in COBOL TSTs

You are cautioned *not* to issue a CLOSE statement at the end of a TST. The CLOSE statement causes the file channel to be disconnected.

3.14.8 The UNLOCK and UNLOCK ALL Verbs

The UNLOCK verb releases all record locks held by the current transaction instance for the specified logical file name. The UNLOCK ALL verb releases all record locks held by the current transaction instance. When a transaction instance closes, all record locks are released automatically.

3.14.9 Unsupported COBOL Syntax

In TSTs designed to run in a production version of a transaction processor, the use of the ACCEPT and DISPLAY verbs, and calling any of the SORT subprograms causes a software error to be logged and the transaction instance to be aborted.

ACCEPT and DISPLAY may be used in conjunction with the transaction processor debug facility described in Chapter 14.

3.15 EXAMPLE OF BASIC-PLUS-2 TST I/O OPERATIONS

The following sections use examples taken from a set of TSTs written for the TRAX Sample Application. In addition, each section discusses the valid language syntax elements allowed for file operations from TSTs.

3.15.1 Preparing for I/O Operations in BASIC-PLUS-2 TSTs

In BASIC-PLUS-2, the channel number corresponding to each logical file name is assigned to an integer data item. The channel number data item is used for all subsequent I/O operations on that data file. For example, if you want to assign the Customer File to I/O Channel 3 the following BASIC-PLUS-2 coding is needed:

```
300         CUSTOM.CHNL% = 3%
```

In BASIC-PLUS-2, a literal or variable must be used to identify files in I/O statements. In TSTs, the file number and the channel number are the same variable. The BASIC-PLUS-2 examples in the remainder of this chapter all use CUSTOM.CHNL% to identify the customer file.

In a BASIC-PLUS-2 TST, the OPEN statement performs much of the I/O preparation. You can specify a set of parameters defining the file organization and characteristics as part of the OPEN statement. TRAX imposes the following constraints on the way you specify the open statement.
• The logical file name must be specified as the file name in the open statement. This name is used by the TRAX data management routines to connect the channel to the file. • The file organization and record type must agree with the attributes of the file. For example, a fixed type record cannot be specified as variable in the OPEN statement.

```
2100        OPEN "CUSTOM" AS FILE CUSTOM.CHNL%,                    &
            ORGANIZATION          INDEXED,                         &
                                  VARIABLE,                        &
            ACCESS                MODIFY,                          &
            ALLOW                 READ,                            &
            MAP                   CUSTOM,                          &
            PRIMARY KEY           CUSTOMER.NUMBER$                 &
            ALTERNATE KEY         CUSTOMER.NAME$                   &
                                  DUPLICATES                       &
                                  CHANGES                          &
```

To create a work file in BASIC-PLUS-2, use the OPEN statement. Specify the ACCESS WRITE clause in the OPEN statement.

The BASIC-PLUS-2 CLOSE statement should not be used except when a TST PUTs records to a sequential file.

3.15.2 Reading Records from BASIC-PLUS-2 TSTs

The only BASIC-PLUS-2 statements you can use for reading files from TSTs are FIND and GET. In BASIC-PLUS-2, the GET statement reads the record from a file into a buffer. You can issue the GET statement in both sequential and random access modes. For example:

```
3000 GET #CUSTOM.CHNL% KEY #1 IS GE EM.CUSTOMER.NAME$
```

The FIND statement locates a record in the file and sets the current record pointer to that location. The found record is not returned to your program.

The ,LOCK clause specifies record locking for both the FIND and GET statements. If you attempt to GET a hard locked record, BASIC-PLUS-2 error 154 is returned to the TST.

If you successfully read a record locked by another task, BASIC-PLUS-2 error 172 is returned to the TST. In the case of error 172, the record is returned to the TST, and the program traps to the line specified as the error routine.

3.15.3 Using BASIC-PLUS-2 to Put Records in a File

The only statement that can be used to write to files in a BASIC-PLUS-2 TST is the PUT statement.

The PUT statement writes a record to the file. Before executing a PUT statement, you must move the record contents to the MAP defined for that file. An example of the PUT statement is:

```
3000 PUT #CUSTOM.CHNL%
```

In addition, you can lock the record you are writing to the file by the addition of the ,LOCK clause to the PUT statement.

3.15.4 Using BASIC-PLUS-2 to Update Records on a File

The UPDATE statement replaces an existing record in the file with a new one. For example:

```
3000 UPDATE #CUSTOM.CHNL%
```

If you hold a lock on the record, executing the UPDATE statement unlocks it, unless you specify the ,LOCK clause with the UPDATE statement.

3.15.5 Using BASIC-PLUS-2 to Delete Records from a File.

The DELETE statement deletes a record from a file. An example is:

```
5000 DELETE #CUSTOM.CHNL%
```

3.15.6 I/O Error Handling in BASIC-PLUS-2 TSTs

BASIC-PLUS-2 provides a means of handling I/O errors through the ON ERROR GO TO statement. This allows you to set up a common error handling block in your TST.

In the special case of a FIND or GET statement executing a successful read operation on a record locked by another task, BASIC-PLUS-2 returns an error code of 172. All other types of record locking errors are returned to BASIC-PLUS-2 programs as error 154.

NOTE

You cannot use the ERT$ function in BASIC-PLUS-2 TSTs.

The following example is taken from the TST "NEXT" and shows the error handling statements to deal with record locks encountered during a GET operation.

```
1000        ON ERROR GO TO 19000
                    •
                    •
                    •

19000       !$*****************************************************************&
            !          S T A N D A R D   E R R O R   H A N D L I N G           &
            !******************************************************************&

19020       \   IF ERR = 155%
                AND (ERL = 4110% OR ERL = 4210%)                              &
                THEN RESUME 5200                                              &
            !                                                                 &
            !              Trap for Customer ID not on file                   &
            \   IF ERR = 172% THEN RESUME 4800                                &
            !                                                                 &
            !       Trap for Record 000000 Returned but locked                &
            !                                                                 &
            \   IF ERR = 154%                                                 &
                THEN                                                          &
                REPLY.BUF$ = "Access Denied, Record Locked by Another Task"   &
            \   CALL REPLY BY REF (REPLY.BUF$,LEN(REPLY.BUF$),2%,STATUS%()) &
            \   GO TO 19950   IF STATUS%(0%) <> 1%                            &
            \   GO TO 32000                                                   &

19900       \   REPLY.BUF$ = " I-O Error Number "                             &
                            +NUM$(ERR)                                        &
                            +"at Line # "                                     &
                            +NUM$(ERL)                                        &
            !                                                                 &
            !       For unexpected errors, go to                              &
            !          system default error dump out.                         &
```

3.15.7 The UNLOCK and FREE statements

BASIC-PLUS-2 TSTs can use the UNLOCK and FREE statements in conjunction with record locking operations.

The UNLOCK statement removes the lock from the last record locked by this transaction instance. The FREE statement removes all locks that currently exist on the specified channel.

3.15.8 Non-supported BASIC-PLUS-2 file syntax

BASIC-PLUS-2 TSTs cannot reference virtual array or terminal format files. The INPUT #, INPUT LINE #, LINPUT #, and PRINT # statements are not recognized in BASIC-PLUS-2 programs compiled with the /TST switch.

CHAPTER 4
CALLING THE TRAX SYSTEM LIBRARY FROM A TST

4.1 TRAX SYSTEM LIBRARY ROUTINES

The TRAX system library routines allow TSTs to perform a variety of functions related to transaction processing. The library routines fall into several functional groups:

1. Report Message – The REPORT routine sends data from a TST to an output-only terminal station. The terminal station merges this data with a report form definition and prints the resulting report on an LA-180 hard copy terminal.
2. Response Messages – The REPLY, PRCEED, ABORT, STPRPT, CLSTRN, and TRNSFR routines send response messages from a TST.

   A response message carries data from a TST to the terminal station, where it is reformatted for display on the application terminal. At the same time, response messages control the sequence of exchange execution within the transaction.

3. Transaction Control – The TSPAWN, RESTRT, and TABORT routines initiate, restart, or abort a transaction from within a TST.
4. Routing List Control – The AROUTE, DROUTE, and DALLRT library routines add or delete stations from the current exchange routing list.
5. Mailbox Storage and Retrieval – The SNDMBX, GETMBX, and MBXNUM routines allow your TST to direct messages to or retrieve messages from a mailbox station defined in your transaction processor.
6. System Information – The GETIME, GETSTN, GETSRC, GETRAN and GETFIL routines obtain system information, such as date, time, station identifiers, transaction names, and file specifications for use by your TST.
7. Logging Transactions – The LOGTRN routine logs information that you specify to the journal file.

You call these routines from a TST using the standard calling sequence of the appropriate programming language. The parameters for each routine must be specified in the order shown in the library routine descriptions.

The buffer referred to in the parameter descriptions can be a record, variable, data item, etc. It contains information requested by the initiating station.

The buffer size should be the exact length of the message being sent.

These routines are stored in a resident shared system library. The TSTBLD utility automatically links these routines to your TST task image.

The following sections describe each routine in detail and includes the calling sequences from COBOL and BASIC-PLUS-2 programs. Status return information is given for each routine.

Each functional grouping of system library routines is preceded by a paragraph explaining the purpose and use of that group of routines.

4.2 THE REPORT ROUTINE

Sending a Report Message to an Output-Only Terminal Station

Description: You call the REPORT routine to send preformatted reports from a TST to an output-only terminal station. The terminal station merges this data with a report form definition and prints the resulting report on an LA-180 hard copy terminal. to output-only terminal stations.

The name of an existing ATL form definition must be specified in the parameter list.

Only one form definition can be associated with the call to the REPORT routine.

If the number of characters in the report message is greater than the number of characters requested by print fields on the receiving form, the excess characters are ignored. If the number of characters in the message is less than the number requested by the form, spurious characters appear in the unspecified fields.

If you want to print a report with more than one page, the pagination must be specified as part of the form definition. The REPORT routine allows only one form name. For this reason, the form must specify all of the page layouts in a multi-page report.

4.2.1 Using the REPORT Routine from COBOL TSTs

Calling Parameters:

The REPORT routine parameters listed here must be specified in the order shown.

Parameter	Description and Use
buffer	The name of the buffer containing data to be sent to the form. The buffer parameter refers to a data item name in the DATA DIVISION. For example: 01 WS-CUSTOMER-FILE-RECORD. (Record data structure items)
buffer size	The length (in bytes) of the data buffer. The buffer-size parameter must be a data item of type PIC S9(4) COMP. The data item is used to specify this parameter in the DATA DIVISION of the example TST: 01 BUFFER-SIZE PIC S9(4) COMP.

station name — The 6-character ASCII name of the output-only terminal station where you want the report printed. This is the station name assigned in the STADEF utility. The station name should be defined as a character data item in the data division. For example:

```
01  STATION-NAME PIC X(6).
```

form name — The 6-character ASCII name of the form definition that is to be used to format the report. The form name should be specified as a character data item in the DATA DIVISION. For example:

```
01  FORM-NAME PIC X(6).
```

The form definition named by this parameter must be in the forms definition file for your transaction processor, and defined as an OUTPUT-ONLY type of form definition. See the *TRAX ATL Language Reference Manual* for a complete discussion of Report Forms.

status — The status array consists of two full-word computational data items used to return status information to the calling TST. The first item holds the return code; the second item is reserved for future use. Each data item is a full word.

The status data items can be defined as:

```
01  STATUS-WORDS.
03  STATUS-WORD-1  PIC S9(4) COMP.
03  STATUS-WORD-2  PIC S9(4) COMP.
```

Example of COBOL Usage:
In the following example, the TST is sending a report message containing a customer record to an output-only terminal station "LA1801", using the report form definition "PRTCUS".

The example shows the COBOL assignment statements and calling sequence used to send a report message.

```
MOVE "LA180S" TO STATION-NAME.
MOVE "PRTCUS" TO FORM-NAME.
MOVE 205 TO BUFFER-SIZE,
CALL "REPORT" USING WS-CUSTOMER-FILE-RECORD,
                    BUFFER-SIZE,
                     STATION-NAME,
                      FORM-NAME,
                       STATUS-WORDS.
              IF STATUS-WORD-1 IS NOT EQUAL TO 1
                  GO TO STATUS-ERROR-ABORT.
```

4.2.2 Using the REPORT Routine from BASIC-PLUS-2 TSTs

Calling Parameters:

The parameters you must specify in the REPORT calling sequence are listed in the order they are referenced by the REPORT routine.

Parameter	Description and Use
buffer	The name of the buffer containing data to be sent to the report form. Report message data should be specified as a string variable.
buffer size	The length (in bytes) of the message. The buffer size parameter must be an integer type variable or literal, and must contain the exact length of the message.
station name	The 6-character ASCII name of the output-only terminal station where you want the report printed. This is the station name assigned in the STADEF utility. The station name should be specified as a string variable or as a quoted literal text string.
form name	The 6-character ASCII name of the form definition that is to be used to format the report. The form name should be specified as a string variable or quoted literal text string. The form definition named by this parameter must be in the forms definition file for your transaction processor, and defined as an OUTPUT-ONLY type of form definition. See the *TRAX ATL Language Reference Manual* for a complete discussion of Report Forms.
status	The status parameter is a two-word integer array where status information is returned to the calling TST. The first word holds the return code; the second word is reserved for future use.

Example of BASIC-PLUS-2 Usage:

In the following example, the TST is sending a report message containing a customer record to an output-only terminal station "LA1801", using the report form definition "PRTCUS". The example shows the BASIC-PLUS-2 calling sequence used to send a report message.

```
5000\  CALL REPORT BY REF                                              &
       (WS.CUSTOMER.FILE.RECORD$,205%,"LA1801","PRTCUS",STATUS%())     &
     \ IF STATUS%(0%) IS < 1 GOTO 19500
```

4.2.3 Library Routine Status Return Codes for BASIC and COBOL:

+1	The call to the library routine was successful
−12	The station name you specified does not exist. Check that the station is defined, and spelled correctly in the call.
−30	The parameter list specified a buffer outside the TSTs address space.
−44	The station name specified in the call is not an output-only terminal station. The report message was not sent. Specify the name of an output-only station to receive the report.
−52	Not enough memory resources remain to call the routine.

4.3 SENDING RESPONSE MESSAGES

Response messages are way that TSTs return information to the user. The six specific types of response messages that may be sent are:

REPLY	Alters the form associated with the current exchange according to the reply-number parameter specified in the call.
ABORT	Performs the same processing as the REPLY call. In addition, the transaction instance is aborted when the user presses any terminal function key.
PRCEED	Invokes the exchange specified as the subsequent action to the current exchange. If the current exchange is defined as a repeating exchange, the current exchange is reentered. Otherwise the terminal station displays the initial form, the first form of the current transaction type, or the form of the next exchange depending on the parameters in the current exchange definition.
STPRPT	Is identical to PRCEED except that any repeat specification for the current exchange is ignored. The message always invokes the exchange specified in the subsequent action parameter of the transaction definition.
CLSTRN	Closes the current transaction instance, and returns the terminal to its initial state.
TRNSFR	Invokes the exchange specified in the parameter list.

The following sections describe the response message library routines in the context of terminal initiated transaction. These routines may also be used for non-terminal initiated transactions such as link- or batch-initiated transactions.

The nature of the initiating station determines the way data sent with the response message is formatted. In the case of terminal stations, the data is formatted according to a form definition. In the case of other transactions, the data is forwarded to the initiating station without modification.

See Chapter 5 for a discussion of batch-initiated transactions, and Chapter 6 for a discussion of link-initiated transactions.

4.3.1 The REPLY Routine

Sending A Response Message to Reenter the Current Exchange

Description: The REPLY routine sends a response message to the form currently displayed at the initiating station.

A terminal receiving a reply response message remains in the current exchange of the same transaction instance.

The form display resulting from a reply response message is determined by the contents of the optional data buffer, and the value of the reply number parameter specified in the REPLY call.

For example, if you call the REPLY routine, and specify the number 2 as the reply-number parameter in the calling sequence, then the current form is modified according to the clauses specified in the REPLY = 2 statement in the current form definition.

4.3.1.1 Using the REPLY Routine from COBOL TSTs:

Calling Parameters:

The following list describes the REPLY routine parameters in the order they must appear in the call.

Parameter	Description and Use
buffer	The name of the buffer containing data to be sent back to the initiating station. If no data is supplied, a buffer name must still be specified. The buffer should be a display data item in the DATA DIVISION. For example: 01 REPLY-MESSAGE-BUFFER PIC X(80).
buffer size	The length (in bytes) of the message. If no data is supplied, a value of zero must be specified for this parameter. The buffer-size parameter must be a data item of type PIC S9(4) COMP. The data item is used to specify this parameter in the DATA DIVISION of the example TST: 01 BUFFER-SIZE PIC S9(4) COMP.
reply-number	The number of the form definition reply statement that describes the modifications to be made when this message is received at the initiating station. The reply number specified for this parameter must be in the range 1 to 64 (decimal). An example of the data item used to specify this parameter is: 01 REPLY-NUMBER PIC S9(4) COMP.

status — The status array consists of two full-word computational data items used to return status information to the calling TST. The first item holds the return code; the second item is reserved for future use.

The status data items can be defined as:

```
01  STATUS-WORDS.
    03  STATUS-WORD-1  PIC S9(4) COMP.
    03  STATUS-WORD-2  PIC S9(4) COMP.
```

Example of COBOL Usage:
In the following example, the TST has detected a telephone number that is missing, or has invalid characters. The REPLY routine is called to send a reply response message, consisting of an error line, back to the application initiating station.

The example shows the COBOL assignment statements and calling sequence used to send a reply response message containing the error line to the initiating station.

```
MOVE "Incorrect Telephone Number" TO REPLY-MESSAGE-BUFFER.
MOVE 80 TO BUFFER-SIZE.
MOVE 2 TO REPLY-NUMBER.
CALL "REPLY" USING
REPLY-MESSAGE-BUFFER,BUFFER-SIZE,REPLY-NUMBER,STATUS-WORDS.
                    IF STATUS-WORD-1 IS NOT EQUAL TO 1
                            GO TO STATUS-ERROR ABORT.
```

4.3.1.2 Using the REPLY Routine from BASIC-PLUS-2 TSTs

Calling Parameters:

The parameters you must specify in the REPLY calling sequence are listed in the order they are referenced by the REPLY routine.

Parameter	Description and Use
buffer	The name of the buffer containing data to be sent back to the initiating station. If no data is supplied, a buffer name must still be specified. Response message data should be specified as a string variable.
buffer size	The length (in bytes) of the message. If no data is to be sent with the message, a value of zero must be specified for this parameter. The buffer size parameter must be an integer type variable, and must contain the exact length of the message.
reply-number	The number of the form definition reply statement that describes the modifications to be made when this message is received at the initiating station. The reply number specified for this parameter must be in the range 1 to 64 (decimal).

status — The status parameter is a two-word integer array where status information is returned to the calling TST. The first word holds the return code; the second word is reserved for future use.

Example of BASIC-PLUS-2 Usage:
In the following example, the TST has detected a telephone number that is missing, or has invalid characters. The REPLY routine is called to send a reply response message, consisting of an error line, back to the initiating station.

The example shows the BASIC-PLUS-2 assignment statements and calling sequence used to send a reply response message containing the error line to the initiating station.

```
5000   REPLY.BUFFER$ = "Incorrect Telephone Number"                          &
\      CALL REPLY BY REF (REPLY.BUFFER$,LEN%(REPLY.BUFFER$),2%,STATUS%())&
\            IF STATUS%(0%) < 1% GOTO 19500
```

4.3.1.3 Library Routine Status Return Codes for BASIC and COBOL:

+1 — The call to the library routine was successful

−30 — The parameter list specified a buffer outside the TSTs address space.

−52 — Not enough memory resources remain to call the routine.

4.3.2 The ABORT Routine

Sending a Response Message to Abort the Current Transaction Instance

Description: The ABORT routine sends an abort response message to the initiating station associated with the current exchange. The following events occur when the ABORT routine is called:

- A abort response message (which may contain user-supplied data) is sent to the initiating station.
- The terminal station modifies the current form display using the form definition REPLY specifications that correspond to the reply-number parameter specified in the ABORT call.
- After the modified form is displayed on the terminal, the user presses any enabled function key. When an enabled function key is pressed, the transaction instance is aborted.
- The user's terminal screen then shows the initial form defined for that terminal station.

Calling the ABORT routine automatically deletes any remaining routes for the current exchange message.

4.3.2.1 Using the ABORT Routine from COBOL TSTs:

Calling Parameters:

The following list describes the ABORT routine parameters in the order they must appear in the call.

Parameter	Description and Use
buffer	The name of the buffer containing data to be sent back to the initiating station. If no data is supplied, a buffer name must still be specified. The buffer should be a display data item in the DATA DIVISION. For example: 01 REPLY-MESSAGE-BUFFER PIC X(80).
buffer size	The length (in bytes) of the message. If no data is supplied, a value of zero must be specified for this parameter. The buffer-size parameter must be a data item of type PIC S9(4) COMP. The data item is used to specify this parameter in the DATA DIVISION of the example TST: 01 BUFFER-SIZE PIC S9(4) COMP.
reply-number	The number of the form definition reply statement that describes the modifications to be made when this message is received at the terminal station. The reply number specified for this parameter must be in the range 1 to 64 (decimal). An example of the data item used to specify this parameter is: 01 REPLY-NUMBER PIC S9(4) COMP.
status	The status array consists of two full-word computational data items used to return status information to the calling TST. The first item holds the return code; the second item is reserved for future use. The status data items can be defined as: 01 STATUS-WORDS. 03 STATUS-WORD-1 PIC S9(4) COMP. 03 STATUS-WORD-2 PIC S9(4) COMP.

Example of COBOL Usage:

In the following example, the TST has detected a hard record lock and is unable to continue processing the current transaction instance. The form definition has a specific reply screen defined to handle this error, REPLY #3. No data is sent from the TST to the initiating station, since the error message description has been incorporated into the form definition.

The ABORT routine is called to send an abort response message back to the initiating station. This response message causes the modifications specified in REPLY #3 of the current form definition to be made to the form display.

The example shows the COBOL assignment statements and calling sequence required to abort the transaction instance and display REPLY #3:

```
MOVE 0 TO BUFFER-SIZE.
MOVE 3 TO REPLY-NUMBER.
CALL "ABORT" USING
REPLY-MESSAGE-BUFFER,BUFFER-SIZE,REPLY-NUMBER,STATUS-WORDS.
                    IF STATUS-WORD-1 IS NOT EQUAL TO 1
                        GO TO STATUS-ERROR-ABORT.
```

4.3.2.2 Using the ABORT Routine from BASIC-PLUS-2 TSTs

Calling Parameters:

Parameter	Description and Use
buffer	The name of the buffer containing data to be sent back to the initiating station. If no data is supplied, a buffer name must still be specified. Response message data should be specified as a string variable. A buffer must be specified even when no data is to be returned to the initiating station.
buffer size	The length (in bytes) of the message. If no data is to be sent with the message, a value of zero must be specified for this parameter. The buffer size parameter must be an integer type variable, and must contain the exact length of the message. This value is best obtained by specifying a MAP statement to structure the buffer, and specifying the exact value as a literal value in the parameter list.
reply-number	The number of the form definition reply statement that describes the modifications to be made when this message is received at the initiating station. The reply number specified for this parameter must be in the range 1 to 64 (decimal).
status	The status parameter is a two-word integer array where status information is returned to the calling TST. The first word holds the return code; the second word is reserved for future use.

Example of BASIC-PLUS-2 Usage:

In the following example, the TST has detected a hard record lock and is unable to continue processing the current transaction instance. The form definition has a specific reply screen defined to handle this error, REPLY #3. No data is sent from the TST to the form, since the error message description has been incorporated into the form definition. The ABORT routine is called to send a reply response message back to the terminal station. This response message causes the modifications specified in REPLY #3 of the current form definition to be made to the form display.

The example shows the BASIC-PLUS-2 assignment statements and calling sequence required to abort the transaction instance and display REPLY #3:

```
5000 CALL ABORT BY REF (" ",0%,3%,STATUS%())                    &
\            IF STATUS%(0%) < 1% GOTO 19500
```

4.3.2.3 Library Routine Status Return Codes for BASIC and COBOL:

+1	The call to the library routine was successful
–30	The parameter list specified a buffer outside the TSTs address space.
–52	Not enough memory resources are available to call the routine.

4.3.3 The PRCEED Routine

Sending a Response Message to Proceed to the Next Exchange

Description: The PRCEED library routine sends a response message that directs the initiating station to proceed to the "next" exchange defined for the transaction.

If the current exchange is defined with the REPEAT exchange parameter, then the "next" exchange is a repeated execution of the current exchange.

If the current exchange is non-repeating, then the "next" exchange is determined by the subsequent action parameter specified in the current exchange definition.

- If the subsequent action is INITIAL, then the transaction instance is closed. The next form displayed is the initial form defined for the terminal station.
- If the subsequent action is FIRST, then the transaction instance is closed. The next form displayed is the form defined for the first exchange of the current transaction.
- If the subsequent action is NEXT, processing continues with the next exchange defined for the current transaction. The form associated with the next exchange is displayed on the terminal.

4.3.3.1 Using the PRCEED Routine from COBOL TSTs

Calling Parameters:

The following list describes the PRCEED routine parameters in the order they must appear in the call.

Parameter	Description and Use
buffer	The name of the buffer containing data to be sent to the initiating station. If no data is supplied, a parameter must still be specified. The buffer must be a display data item in the DATA DIVISION.

buffer size — The length (in bytes) of the message. If no data is supplied, a value of zero must be specified for this parameter.

The buffer size parameter must be a data item of type PIC S9(4) COMP. The following COBOL data item shows how you can define the buffer size parameter in the DATA DIVISION of a COBOL TST.

```
01  BUFFER-SIZE  PIC S9(4) COMP.
```

status — The status array consists of two full-word computational data items used to return status information to the calling TST. The first item holds the return code; the second item is reserved for future use.

The status data item can be defined in the DATA DIVISION as:

```
01  STATUS-WORDS.
    03  STATUS-WORD-1  PIC S9(4) COMP.
    03  STATUS-WORD-2  PIC S9(4) COMP.
```

Example of COBOL Usage
The following PRCEED message serves to send the customer record data to the next exchange.

```
MOVE 173 TO BUFFER-SIZE
CALL "PRCEED" USING CUSTOMER-FILE-RECORD,BUFFER-SIZE,STATUS-WORDS.
                    IF STATUS-WORD-1 IS NOT EQUAL TO 1
                        GO TO STATUS-ERROR-ABORT.
```

4.3.3.2 Using the PRCEED Routine from BASIC-PLUS-2 TSTs

Calling Parameters:

The following list describes the PRCEED routine parameters in the order they must appear in the call.

Parameter	Description and Use
buffer	The name of the buffer containing data to be sent back to the initiating station. If no data is supplied, a parameter must still be specified. The buffer must be a string variable.
buffer size	The length (in bytes) of the message. If no data is supplied, a value of zero must be specified for this parameter. In a BASIC-PLUS-2 TST, the buffer size parameter must be an integer type variable, and must contain the exact length of the message.
status	The status parameter is a two-word integer array where status information is returned to the calling TST. The first word holds the return code; the second word is reserved for future use.

Example of BASIC-PLUS-2 Usage
The following PRCEED message call sends a response message containing customer record data to the next exchange.

```
6000 CALL PRCEED BY REF(CUSTOMER.RECORD$,173%,STATUS%())          &
\            IF STATUS%(0%) < 1% GOTO 19500
```

4.3.3.3 Library Routine Status Return Codes for BASIC and COBOL:

+1	The call to the library routine was successful
–30	The parameter list specified a buffer outside the TSTs address space.
–52	Not enough memory resources are available to call the routine.

4.3.4 The STPRPT Routine

Sending a Response Message to Proceed Past a Repeated Exchange

This call directs the initiating station to disregard any existing REPEAT option for the current exchange, and proceed to the next exchange.

Description: The STPRPT library sends a response message to the "next" exchange defined for a transaction. Any REPEAT specification for the current exchange is disregarded.

The "next" exchange is determined by the subsequent action parameter specified in the current exchange definition.

- If the subsequent action is INITIAL, then the transaction instance is closed. The next form displayed is the initial form defined for the terminal station.
- If the subsequent action is FIRST, then the transaction instance is closed. The next form displayed is the form defined for the first exchange of the current transaction.
- If the subsequent action is NEXT, processing continues with the next exchange defined for the current transaction. The form associated with the next exchange is displayed on the terminal.

4.3.4.1 Using the STPRPT Routine from COBOL TSTs

Calling Parameters:

The following list describes the STPRPT routine parameters in the order they must appear in the call.

Parameter	Description and Use
buffer	The name of the buffer containing data to be sent to the initiating station. If no data is supplied, a parameter must still be specified. The buffer must be a display data item in the DATA DIVISION.

buffer size — The length (in bytes) of the message. If no data is supplied, a value of zero must be specified for this parameter.

The buffer size parameter must be a data item of type PIC S9(4) COMP. The following COBOL data item shows how you can define the buffer size parameter in the DATA DIVISION of a COBOL TST.

```
01  BUFFER-SIZE  PIC S9(4) COMP.
```

status — The status array consists of two full-word computational data items used to return status information to the calling TST. The first item holds the return code; the second item is reserved for future use.

The status data item can be defined in the DATA DIVISION as:

```
01  STATUS-WORDS.
    03  STATUS-WORD-1  PIC S9(4) COMP.
    03  STATUS-WORD-2  PIC S9(4) COMP.
```

Example of COBOL Usage
The following STPRPT message serves to send the customer record data retrieved from the file to the next exchange.

```
MOVE 173 TO BUFFER-SIZE
CALL "STPRPT" USING CUSTOMER-FILE-RECORD,BUFFER-SIZE,STATUS-WORDS.
                    IF STATUS-WORD-1 IS NOT EQUAL TO 1
                        GO TO STATUS-ERROR-ABORT.
```

4.3.4.2 Using the STPRPT Routine from BASIC-PLUS-2 TSTs

Calling Parameters:

The following list describes the STPRPT routine parameters in the order they must appear in the call.

Parameter — Description and Use

buffer — The name of the buffer containing data to be sent back to the initiating station. If no data is supplied, a buffer name must still be specified. This buffer must be a string variable.

buffer size — The length (in bytes) of the message. If no data is supplied, a value of zero must be specified for this parameter.

In a BASIC-PLUS-2 TST, the buffer size parameter must be an integer type variable, and must contain the exact length of the message.

status — The status parameter is a two-word integer array where status information is returned to the calling TST. The first word holds the return code; the second word is reserved for future use.

Example of BASIC-PLUS-2 Usage
The following STPRPT message serves to send the customer record data retrieved from the file to the next exchange.

```
6000 CALL STPRPT BY REF(CUSTOMER.RECORD$,173%,STATUS%())          &
\            IF STATUS%(0%) < 1% GOTO 19500
```

4.3.4.3 Library Routine Status Return Codes for BASIC and COBOL:

+1	The call to the library routine was successful
–30	The parameter list specified a buffer outside the TSTs address space.
–52	Not enough memory resources are available to call the routine.

4.3.5 The CLSTRN Routine

Sending a Response Message to Close the Transaction Instance

Description: The CLSTRN library routine sends a response message to the initiating station that closes (normally terminates) the current transaction instance.

The next form displayed on the terminal is the initial form.

4.3.5.1 Using the CLSTRN Routine from COBOL TSTs

Calling Parameters:

The following list describes the CLSTRN routine parameters in the order they must appear in the call.

Parameter	Description and Use
buffer	The name of the buffer containing data to be sent back to the initiating station. If no data is supplied, a buffer name must still be specified. This buffer must be a display data item in the DATA DIVISION.
buffer size	The length (in bytes) of the message. If no data is supplied, a value of zero must be specified for this parameter. The buffer size parameter must be a data item of type PIC S9(4) COMP. The following COBOL data item shows how you can define the buffer size parameter in the DATA DIVISION of a COBOL TST. `01 BUFFER-SIZE PIC S9(4) COMP.`
status	The status array consists of two full-word computational data items used to return status information to the calling TST. The first item holds the return code; the second item is reserved for future use.

The status data item can be defined in the DATA DIVISION as:

```
01  STATUS-WORDS.
    03  STATUS-WORD-1  PIC S9(4) COMP.
    03  STATUS-WORD-2  PIC S9(4) COMP.
```

Example of COBOL Usage
The following CLSTRN call closes a transaction instance and returns the initiating station to its initial state.

```
MOVE ZERO TO BUFFER-SIZE
CALL "CLSTRN" USING CUSTOMER-FILE-RECORD,BUFFER-SIZE,STATUS-WORDS.
                    IF STATUS-WORD-1 IS NOT EQUAL TO 1
                        GO TO STATUS-ERROR-ABORT.
```

4.3.5.2 Using the CLSTRN Routine from BASIC-PLUS-2 TSTs

Calling Parameters:

The following list describes the CLSTRN routine parameters in the order they must appear in the call.

Parameter	Description and Use
buffer	The name of the buffer containing data to be sent back to the initiating station. This buffer must be a string variable or a literal. If no data is supplied, a buffer parameter must still be specified.
buffer size	The length (in bytes) of the message. If no data is supplied, a value of zero must contain the exact length of the message.
status	The status parameter is a two-word integer array where status information is returned to the calling TST. The first word holds the return code; the second word is reserved for future use.

Example of BASIC-PLUS-2 Usage
The following CLSTRN call closes the transaction instance and returns the initiating station to it's initial state.

```
6000  CALL CLSTRN BY REF(" ",0%,STATUS%())
 \              IF STATUS%(0%) < 1% GOTO 19500
```

4.3.5.3 Library Routine Status Return Codes for BASIC and COBOL:

+1	The call to the library routine was successful
–30	The parameter list specified a buffer outside the TSTs address space.
–52	Not enough memory resources are available to call the routine.

4.3.6 The TRNSFR Routine

Sending a Response Message to Transfer to a Named Exchange

Description: The TRNSFR routine sends a response message directing the initiating station to begin execution of the exchange specified in the call.

If the initiating station is an interactive terminal station, the form associated with the new exchange is displayed, and data may be sent as part of the response message

4.3.6.1 Using the TRNSFR Routine from COBOL TSTs

Calling Parameters:

The following list describes the TRNSFR routine parameters in the order they must appear in the call.

Parameter	Description and Use
buffer	The name of the buffer containing data to be sent back to the initiating station. If no data is supplied, a buffer name must still be specified. This buffer must be a display data item in the DATA DIVISION.
buffer size	The length (in bytes) of the message. If no data is supplied, a value of zero must be specified for this parameter. The buffer size parameter must be a data item of type PIC S9(4) COMP. The following COBOL data item shows how you can define the buffer size parameter in the DATA DIVISION of a COBOL TST. 01 BUFFER-SIZE PIC S9(4) COMP.
exchange name	The 6-character ASCII name of exchange where control is being transferred to. The following data item in DATA DIVISION can be used to define this parameter: 01 EXCHANGE-NAME PIC X(6).
status	The status array consists of two full-word computational data items used to return status information to the calling TST. The first item holds the return code; the second item is reserved for future use. The status data item can be defined in the DATA DIVISION as: 01 STATUS-WORDS. 03 STATUS-WORD-1 PIC S9(4) COMP. 03 STATUS-WORD-2 PIC S9(4) COMP.

Example of COBOL Usage
The following TRNSFR message serves to send the customer record data retrieved from the file to the form defined for the EDIT exchange.

```
MOVE "EDIT" TO EXCHANGE-NAME.
MOVE 173 TO BUFFER-SIZE.
CALL "TRNSFR" USING CUSTOMER-FILE-RECORD,
                    BUFFER-SIZE,
                     EXCHANGE-NAME,
                      STATUS-WORDS.
                IF STATUS-WORD-1 IS NOT EQUAL TO 1
                   GO TO STATUS-ERROR-ABORT.
```

4.3.6.2 Using the TRNSFR Routine from BASIC-PLUS-2 TSTs

Calling Parameters:

The following list describes the TRNSFR routine parameters in the order they must appear in the call.

Parameter	Description and Use
buffer	The name of the buffer containing data to be sent back to the initiating station. This buffer must be a string variable or literal. If no data is supplied, a buffer parameter must still be specified.
buffer size	The length (in bytes) of the message. If no data is supplied, a value of zero must be specified for this parameter. In a BASIC-PLUS-2 TST, the buffer size parameter must be an integer type variable, and must contain the exact length of the message.
exchange name	The 6-character ASCII name of exchange where control is being transferred to. In BASIC-PLUS-2, this parameter is specified as either a string variable or literal. Using a variable and assigning exchange names to it may save you from having to write several different TRNSFR calls in the same TST.
status	The status parameter is a two-word integer array where status information is returned to the calling TST. The first word holds the return code; the second word is reserved for future use.

Example of BASIC-PLUS-2 Usage
The following TRNSFR message serves to send the customer record data retrieved from the file to the form defined for the EDIT exchange.

```
6000  IF KEY.CAP$="ENTER" THEN EXCHANGE$="EDIT"                    &
\       GO TO 6200                                                  &
                •
                •
                •
6200  CALL TRNSFR BY REF(CUSTOMER.RECORD$,173%,EXCHANGE$,STATUS%()) &
\               IF STATUS%(0%) < 1% GOTO 19500
```

4.3.6.3 Library Routine Status Return Codes for BASIC and COBOL:

+1	The call to the library routine was successful
–30	The parameter list specified a buffer outside the TSTs address space.
–52	Not enough memory resources are available to call the routine.

4.4 THE RESTRT ROUTINE – RESTARTING AN EXCHANGE

Description: The RESTRT routine directs the transaction processor to restart the current exchange.

If the transaction definition containing the exchange is not defined with exchange recovery, calling the RESTRT routine aborts the transaction instance, and an error is written to the software error log.

If the transaction type has been defined with exchange recovery, all resources (connected file streams, sent messages, record locks, and staged records) held by the current exchange are released, and the exchange message is reloaded and sent to the first station on the exchange routing list.

Calling the RESTRT Routine bypasses the remaining statements in the TST. In order to avoid looping through a succession of RESTRT calls until exchange timeout, you should check for the number of restarts in your TST. You can find the number of restarts by calling the GETSTN, GETRAN, or GETSRC routine immediately prior to calling RESTRT. The second status word contains the number of times the exchange has been restarted.

4.4.1 Using the RESTRT Routine from COBOL TSTs

The RESTRT routine requires only one parameter:

status — The status array consists of two full-word computational data items used to return status information to the calling TST. The first item holds the return code; second item is reserved for future use.

You can define the status data items in the DATA DIVISION of a COBOL TST as:

```
01  STATUS-WORDS.
    03  STATUS-WORD-1  PIC S9(4) COMP.
    03  STATUS-WORD-2  PIC S9(4) COMP.
```

Example of COBOL Usage:

The following calling sequence restarts the current exchange if the conditions for exchange recovery are present in the transaction definition.

```
CALL "RESTRT" USING STATUS-WORDS.
```

4.4.2 Using the RESTRT Routine from BASIC-PLUS-2 TSTs

The RESTRT routine requires only one parameter:

status — The status parameter is a two-word integer array where status information is returned to the calling TST. The first word holds the return code; the second word is reserved for future use.

Example of BASIC-PLUS-2 Usage:
The following calling sequence restarts the current exchange if the conditions for exchange recovery are present in the transaction definition.

```
CALL RESTRT BY REF (STATUS%())
```

4.4.3 Library Routine Status Return Codes for BASIC and COBOL:

NOTE

If the RESTRT routine is called successfully, no status value is returned to the TST. If the RESTRT routine is unsuccessful, the transaction instance is aborted.

4.5 THE TSPAWN ROUTINE – SPAWNING A TRANSACTION FROM WITHIN A TST

TRAX allows a TST to "spawn" or initiate another independent transaction instance. Once spawned, the new transaction instance processes concurrently with all existing transaction instances.

To spawn another transaction, the source station TST must supply an exchange message to the first exchange of the spawned transaction. The spawned transaction must have only one exchange.

4.5.1 Spawning a Transaction Instance – The TSPAWN Routine

Description: A TST calls the TSPAWN Routine to initiate an independent single exchange transaction. The calling sequence specifies the exchange message to be processed by the spawned transaction.

The spawned transaction instance operates asynchronously from the current transaction instance, and benefits from all the transaction recovery features available in the transaction processing environment.

4.5.1.1 Using the TSPAWN Routine from COBOL TSTs

Calling Parameters:

The following list describes the TSPAWN routine parameters and the order they must be used in the call.

Parameter — Description and Use

transaction name — The name of the transaction type that you want to spawn. This parameter must be defined as a 6-character ASCII data item. For example:

```
01  TRANSACTION-NAME  PIC X(6).
```

buffer — The data name of the exchange message you want processed by the spawned transaction instance.

buffer size — The size (in bytes) of the exchange message. This parameter must be defined as a computational data item as shown here:

```
01  BUFFER-SIZE  PIC S9(4) COMP.
```

instance number — An internal identifier by which the transaction instance is known to the system. If the call to TSPAWN successfully spawns a new transaction instance, the transaction instance number is returned to the calling TST in the data item specified for this parameter.

The transaction instance number is required when you want to use TABORT to abort a spawned transaction instance.

This parameter should be defined as a double computational variable, for example:

```
INSTANCE-NUMBER  PIC S9(9) COMP.
```

status — The status array consists of two full-word computational data items used to return status information to the calling TST. The first item holds the return code; the second item is reserved for future use.

You can define the status array in the DATA DIVISION as:

```
01  STATUS-WORDS.
    03  STATUS-WORD-1  PIC S9(4) COMP.
    03  STATUS-WORD-2  PIC S9(4) COMP.
```

Example of COBOL Usage

The following example shows the calling sequence used to spawn an instance of the transaction "UPDATE" using the customer record as an exchange message.

```
MOVE "UPDATE" TO TRANSACTION-NAME,
MOVE 205 TO BUFFER-SIZE.
CALL "TSPAWN" USING TRANSACTION-NAME,
                    CUSTOMER-FILE-RECORD,
                      BUFFER-SIZE,
                        INSTANCE-NUMBER,
                          STATUS-WORDS.
IF STATUS-WORD-1 IS NOT EQUAL TO 1
    GO TO STATUS-ERROR-ABORT.
```

4.5.1.2 Using the TSPAWN Routine from BASIC-PLUS-2 TSTs

Calling Parameters:

The following list describes the TSPAWN routine parameters and the order they must be used in the call.

Parameter	Description and Use
transaction name	The 6-character name of the transaction type that you want to spawn. This parameter should be specified as a string variable or literal. The example at the end of this section shows a literal value being used in the calling sequence.
buffer	This parameter generally consists of the data name of the exchange message you want processed by the spawned transaction. The data name should be a string type variable.
buffer size	The size (in bytes) of the exchange message. This parameter should be specified as an integer variable or literal.
instance number	An internal identifier by which the transaction instance is known to the system. If the call to TSPAWN successfully spawns a new transaction instance, the transaction instance number is returned to the calling TST in the data item specified for this parameter. The transaction instance number is required when you want to use the TABORT routine to abort a spawned transaction instance. This parameter should be defined as a two-word integer array.
status	The status parameter is a two-word integer array where status information is returned to the calling TST. The first word holds the return code; the second word is reserved for future use.

Example of BASIC-PLUS-2 Usage

The following example shows the calling sequence use to spawn an instance of the transaction type "UPDATE". Customer Record data is specified as an exchange message of length 205.

```
5000  CALL TSPAWN BY REF ("UPDATE",CUSTOMER.RECORD$,"205%",I%(),STATUS%()) &
\                   IF STATUS%(0%) < 1% GO TO 19500
```

4.5.1.3 Library Routine Status Return Codes for BASIC and COBOL:

+1	The call to the library routine was successful
–10	The transaction name specified in the call does not exist.
–30	An argument specified a buffer outside the TSTs address space.

–36	The library routine attempted to spawn a transaction type that has been disabled. (The transaction processor on your system manager can disable a transaction that does not function properly.)
–52	Not enough memory is available to execute the routine.
–56	The library routine encountered a fatal system error in its attempt to spawn the transaction. The transaction was not spawned.

4.5.2 The TABORT Routine

Aborting a Spawned or Current Transaction Instance

Description: The TABORT routine should be used to abort a transaction only when one of the following circumstances exist:

- A spawned transaction must be aborted
- The ABORT routine has failed, and you must abort the current transaction instance.
- A response message has already been sent.

4.5.2.1 Using the TABORT Routine from COBOL TSTs – Two parameters are required when calling TABORT:

instance number

If you are aborting a spawned transaction, this parameter must be the data item containing this value as returned by the TSPAWN call (see Section 4.3.1).

If you are aborting the current transaction instance, this parameter must contain a value of zero. If you specify zero for the instance number, the TABORT routine will abort the current transaction instance, and will not return status to the TST that issued the call. The initiating station is returned to its initial state by the system.

This parameter should be defined as a two-word computational data item. For example:

```
INSTANCE-NUMBER  PIC S9(9) COMP.
```

status

The status array consists of two full-word computational data items used to return status information to the calling TST. The first item holds the return code; the second item is reserved for future use.

As you can define the status array in the DATA DIVISION as:

```
01  STATUS-WORDS.
    03  STATUS-WORD-1  PIC S9(4) COMP.
    03  STATUS-WORD-2  PIC S9(4) COMP.
```

Example of COBOL Usage
The example shown here aborts a spawned transaction instance using TABORT.

```
CALL "TABORT" USING INSTANCE-NUMBER, STATUS-WORDS.
```

4.5.2.2 Using the TABORT Routine from BASIC-PLUS-2

Calling Parameters:

Two parameters are required when calling TABORT:

instance number — If you are aborting a spawned transaction, this parameter must be the variable containing this value as returned by the TSPAWN call (see Section 4.3.1).

If you are aborting the current transaction instance, this parameter must contain a value of zero. If you specify zero for the instance number, the TABORT routine will abort the current transaction instance, and will not return status to the TST that issued the call. The initiating station is returned to its initial state by the system.

This parameter must be defined as a two-word integer array.

status — The status parameter is a two-word integer array where status information is returned to the calling TST. The first word holds the return code; the second word is reserved for future use.

Example of BASIC-PLUS-2 Usage
The example shown here calls TABORT to abort a spawned transaction instance.

```
5000    CALL TABORT BY REF (I%(),STATUS%())
```

4.5.2.3 Library Routine Status Return Codes for BASIC and COBOL

+1 — The spawned transaction instance was successfully aborted.

–46 — The library routine specified a non-existent transaction instance value, or the transaction instance value identified a transaction that was not spawned by the TST that called the TABORT routine.

NOTE

If the TABORT routine is called successfully to abort oneself, the transaction instance is aborted immediately. Hence no status value is returned.

4.6 ROUTING LIST CONTROL

Every exchange in a transaction definition has its own routing list. The routing list consists of up to 8 station names. The exchange message is processed by each of the stations in the routing list, in the order that the stations appear in the transaction definition.

At the time an exchange is initiated, the TRAX executive copies the routing list (a maximum of 8 entries) from the transaction definition to a temporary area.

Three TST library routines, AROUTE,DROUTE, and DALLRT are available for your use in dynamically altering routing list characteristics of the current exchange. The action performed by these routines applies only to the current transaction instance, and in no way affects the original definition of the exchange.

4.6.1 The AROUTE Routine

Adding a Station to An Exchange Routing List

Description: The AROUTE routine is used to temporarily add a station to the end of the current exchange routing list. This allows a TST to change the normal routing of an exchange message. The size of the route list at any time during the execution of the exchange is limited to eight (8) remaining destination entries.

4.6.1.1 Using the AROUTE routine from COBOL TSTs – AROUTE requires that you specify the following two parameters, in order.

station name — The 6-character ASCII name of the station you want to add to the end of the current exchange routing list. The station name parameter should be defined in the DATA DIVISION as:

```
01  STATION-NAME  PIC X(6).
```

status — The status array consists of two full-word computational data items used to return status information to the calling TST. The first item holds the return code; the second item is reserved for future use.

You can define the status array in the DATA DIVISION as:

```
01  STATUS-WORDS.
    03  STATUS-WORD-1  PIC S9(4) COMP.
    03  STATUS-WORD-2  PIC S9(4) COMP.
```

Example of COBOL Usage

The following example shows a COBOL call of AROUTE to add the station "CLSOUT" to the end of the current exchange routing list.

```
MOVE "CLSOUT" TO STATION-NAME.
CALL "AROUTE" USING STATION-NAME, STATUS-WORDS.
IF STATUS-WORD-1 IS NOT EQUAL TO 1
GO TO STATUS-ERROR-ABORT.
```

4.6.1.2 Using the AROUTE routine from BASIC-PLUS-2 TSTs – AROUTE requires that you specify the following two parameters, in order.

station name — The 6-character ASCII name of the station you want to add to the end of the current exchange routing list. This parameter can be specified as a string variable, or as a literal value in the parameter list.

status — The status parameter is a two-word integer array where status information is returned to the calling TST. The first word holds the return code; the second word is reserved for future use.

Example of BASIC-PLUS-2 Usage
The following example shows a BASIC-PLUS-2 call of AROUTE that adds "CLSOUT" to the end of the current exchange routing list.

```
5000 CALL AROUTE BY REF ("CLSOUT",STATUS%())                    &
\          IF STATUS%(0%) < 1% GO TO 19500
```

4.6.1.3 Library Routine Status Return Codes for BASIC and COBOL:

+1 — The call to the library routine was successful

−12 — The station name you specified is not defined for this transaction processor; or you attempted to add a terminal or mailbox station to the routing list.

−14 — The route list associated with this exchange already contains eight station names.

4.6.2 The DROUTE Routine
Deleting a Station from An Exchange Routing List

Description: The DROUTE routine allows you to delete one station from the current exchange routing list. If the station name specified in the call appears more than once in the route list, the first remaining station by that name is deleted.

4.6.2.1 Using the DROUTE routine from COBOL TSTs – DROUTE requires that you specify the following two parameters, in order.

station name — The 6-character ASCII name of the station you want to delete from the current exchange routing list. The station name parameter should be defined in the DATA DIVISION as:

```
01  STATION-NAME  PIC X(6).
```

status — The status array consists of two full-word computational data items used to return status information to the calling TST. The first item holds the return code; the second item is reserved for future use.

You can define the status array in the DATA DIVISION as:

```
01  STATUS-WORDS.
    03  STATUS-WORD-1  PIC S9(4) COMP.
    03  STATUS-WORD-2  PIC S9(4) COMP.
```

Example of COBOL Usage
The following example shows a COBOL call of DROUTE to delete the station "CLSOUT" from the current exchange routing list.

```
MOVE "CLSOUT" TO STATION-NAME.
CALL "DROUTE" USING STATION-NAME, STATUS-WORDS.
IF STATUS-WORD-1 IS NOT EQUAL TO 1
GO TO STATUS-ERROR-ABORT.
```

4.6.2.2 Using the DROUTE routine from BASIC-PLUS-2 TSTs – DROUTE requires that you specify the following two parameters, in order.

station name — The 6-character ASCII name of the station you want to delete from the current exchange routing list. This parameter can be specified as a string variable, or as a literal value in the parameter list.

status — The status parameter is a two-word integer array where status information is returned to the calling TST. The first word holds the return code; the second word is reserved for future use.

Example of BASIC-PLUS-2 Usage
The following example shows a BASIC-PLUS-2 call of DROUTE that deletes "CLSOUT" from the current exchange routing list.

```
5000  CALL DROUTE BY REF ("CLSOUT",STATUS%())                    &
\           IF STATUS%(0%) < 1% GO TO 19500
```

4.6.2.3 Library Routine Status Return Codes for BASIC and COBOL:

+1 — The call to the library routine was successful

−16 — The station named in the call does not appear in the routing list.

4.6.3 The DALLRT Routine
Deleting All Remaining Stations from the Exchange Routing List

Description: The DALLRT routine deletes all remaining stations from the current exchange routing list. The exchange terminates at the conclusion of processing by the current TST station.

4.6.3.1 Using the DALLRT routine from COBOL TSTs – DALLRT requires that you specify one parameter:

status — The status array consists of two full-word computational data items where status information is returned to the calling TST. The first word holds the return code; the second word is reserved for future use.

You can define the status array in the DATA DIVISION as:

```
01  STATUS-WORDS.
    03  STATUS-WORD-1  PIC S9(4) COMP.
    03  STATUS-WORD-2  PIC S9(4) COMP.
```

Example of COBOL Usage
The following example shows a COBOL call of DALLRT to delete all remaining routes from the current exchange routing list.

```
CALL "DALLRT" USING STATUS-WORDS.
IF STATUS-WORD-1 IS NOT EQUAL TO 1
GO TO STATUS-ERROR-ABORT.
```

4.6.3.2 Using the DALLRT routine from BASIC-PLUS-2 TSTs – DALLRT requires that you specify one parameter:

status — The status parameter is a two-word integer array where status information is returned to the calling TST. The first word holds the return code; the second word is reserved for future use.

Example of BASIC-PLUS-2 Usage
The following example shows a BASIC-PLUS-2 call of DALLRT that deletes all remaining stations from the current exchange routing list.

```
5000  CALL DALLRT BY REF (STATUS%())                    &
\     IF STATUS%(0%) < 1% GO TO 19500
```

4.6.3.3 Library Routine Status Return Codes for BASIC and COBOL:

+1 — The call to the library routine was successful

4.7 USING MAILBOX STATIONS FROM A TST

Mailbox stations are used to store data that is useful to later transaction instances.

A mailbox station resides on the system disk and can be accessed only from a TST. The messages deposited at a mailbox station are retrieved in a first-in, first-out sequence.

Three system library routines are available for accessing mailboxes: SNDMBX sends a message to a mailbox station, GETMBX retrieves a message from a mailbox, and MBXNUM returns the number of messages at the specified mailbox.

4.7.1 The SNDMBX Routine

Sending a Message to a Mailbox Station

Description: The SNDMBX routine sends the message specified in the parameter list to the specified mailbox station.

4.7.1.1 Using the SNDMBX Routine from COBOL TSTs

Calling Parameters:

The parameters you must specify in the SNDMBX calling sequence are listed in the order they are referenced by the SNDMBX routine.

Parameter	Description and Use
buffer	The name of the buffer containing the message to be stored in the mailbox. The buffer should be a character data item. For example: 01 MAILBOX-BUFFER. 03 MBX-BUF-CUST-NO PIC X(6). 03 MBX-BUF-CRED-LIM PIC S9(10)V99. 03 MBX-BUF-CURR-BAL PIC S9(10)V99. 03 MBX-BUF-TOD-DATE PIC S9(6). 03 MBX-BUF-STA-ID PIC X(6).
buffer size	The size (in bytes) of the mailbox message. The buffer size parameter must be a data item of the type PIC S9(4) COMP. An example specification of this data item would be: 01 BUFFER-SIZE PIC S9(4) COMP.
station name	The name (6 character ASCII) of the mailbox station receiving this message. The station name must be assigned to a character data item with PIC X(6). For example: 01 STATION-NAME PIC X(6).
status	The status array consists of two full-word computational data items used to return status information to the calling TST. The first item holds the return code; the second item is reserved for future use. You can define the status data items as: 01 STATUS-WORDS. 03 STATUS-WORD-1 PIC S9(4) COMP. 03 STATUS-WORD-2 PIC S9(4) COMP.

Example of COBOL Usage:
The example shows the COBOL calling sequence used to send a mailbox message containing the customer data to a mailbox station called OVRCRL.

```
MOVE STATION-ID TO MBX-BUF-STA-ID.
MOVE "OVRCRL" TO STATION-NAME.
MOVE 42 TO BUFFER-SIZE.
CALL "SNDMBX" USING MAILBOX-BUFFER,BUFFER-SIZE,STATION-NAME,STATUS-WORDS.
                IF STATUS-WORD-1 IS NOT EQUAL TO 1
                    GO TO STATUS-ERROR-ABORT.
```

4.7.1.2 Using the SNDMBX Routine from BASIC-PLUS-2 TSTs

Calling Parameters:

The parameters you must specify in the SNDMBX calling sequence are listed in the order they are referenced by the SNDMBX routine.

Parameter	Description and Use

buffer	The name of the buffer containing the message to be stored in the mailbox.
	
	The buffer should be a string variable. For example:
	
	`950 \  MAP (MBXBUF)  &`
	`  MAILBOX.BUFFER$ = 42  &`
	`!  &`
	`  \  MAP (MBXBUF)  &`
	`  MBX.BUF.CUST.NO$ = 6  &`
	`'  MBX.BUF.CRED.LIM$ = 12  &`
	`'  MBX.BUF.CURR.BAL$ = 12  &`
	`'  MBX.BUF.TOD.DATE$ = 6  &`
	`'  MBX.BUF.STA.ID$ = 6`

buffer size	The size (in bytes) of the mailbox message. The buffer size parameter must be an integer data item or literal. The value you specify must exactly match the number of characters in the message.
station name	The name (6 character ASCII) of the mailbox station receiving this message. The station name must be a string variable or quoted literal. If your TST references several mailboxes, assigning the mailbox name to a string variable allows you to have a common calling sequence in your TST. If the station name is less than six characters, you must insure that the padding spaces are inserted in the same positions used to define the station name in the STADEF utility.
status	The status parameter is a two-word integer array where status information is returned to the calling TST. The first word holds the return code; the second word is reserved for future use.

Example of BASIC-PLUS-2 Usage:
The example shows the BASIC-PLUS-2 calling sequence used to construct and send a mailbox message containing the customer data to a mailbox station called OVRCRL.

```
12000 \ STATION.NAME$ = "OVRCRL"                                          &
      \ CALL SNDMBX BY REF (MAILBOX.BUFFER$,42%,STATION.NAME$,STATUS%()) &
      \          IF STATUS%(0%) < 1% GO TO 19500
```

4.7.1.3 Library Routine Status Return Codes for BASIC and COBOL:

+1	The call to the library routine was successful
−12	The station specified in the call is not known to the system.
−30	The buffer specified in the call is outside the TSTs address space.
−44	The station specified in the call is not a mailbox station
−52	Not enough memory resources are available to call the routine.

4.7.2 The GETMBX Routine

Retrieving a Message from a Mailbox Station

Description: The GETMBX Routine retrieves a message from a mailbox station.

4.7.2.1 Using the GETMBX Routine from COBOL TSTs

Calling Parameters:

The parameters you must specify in the GETMBX calling sequence are listed in the order they are referenced by the GETMBX routine.

Parameter	Description and Use
buffer	The name of the buffer where you want the routine to store the returned mailbox message. The buffer should be a character data item. For example: 01 MAILBOX-BUFFER. 03 MBX-BUF-CUST-NO PIC X(6). 03 MBX-BUF-CRED-LIM PIC S9(10)V99. 03 MBX-BUF-CURR-BAL PIC S9(10)V99. 03 MBX-BUF-TOD-DATE PIC S9(6). 03 MBX-BUF-STA-ID PIC X(6).
buffer size	The size (in bytes) of the buffer receiving the mailbox message. The buffer size parameter must be a data item of the type PIC S9(4) COMP. An example specification of this data item would be: 01 BUFFER-SIZE PIC S9(4) COMP.

station name — The name (6 character ASCII) of the mailbox station from which you are retrieving a message.

The station name must be assigned to a character data item with PIC X(6). If the station name is less than six characters, you must insure that the padding spaces are inserted in the same positions used to define the station name in the STADEF utility. An example of the definition of this data item would be:

```
01  STATION-NAME  PIC X(6).
```

status — The status array consists of two full-word computational data items used to return status information to the calling TST. The first item holds the return code; if the call is successful, the second data item contains the size (in bytes) of the message obtained from the mailbox station.

You can define the status data items as:

```
01  STATUS-WORDS.
    03  STATUS-WORD-1  PIC S9(4) COMP.
    03  STATUS-WORD-2  PIC S9(4) COMP.
```

Example of COBOL Usage:
The example shows the COBOL and calling sequence used to retrieve a mailbox message from a mailbox station called OVRCRL.

```
MOVE "OVRCRL" TO STATION-NAME.
MOVE 42 TO BUFFER-SIZE.
CALL "GETMBX" USING MAILBOX-BUFFER,BUFFER-SIZE,STATION-NAME,STATUS-WORDS.
        IF STATUS-WORD-1 IS NOT EQUAL TO 1
            GO TO STATUS-ERROR-ABORT.
```

4.7.2.2 Using the GETMBX Routine from BASIC-PLUS-2 TSTs.

Calling Parameters:

The parameters you must specify in the GETMBX calling sequence are listed in the order they are referenced by the GETMBX routine.

Parameter	Description and Use
buffer	The name of the buffer where you want the routine to store the message it gets from the mailbox station.

The buffer should be a string variable. For example:

```
950 \  MAP (BMXBUF)                                    &
       MAILBOX.BUFFER$ = 42                            &
!                                                      &
    \  MAP (MBXBUF)                                    &
         MBX.BUF.CUST.NO$ = 6                          &
,        MBX.BUF.CRED.LIM$ = 12                        &
,        MBX.BUF.CURR.BAL$ = 12                        &
,        MBX.BUF.TOD.DATE$ = 6                         &
,        MBX.BUF.STA.ID$ = 6
```

buffer size — The size (in bytes) of the mailbox message.

The buffer size parameter must be an integer data item or literal. The value you specify must exactly match the number of characters in the receiving buffer.

station name — The name (6 character ASCII) of the mailbox station from which you are retrieving a mailbox message.

The station name must be a string variable or quoted literal. If your TST references several mailboxes, assigning the mailbox name to a string variable allows you to have a common calling sequence in your TST. If the station name is less than six characters, you must insure that the padding spaces are inserted in the same positions used to define the station name in the STADEF utility.

status — The status parameter is a two-word integer array where status information is returned to the calling TST. The first word holds the return code; the second word is reserved for future use.

Example of BASIC-PLUS-2 Usage:
The example shows the BASIC-PLUS-2 calling sequence used to retrieve a mailbox message from a mailbox station called OVRCRL.

```
12000 \    STATION.NAME$ = "OVRCRL"                                          &
      \    CALL GETMBX BY REF (MAILBOX.BUFFER$,42%,STATION.NAME$,STATUS()) &
               IF STATUS%(0%) < 1% GO TO 19500
```

4.7.2.3 Library Routine Status Return Codes for BASIC and COBOL:

+1 — The call to the library routine was successful

−12 — The station specified in the call is not known to the system.

−30 — The buffer specified in the call is outside the TSTs address space.

–44	The station specified in the call is not a mailbox station
–52	Not enough memory resources are available to call the routine.
–54	No message queued at specified mailbox station.

4.7.3 The MBXNUM Routine

Determine the Number of Messages Stored at a Mailbox Station

Description: The MBXNUM routine returns the number of messages that are currently stored in the mailbox.

4.7.3.1 Using the MBXNUM Routine from COBOL TSTs

Calling Parameters:

The parameters you must specify in the MBXNUM calling sequence are listed in the order they are referenced by the MBXNUM routine.

Parameter	Description and Use
buffer	The location where the routine is to return the count of the number of messages in the mailbox. This data item must be in the form PIC S9(4) COMP. For example: 01 MAILBOX-MSG-COUNT PIC S9(4) COMP.
station name	The name (6 character ASCII) of the mailbox station you are interrogating. The station name must be assigned to a character data item with PIC X(6). If the station name is less than six characters, you must insure that the padding spaces are inserted in the same positions used to define the station name in the STADEF utility. An example of the definition of this data item would be: 01 STATION-NAME PIC X(6).
status	The status array consists of two full-word computational data items used to return status information to the calling TST. The first item holds the return code; the second item is reserved for future use. You can define the status data items as: 01 STATUS-WORDS. 03 STATUS-WORD-1 PIC S9(4) COMP. 03 STATUS-WORD-2 PIC S9(4) COMP.

Example of COBOL Usage:

In the following example, you want to interrogate the mailbox called OVRCRL to see how many messages are there. This information is needed to determine if enough credit limits have been exceeded to warrant calling the accounting department.

The example shows the COBOL assignment statements and calling sequence used to ask the mailbox OVRCRL to return the number of messages it currently has in its queue.

```
MOVE "OVRCRL" TO STATION-NAME.
CALL "MBXNUM" USING MAILBOX-MSG-COUNT,STATION-NAME,STATUS-WORDS.
                IF STATUS-WORD-1 IS NOT EQUAL TO 1
                  GO TO STATUS-ERROR-ABORT.
```

4.7.3.2 Using the MBXNUM Routine from BASIC-PLUS-2 TSTs.

Calling Parameters:

The parameters you must specify in the MBXNUM calling sequence are listed in the order they are referenced by the MBXNUM routine.

Parameter	Description and Use
buffer	The location where the routine is to return the count of the number of messages in the mailbox. This parameter must be an integer variable such as MAILBOX. MSG.COUNT%.
station name	The name (6 character ASCII) of the mailbox station receiving this message. The station name must be a string variable or quoted literal.
status	The status parameter is a two-word integer array where status information is returned to the calling TST. The first word holds the return code; the second word is reserved for future use.

Example of BASIC-PLUS-2 Usage:

The example shows the BASIC-PLUS-2 calling sequence used to ask the mailbox OVRCRL to return the number of messages it currently has in its queue.

```
12000 \ STATION.NAME$ = "OVRCRL"                                          &
      \ CALL MBXNUM BY REF (MAILBOX.MSG.COUNT%,STATION,NAME$,STATUS%()) &
      \          IF STATUS%(0%) < 1% GO TO 19500
```

4.7.3.3 Library Routine Status Return Codes for BASIC and COBOL:

+1	The call to the library routine was successful
−12	The station specified in the call is not known to the system.
−30	The buffer specified in the call is outside the TSTs address space.
−44	The station specified in the call is not a mailbox station
−52	Not enough memory resources are available to call the routine.

4.8 SYSTEM INFORMATION ROUTINES

TRAX provides you with a set of five system information routines that allow you to obtain data concerning stations, data files, and transactions as well as the date and time. The system information routines return the following data to your TST.

Routine	*Data Returned*
GETIME	Time and Date
GETSTN	TST Station ID
GETSRC	Initiating Station ID
GETRAN	Transaction ID
GETFIL	Physical File Specification

4.8.1 The GETIME Routine – Determining the Current Time and Date

Description: The GETIME routine allows you to obtain date and time information from the system.

4.8.1.1 Using the GETIME routine from COBOL TSTs

Calling Parameters:

The parameters you must specify in the GETIME calling sequence are listed in the order they are referenced by the GETIME routine.

Parameter — Description and Use

buffer — The GETIME call returns eight data items to your TST. The data structure that describes the information returned by GETIME is:

```
01  DATE-FIELDS.
    03  YEAR-SINCE-1900          PIC S9(4) COMP.
    03  MONTH-OF-YEAR            PIC S9(4) COMP.
    03  DAY-OF-MONTH             PIC S9(4) COMP.
    03  HOUR-OF-DAY              PIC S9(4) COMP.
    03  MINUTES-PAST-HOUR        PIC S9(4) COMP.
    03  SECONDS-PAST-MINUTE      PIC S9(4) COMP.
    03  TICKS-THIS-SECOND        PIC S9(4) COMP.
    03  CLOCK-TICKS-PER-SEC      PIC S9(4) COMP.
```

where:

YEAR . . . contains the number of years since 1900.

MONTH . . . contains the month as a number from 1 to 12.

DAY . . . contains the day as a number from 1 to 31.

HOUR . . . contains the hour as a number from 0 to 23.

MINUTES . . . contains the minute as a number from 0 to 59.

SECONDS . . . contains the second as a number from 0 to 59.

TICKS . . . contains the number of ticks since the last full second.

CLOCK . . . contains the number of clock ticks in a second. The number of clock ticks depends upon your hardware configuration.

status The status array consists of two full-word computational data items used to return status information to the calling TST. The first item holds the return code, the second item is reserved for future use.

The status data items can be defined as:

```
01  STATUS-WORDS.
    03  STATUS-WORD-1  PIC S9(4) COMP.
    03  STATUS-WORD-2  PIC S9(4) COMP.
```

Example of COBOL Usage:
In the following example, a TST calls GETIME to determine the time and date of a transaction.

```
CALL "GETIME" USING DATE-FIELDS,STATUS-WORDS.
IF STATUS-WORD-1 IS NOT EQUAL TO 1
GO TO STATUS-ERROR-ABORT.
```

When the call is made in this manner, each individual data item can be examined if you supply a data structure similar to that shown in the "buffer" parameter description.

4.8.1.2 Using the GETIME routine from BASIC-PLUS-2 TSTs

Calling Parameters:

The parameters you must specify in the GETIME calling sequence are listed in the order they are referenced by the GETIME routine.

Parameter Description and Use

buffer The GETIME call returns eight integer values to your TST. The data structure that describes the information returned by GETIME is:

```
951 MAP (TIMER) YEAR%, MONTH%, DAY%,                    &
    HOUR%, MINUTES%, SECONDS%, TICKS%, CLOCK%
```

where:

YEAR% contains the number of years since 1900.

MONTH% contains the month as a number from 1 to 12.

DAY%	contains the day as a number from 1 to 31.
HOUR%	contains the hour as a number from 0 to 23.
MINUTES%	contains the minute as a number from 0 to 59.
SECONDS%	contains the second as a number from 0 to 59.
TICKS%	contains the number of ticks since the last full second.
CLOCK%	contains the number of clock ticks in a second. The number of clock ticks depends upon the configuration of your hardware.
status	The status parameter is a two-word integer array where status information is returned to the calling TST. The first word holds the return code; the second word is reserved for future use.

Example of BASIC-PLUS-2 Usage:
In the following example, a TST calls GETIME to determine the time and date of a transaction.

```
5000 CALL GETIME BY REF(YEAR%,STATUS%())                    &
\       IF STATUS%(0%) < 1% GO TO 19500
```

When the call is made in this manner, each individual data item can be examined if you supply a data structure similar to that shown in the map statement (TIMER). Specifying the variable "YEAR%" causes the date and time to be returned in the eight contiguous variables starting with YEAR%.

4.8.1.3 Library Routine Status Return Codes for BASIC and COBOL:

+1	The call to the library routine was successful
–30	The buffer specified in the parameter list was outside the TSTs address space.

4.8.2 The GETSTN Routine – Determining the Current TST Station Name

Description: The GETSTN routine allows you to obtain the 6-character name of the current TST station. GETSTN returns the station name into a buffer specified in the parameter list.

4.8.2.1 Using the GETSTN Routine from COBOL TSTs

Calling Parameters:

GETSTN requires you to specify two parameters:

Parameter	Description and Use
buffer	The buffer parameter must specify a data item defined to hold a 6-character ASCII station name. The GETSTN routine returns the station name in this data item. For example:

```
01  STATION-NAME  PIC X(6).
```

status — The status array consists of two full-word computational data items used to return status information to the calling TST. The first item holds the return code, the second item is reserved for future use.

The status data items can be defined as:

```
01  STATUS-WORDS.
    03  STATUS-WORD-1  PIC S9(4) COMP.
    03  STATUS-WORD-2  PIC S9(4) COMP.
```

Example of COBOL Usage:
In the following example, a TST calls GETSTN to determine the name of the TST station that is currently processing the exchange message.

```
CALL "GETSTN" USING STATION-NAME, STATUS-WORDS.
IF STATUS-WORD-1 IS NOT EQUAL TO 1
GO TO STATUS-ERROR-ABORT.
```

4.8.2.2 Using the GETSTN routine from BASIC-PLUS-2 TSTs

Calling Parameters:

The parameters you must specify in the GETSTN calling sequence are listed in the order they are referenced by the GETSTN routine.

Parameter	Description and Use
buffer	The buffer parameter must specify a data item defined to hold a 6-character ASCII station name. The GETSTN routine returns the station name in this variable. The variable specified as this parameter should be defined in a MAP or COMMON statement with a length of 6 characters. For example: `951 MAP (PARMS) STATION$=6, . . .`
status	The status parameter is a two-word integer array where status information is returned to the calling TST. The first word holds the return code; the second word is reserved for future use.

Example of BASIC-PLUS-2 Usage:
In the following example, a TST calls GETSTN to determine the name of the TST station that is processing the current exchange message.

```
5000 CALL GETSTN BY REF(STATION$,STATUS%())                    &
\       IF STATUS%(0%) < 1% GO TO 19500
```

4.8.2.3 Library Routine Status Return Codes for BASIC and COBOL:

+1	The call to the library routine was successful
–30	The buffer specified in the parameter list was outside the TSTs address space.

4.8.3 The GETSRC Routine – Determining the Initiating Station ID

Description: The GETSRC routine allows you to determine the 6-character ASCII identifier assigned to the station that initiated the current transaction instance (the source station).

GETSRC returns the station name into a buffer specified in the parameter list.

4.8.3.1 Using the GETSRC Routine from COBOL TSTs

Calling Parameters:

GETSRC uses two parameters:

Parameter	Description and Use
buffer	The buffer parameter must specify a data item defined to hold a 6-character ASCII station name. The GETSRC routine returns the station name in this data item. For example: 01 STATION-NAME PIC X(6).
status	The status array consists of two full-word computational data items used to return status information to the calling TST. The first item holds the return code, the second item contains the number of times the current exchange has been restarted. The status data items can be defined as: 01 STATUS-WORDS. 03 STATUS-WORD-1 PIC S9(4) COMP. 03 STATUS-WORD-2 PIC S9(4) COMP.

Example of COBOL Usage:
In the following example, a TST calls GETSRC to determine the name of the station that initiated the current transaction instance.

```
CALL "GETSRC" USING STATION-NAME, STATUS-WORDS.
IF STATUS-WORD-1 IS NOT EQUAL TO 1
GO TO STATUS-ERROR-ABORT.
```

4.8.3.2 Using the GETSRC routine from BASIC-PLUS-2 TSTs

Calling Parameters:

The parameters you must specify in the GETSRC calling sequence are listed in the order they are referenced by the GETSRC routine.

Parameter	Description and Use
buffer	The buffer parameter must specify a data item defined to hold a 6-character ASCII station name. The GETSRC routine returns the station name in this

variable. The variable specified as this parameter should be defined in a MAP or COMMON statement with a length of 6 characters. For example:

```
951 MAP (PARMS) STATION$=6, . . .
```

status — The status parameter is a two-word integer array where status information is returned to the calling TST. The first word holds the return code; the second word is reserved for future use.

Example of BASIC-PLUS-2 Usage:
In the following example, a TST calls GETSRC to determine the name of the station that initiated the current transaction instance.

```
5000 CALL GETSRC BY REF(STATION$,STATUS%())                    &
\    IF STATUS%(0%) < 1% GO TO 19500
```

4.8.3.3 Library Routine Status Return Codes for BASIC and COBOL:

+1 — The call to the library routine was successful

–30 — The buffer specified in the parameter list was outside the TSTs address space.

4.8.4 The GETRAN Routine – Determining the Transaction Type ID

Description: The GETRAN routine allows you to determine the 6-character ASCII identifier assigned to the transaction type in which your TST is executing. The transaction name is returned into a buffer specified in the parameter list.

4.8.4.1 Using the GETRAN Routine from COBOL TSTs

Calling Parameters:

GETRAN uses two parameters:

Parameter — Description and Use

buffer — The buffer parameter must specify a data item defined to hold a 6-character ASCII transaction name. The GETRAN routine returns the transaction name in this data item. For example:

```
01 TRANSACTION-NAME PIC X(6).
```

status — The status array consists of two full-word computational data items used to return status information to the calling TST. The first item holds the return code, the second item contains the number of times the current exchange has been restarted.

The status data items can be defined as:

```
01 STATUS-WORDS.
   03 STATUS-WORD-1 PIC S9(4) COMP.
   03 STATUS-WORD-2 PIC S9(4) COMP.
```

Example of COBOL Usage:
In the following example, a TST calls GETRAN to determine the name of the transaction type used in the current transaction instance.

```
CALL "GETRAN" USING TRANSACTION-NAME, STATUS-WORDS.
IF STATUS-WORD-1 IS NOT EQUAL TO 1
GO TO STATUS-ERROR-ABORT.
```

4.8.4.2 Using the GETRAN routine from BASIC-PLUS-2 TSTs

Calling Parameters:

The parameters you must specify in the GETRAN calling sequence are listed in the order they are referenced by the GETRAN routine.

Parameter	Description and Use
buffer	The buffer parameter must specify a data item defined to hold a 6-character ASCII transaction name. The GETRAN routine returns the transaction name in this variable. The variable specified as this parameter should be defined in a MAP or COMMON statement with a length of 6 characters. For example: 951 MAP (PARMS) TRANSACTION$=6, . . .
status	The status parameter is a two-word integer array where status information is returned to the calling TST. The first word holds the return code; the second word is reserved for future use.

Example of BASIC-PLUS-2 Usage:
In the following example, a TST calls GETRAN to determine the name of the transaction type used in the current transaction instance.

```
5000 CALL GETRAN BY REF(TRANSACTION$,STATUS%())                    &
\      IF STATUS%(0%) < 1% GO TO 19500
```

4.8.4.3 Library Routine Status Return Codes for BASIC and COBOL:

+1	The call to the library routine was successful
–30	The buffer specified in the parameter list was outside the TSTs address space.

4.8.5 The GETFIL Routine – Determining a Physical File Specification

Description: The GETFIL routine allows you to determine the physical file specification that corresponds to a logical file name currently accessed by this transaction instance. GETFIL returns the physical RMS file specification to a 33-character buffer that you must specify in the parameter list.

4.8.5.1 Using the GETFIL Routine from COBOL TSTs

Calling Parameters:

The following list describes the GETFIL parameters in the order they must be specified in the calling sequence.

Parameter	Description and Use
logical name	This parameter specifies a 6-character logical file name, padded with BLANKs if necessary. This name is the logical file name specified in the file definition record (See Chapter 12). The data item used to specify this parameter must be specified as shown in the following example: 01 LOGICAL-FILE-NAME PIC X(6).
buffer	The buffer parameter must specify a data item defined to hold a 33-character TRAX file specification. The GETFIL routine returns the physical file specification in this data item. For example: 01 FILE-SPEC-BUFFER PIC X(33).
status	The status array consists of two full-word computational data items used to return status information to the calling TST. The first item holds the return code, if the call is successful, the second item contains the number of characters in the file specification returned to the TST by GETFIL. The status data items can be defined as: 01 STATUS-WORDS. 03 STATUS-WORD-1 PIC S9(4) COMP. 03 STATUS-WORD-2 PIC S9(4) COMP.

Example of COBOL Usage:

In the following example, a TST calls GETFIL to determine the physical file specification associated with the logical file name "CUSTOM".

```
MOVE "CUSTOM" TO LOGICAL-NAME.
CALL "GETFIL" USING LOGICAL-FILE-NAME,
                    FILE-SPEC-BUFFER,
                      STATUS-WORDS.
IF STATUS-WORD-1 IS NOT EQUAL TO 1
  GO TO STATUS-ERROR-ABORT.
```

4.8.5.2 Using the GETFIL routine from BASIC-PLUS-2 TSTs

Calling Parameters:

The parameters you must specify in the GETFIL calling sequence are listed in the order they are referenced by the GETFIL routine.

Parameter	Description and Use
logical name	This parameter specifies a 6-character logical file name, padded with BLANKs if necessary. This name must correspond with a file definition record in the file [1,300] tpname.FIL. This data item is usually specified as a string variable or literal string in the parameter list. See the example following this section for an example of a literal string in the parameter list.
buffer	The buffer parameter must specify a data item defined to hold a 33-character TRAX file specification. The GETFIL routine returns the physical file specification associated with the logical file name into this buffer. The variable specified as this parameter should be defined in a MAP or COMMON statement with a length of 33 characters. For example: 951 MAP (PARMS) PHYS.FILE.SPEC$=33
status	The status parameter is a two-word integer array where status information is returned to the calling TST. The first word holds the return code; the second word is reserved for future use.

Example of BASIC-PLUS-2 Usage:
In the following example, a TST calls GETFIL to determine the physical file specification associated with the logical file name "CUSTOM".

```
5000 CALL GETFIL BY REF("CUSTOM",PHYS.FILE.SPEC$,STATUS%())          &
\      IF STATUS%(0%) < 1% GO TO 19500
```

4.8.5.3 Library Routine Status Return Codes for BASIC and COBOL:

+1	The call to the library routine was successful
–30	The buffer specified in the parameter list is outside the TSTs address space.
–64	The logical name parameter referred to a non-existent file.

4.9 LOGGING INFORMATION TO THE JOURNAL FILE

4.9.1 The LOGTRN Routine – Log Specified Data to the Journal

Description: The LOGTRN routine allows you to write data to the journal file.

The log records written by this routine can be interpreted using the SHOLOG utility. (See the TRAX System Manager's Guide.)

The log records written to the journal file by this routine are independent and should not be confused with the journalling of transaction slots.

NOTE

Records are written to the log in blocked form. If the system should crash, log records may be lost during the crash.

4.9.1.1 Using the LOGTRN Routine from COBOL TSTs;

Calling Parameters:

The following list describes the LOGTRN routine parameters in the order they must appear in the call.

Parameter	Description and Use
buffer	The name of the buffer containing data to be logged to the journal. The buffer should be a character data item in the DATA DIVISION. For example: 01 LOG-DATA-BUFFER 03 LOG-BUFFER-EM PIC X(36). 03 LOG-BUFFER-NOTE PIC X(80).
buffer size	The length (in bytes) of the data buffer. The buffer-size parameter must be a data item of type PIC S9(4) COMP. The data item is used to specify this parameter in the DATA DIVISION of the example TST: 01 BUFFER-SIZE PIC S9(4) COMP.
log code letter	A 1 character (ASCII) alphabetic field which contains a user defined code. This code identifies the specific type of log record that you are writing. It is used by the SHOLOG utility when it processes the logged records from the journal. You can identify up to 26 (A-Z) distinct types of logged messages using this code. The parameter must be defined in the DATA DIVISION as a single character data item: 01 LOG-CODE-LETTER PIC X(1)

status — The status array consists of two full-word computational data items used to return status information to the calling TST. The first item holds the return code; the second item is reserved for future use.

The status data items can be defined as:

```
01  STATUS-WORDS.
    03  STATUS-WORD-1  PIC S9(4) COMP.
    03  STATUS-WORD-2  PIC S9(4) COMP.
```

Example of COBOL Usage:
The example shows the COBOL assignment statements and calling sequence used to log a 116-character data buffer to the journal using the code letter "K".

```
MOVE EXCHANGE-MESSAGE TO LOG-BUFFER-EM.
MOVE STATION-ID-CODE TO LOG-BUFFER-NOTE.
MOVE "K" TO LOG-CODE-LETTER.
MOVE 116 TO BUFFER-SIZE.
CALL "LOGTRN" USING LOG-BUFFER,
                    BUFFER-SIZE,
                    LOG-CODE-LETTER,
                    STATUS-WORDS.
    IF STATUS-WORD-1 IS NOT EQUAL TO 1
      GO TO STATUS-ERROR-ABORT.
```

4.9.1.2 Using the LOGTRN Routine from BASIC-PLUS-2 TSTs

Calling Parameters:

The parameters you must specify in the LOGTRN calling sequence are listed in the order they are referenced by the LOGTRN routine.

Parameter	Description and Use
buffer	The name of the buffer containing data to be logged to the journal. Log data should be specified as a string variable. An example of buffer usage is given at the end of this section.
buffer size	The length (in bytes) of the data buffer. The buffer size parameter must be an integer type variable, and must contain the exact length of the data buffer. This value is best obtained by specifying the LEN%(buffer.name$) function in the parameter list. See the example at the end of this section for an illustration of how the LEN% function is used.

log code letter — a 1 character (ASCII) alphabetic field which contains a user defined code. This code identifies the specific type of log record that you are writing. It is used by the SHOLOG utility when it processes the logged records from the journal. You can identify up to 26 (A-Z) distinct types of logged messages using this code.

status — The status parameter is a two-word integer array where status information is returned to the calling TST. The first word holds the return code, the second word is reserved for future use.

Example of BASIC-PLUS-2 Usage:
The example shows the BASIC-PLUS-2 assignment statements and calling sequence used to log the information in a buffer called LOG.DATA.BUFFER$.

```
5000  CALL LOGTRN BY REF (LOG.BUFFER$,LEN%(LOG.BUFFER$),"K",STATUS%()) &
      \   IF STATUS%(0%) < 1% GO TO 19500
```

4.9.1.3 Library Routine Status Return Codes for BASIC and COBOL:

+1 — The library call was successful

−30 — The buffer specified in the parameter list is outside the TSTs address space.

−56 — A fatal system error was encountered by the routine. The message was not written to the journal file.

CHAPTER 5

USING BATCH FACILITIES WITH A TRANSACTION PROCESSOR

TRAX provides facilities for:

- Submitting a batch job from a transaction instance
- Initiating a Transaction Instance from a Batch Job.

TRAX Batch Facilities are documented in detail in the *TRAX Support Environment Programmer's Guide.* Before reading this chapter, it is suggested that you be familiar with the functionality of the batch processor.

5.1 SUBMITTING A BATCH JOB FROM A TRANSACTION INSTANCE

To submit a batch job, the transaction processor definition must include a submit batch station. The submit batch station must also be defined. (See the description of transaction processor definition in Chapter 9, and station definition in Chapter 10.)

A transaction instance submits a batch job in the following manner.

1. The submit batch station is included in the exchange routing list.
2. The exchange message must contain a valid DCL SUBMIT command. The first two bytes in the exchange message are defined as an integer field where you must store the number of characters in the SUBMIT command that follows it. (The SUBMIT command is documented in the *TRAX Support Environment Programmer's Guide.*)

   The following examples describe how to code the exchange message, and construct the SUBMIT command in COBOL and BASIC-PLUS-2. The SUBMIT command shown in this example starts a batch job using the command file [300,300] CRDLTR.CMD after 5 p.m. on the current date. If this batch job fails, the job is not restarted.

   COBOL Exchange Message

```
01  EXCHANGE MESSAGE,
    03  EM-CHARACTER-COUNT      PIC S9(4) COMP.
    03  EM-SUBMIT-COMMAND       PIC X(79).
```

   COBOL Command Specification

```
MOVE "SUBMIT/AFTER:(17:00)/NORESTART [300,300]CRDLTR"
    TO EM-SUBMIT-COMMAND.
MOVE 46 TO EM-CHARACTER-COUNT.
```

   BASIC-PLUS-2 Exchange Message

```
1000    MSGMAP EM.CHARACTER.COUNT%,EM.SUBMIT.COMMAND$=79
```

BASIC-PLUS-2 Command Specification

```
3000    EM.SUBMIT.COMMAND$="SUBMIT/AFTER:(17:00)/NORESTART" &
        + "[300,300] CRDLTR"&
\       EM.CHARACTER.COUNT%=46%
```

3. When the exchange message arrives at the submit batch station, the submit batch station forwards the SUBMIT command to the queue manager.
4. If the queue manager accepts the DCL SUBMIT command, then the submit batch station alters the first two characters of the exchange message to contain the characters SS. If the batch submission was rejected, then the first two characters in the exchange message contains SE.

NOTE

The only part of the batch submission performed by a transaction instance is sending an exchange message in the form of a SUBMIT command to the submit batch station. The entry into a queue and subsequent batch processing are performed in the support environment.

5.2 INITIATING A TRANSACTION FROM A BATCH JOB

Using a slave batch station, TRAX allows you to invoke a transaction instance from a running program in the support environment. The slave batch station can:

1. Invoke a single exchange transaction and initiate the exchange, using the data received from the batch program to create the exchange message.
2. Wait for a reply response message from a TST processing the exchange. When this message is received at the slave batch station, the transaction instance is closed and the first 24 characters of reply message data are sent to the support environment program that initiated the transaction.
3. If the transaction type was defined with a subsequent action parameter of NOWAIT, the transaction instance is closed, and a blank message sent to the support environment task that initiated the transaction instance.

5.3 INITIATE TRANSACTION – THE STTRAN ROUTINE

Description: The STTRAN Routine starts a transaction instance from any batch program. When your program calls STTRAN, it spawns a task to perform all required I/O and memory mapping. This task is named by replacing the first two letters of the initiating program with the letters ZZ. For this reason, the initiating task cannot have a task name that begins with ZZ.

The initiating task disables checkpointing until the message is accessed by the batch manager. Using STTRAN may affect performance in the support environment.

5.4 USING THE STTRAN ROUTINE FROM COBOL SUPPORT ENVIRONMENT PROGRAMS

Calling Parameters:

The following list describes the STTRAN routine parameters in the order they appear in the call.

Parameter	Description and Use
tpname	The 1- to 6-character name of the active transaction processor where the transaction instance is to be invoked. This parameter must be defined in a COBOL program as a 6-character data item. For example: 01 TXN-PROC-NAME PIC X(6).
transaction name	The transaction type you wish to invoke. This must be a single exchange transaction. This parameter must be defined in a COBOL program as a 6-character data item. For example: 01 TRANSACTION-NAME PIC X(6).
buffer	The location in the calling program where the exchange message data is stored. In COBOL programs, this parameter must identify a display data item or group item, such as: 01 TXN-EXCHANGE-MESSAGE. 03 (Exchange Message Data Structure) • • •
buffer size	You must specify an integer variable where your TST can place a value representing the number of characters of data in the exchange message buffer. In a COBOL program, this parameter must be defined as a computational data item. For example: 01 BUFFER-SIZE PIC S9(4) COMP.
return buffer	The location where the first 24 characters of the reply response message are placed when the transaction instance terminates. This location must accomodate at least 24 characters. You must define the data item describing this parameter in the following manner: 01 RETURN-BUFFER-AREA PIC X(24).
status	The status array consists of two data items. The first item contains status return information from the library routine. The second item is reserved for future use.

The status data items can be defined as:

```
01  STATUS-WORDS.
    03  STATUS-WORD-1 PIC S9(4) COMP.
    03  STATUS-WORD-2 PIC (S9(4) COMP.
```

Example of COBOL Usage:
In the following example, the transaction processor "SAMPLE" has a single exchange transaction called "CHKMBX"; it is designed to interrogate and report the number of messages stored in each system mailbox. The following call, placed in a COBOL batch program, causes the system to invoke CHKMBX.

```
MOVE "SAMPLE" TO TXN-PROC-NAME.
MOVE "CHKMBX" TO TRANSACTION-NAME.
MOVE "01" TO BUFFER-SIZE.
CALL "STTRAN" USING TXN-PROC-NAME,
                    TRANSACTION-NAME,
                     ST-EXCHANGE-MESSAGE,
                      BUFFER-SIZE,
                       RETURN-BUFFER-AREA,
                        STATUS-WORDS.
```

5.5 USING THE STTRAN ROUTINE FROM BASIC-PLUS-2 SUPPORT ENVIRONMENT PROGRAMS

Calling Parameters:

The following list describes the STTRAN routine parameters in the order they appear in the call.

Parameter	Description and Use
tpname	The 1- to 6-character name of the active transaction processor where the transaction instance will be invoked. This parameter must be defined in a BASIC-PLUS-2 program as a 6-character string variable or literal.
transaction name	The transaction type you wish to invoke. This must be a single exchange transaction and cannot have a form associated with it. This parameter must be defined in a BASIC-PLUS-2 program as a 6-character string variable or literal.
buffer	The location in the calling program where the exchange message data is stored. In BASIC-PLUS-2 programs, this parameter must identify a string variable containing exchange message data, or a literal value.
buffer size	This must be defined as an integer variable corresponding to the exact length of the data in the exchange message.

return buffer — The location where the first 24 characters of the reply response message are placed when the transaction intance terminates. This location must accomodate at least 24 characters. You must define the data name describing this area as a string variable.

status — The status array consists of two variables. The first element contains status return information from the library routine. The second element is reserved for future use.

The status elements can be defined as an integer array STATUS%. STATUS%(0%) contains the first status word, and STATUS%(1%) the second.

Example of BASIC-PLUS-2 Usage:
In the following example, the transaction processor "SAMPLE" has a single exchange transaction called "CHKMBX"; it is designed to interrogate and report the number of messages stored in each system mailbox. The following call, placed in a BASIC-PLUS-2 batch program, invokes "CHKMBX."

```
4000  CALL STTRAN BY REF ("SAMPLE", "CHKMBX", "  ",1%,RETURN.AREA$,
      STATUS%( ) )
```

5.6 LIBRARY ROUTINE STATUS RETURN CODES

+1	The transaction completed successfully.
-8	Internal directive error.
-30	Parameter validation error.
-36	The transaction name you specified is disabled. See your system manager to determine why it was disabled.
-48	The transaction processor named in the call is not running.
-56	An I/O error encountered during execution of transaction.
-60	Invalid transaction processor name.
-62	The started transaction terminated abnormally.

CHAPTER 6
COMMUNICATION BETWEEN TRANSACTION PROCESSORS TRAX/TL AND TRAX/3271-TL

TRAX/TL and TRAX/3271-TL are software modules that allow you to initiate a transaction in another transaction processor or IBM system. TRAX/TL (Transaction Link) permits a TRAX system to be connected to other TRAX systems. TRAX/TL allows an executing transaction within a TRAX system to initiate a transaction within the same or a physically different TRAX system.

TRAX/3271-TL is a protocol emulator that permits transactions running under TRAX to communicate interactively with tasks in an IBM 360 or 370 system running CICS/OS or CICS/VS.

This chapter is organized in two major sections. Section 6.1 describes the application programming techniques required to use TRAX/TL. Section 6.2 describes the application programming considerations for the TRAX side of TRAX/3271-TL.

6.1 TRAX/TL

To communicate between two TRAX transaction processors using TRAX/TL you must define a master link station in the sending transaction processor, and one or more slave link stations in the receiving transaction processor.

The master link station is a system provided TST which always overwrites the exchange message it receives.

A master link station in a transaction processor with TL uses a number of sub-links, which correspond to the number of slave link stations defined in the receiving transaction processor for use with the specified link. Sub-links allow a single master station to simultaneously control several slave transaction instances.

Processing across a link involves the following set of operations.

1. The transaction is invoked. Processing proceeds in the same manner as a normal transaction. A master link station is specified in the exchange routing list for the transaction.

NOTE

When you define the master transaction, specify the exchange message size as the sum of the 10-character header and the user data. When defining the slave transaction use the same exchange message size as you used in the master transaction definition.

2. The exchange message arriving at the master link station must be formatted into two sections; the message header and the data buffer. The message header contains control information for use by the master and slave link stations. The data buffer contains the data used as an exchange message by the slave transaction instance. The exchange message formatting

required by the master link station is normally performed by a TST preceding the master link station in the transaction routing list.

3. When an exchange message arrives at a master link station, the header is used to transmit the entire exchange message to the slave transaction processor.
4. The slave link station uses the information in the header to initiate the slave transaction named in the exchange message header. The slave link station creates an exchange message for this transaction from the section of the link message containing the data buffer. The transaction instance on the master side is suspended until the slave link station responds with a message.
5. The response message generated by a TST in the slave transaction instance is sent to the slave link station that initiated the transaction. The slave link station performs normal exchange control operations, and also forwards the response message to the master link station. Message area and performs the operations directed by the response message type. The master link station exits at this point. Exchange message processing continues at the next station in the routing list.

6.1.1 Operations from a Master Link Station

The four basic operations that can be performed over a link are:

1. Initiate a single exchange transaction. This type of processing requires that you specify the following data in the exchange message header:

   - A message type code of "I".
   - The slave transaction name as a 6-character string.
   - The number of characters of data following the header.

2. Initate the first exchange of a multiple exchange transaction. This type of processing requires that you specify the following data in the exchange message header:

   - A message type code of "R".
   - The slave transaction name as a 6-character string.
   - The number of characters of data following the header.

3. Continue with subsequent exchanges of a multiple exchange transaction. This type of processing requires that you specify the following data in the exchange message header.

   - A message type code of "C".
   - The number of characters of data following the header.

4. Abort a previously initiated transaction instance.

   - A message type code of "C".
   - A value of zero in the data length field.

The data structure and an example of formatting the message header is shown in Section 6.3.1 for COBOL, and 6.3.2 for BASIC-PLUS-2.

The exchange message created by the master link station consists of either a message received from a slave link station, or an error generated by the master link station explaining why it could not contact the slave link station.

The master link station constructs the exchange message in the following manner:

1. If TRAX/TL is down, a two-character status message (SL) replaces the original exchange message.
2. If the original master exchange message requested that the slave transaction be aborted, and the abort actually takes place, then a 2-character status message (SA) replaces the original message.
3. If the slave transaction sends a response message indicating that it has reached the end of (closed) a transaction, the original exchange message is replaced with the response message data as it was received from the slave transaction. If a sub-link was reserved by the master link, it is released at this time.
4. If the slave transaction sends a reply with abort message, the system aborts the current slave transaction instance and the reply with abort message (RA) is passed on to the next station in the master transaction routing list. The master transaction is not aborted.
5. If the response message from the slave does not abort or close the transaction, the response message data directly replaces the exchange message and the exchange message continues at the next station on the route list.

6.1.2 Preparing the Exchange Message for the Master Link Station

In a transaction that initiates a link using a master link station, the master station appears in the routing list of the exchange exactly as if it were a TST station.

In the TST that immediately precedes the master link station in the routing list, you must prepare an exchange message in the format expected by the master link station.

The header consists of an area message type code, a reserved area for system use, an area that contains the slave transaction name, and an area containing the number of characters of user data. The examples in the following section show the required field sizes and data types for the message header.

6.1.3 COBOL Master to Slave Message Format

In a COBOL TST, the following is an example of the format required for a message that is submitted to a master link station.

```
LINKAGE SECTION.
01 EXCHANGE-MESSAGE.

  02 LINK-HEADER.

    03 MESSAGE-TYPE-CODE          PIC X.
    03 FILLER                     PIC X.
    03 SLAVE-TRANSACTION-ID       PIC X(6).
    03 LINK-MESSAGE-LENGTH        PIC S9(4) COMP.

  02 LINK-EXCHANGE-MESSAGE.

    03 CUSTOMER-NUMBER            PIC X(6).
    03 CUSTOMER-NAME              PIC X(30).
```

If you use an exchange message similar to the one shown above, you must supply the header data in a TST preceding the master link station in the exchange routing list.

After you define the link message header in the LINKAGE SECTION of a TST, you must perform the following operations in that TST's PROCEDURE DIVISION:

- Move the appropriate type code (I,R,C or A) to the exchange message.
- Move the 6-character string identifying the slave transaction to the exchange message.
- Move the length of user data to be sent across the link into the exchange message.

The following MOVE statements set up the message header to initiate a single exchange transaction to retrieve a customer record. The data structure of the message header is shown at the beginning of this section.

```
MOVE "I" TO MESSAGE-TYPE-CODE.
MOVE "RDCUST" TO SLAVE-TRANSACTION-NAME.
MOVE 36 TO LINK-MESSAGE-LENGTH.
```

6.1.4 BASIC-PLUS-2 Master to Slave Message Format

In a BASIC-PLUS-2 TST, the following is an example of the format required in the MSGMAP statement immediately preceding the master link station in the exchange routing list.

```
600   \     MSGMAP   EM.MSG.TYPE.CODE$     =   1    &
            '        EM.FILLER$            =   1    &
            '        EM.SLAVE.TRANS.ID     =   6    &
            '        EM.LINK.MSG.LEN%          ,    &
            '        EM.CUSTOMER.NUMBER    =   6    &
            '        EM.CUSTOMER.NAME$     =   30
```

If you use an exchange message similar to the one shown above, you must supply the header data in a TST preceding the master link station in the exchange routing list.

After you define the link message header in the MSGMAP statement, you must perform the following operations

- Assign the appropriate message type code (I,R,C or A) to the type code field in the exchange message header.
- Assign the 6-character string identifying the slave transaction to the slave transaction field in the exchange message header.
- Calculate the length of user data to be sent across the link and assign this value to the message length field in the exchange message header.

The following BASIC-PLUS-2 statements set up the message header to initiate a single exchange transaction to retrieve a customer record. The data structure of the message header is shown at the beginning of this section.

```
1000  EM.MSG.TYPE.CODE$ = "I"                &
   \  EM.SLAVE.TRANS.ID$ = "RDCUST"          &
   \  EM.LINK.MSG.LEN% = 36%
```

6.1.5 Slave Link Station

Messages sent from a master link station over a sub-link to a slave transaction processor are directed to a slave link station. The slave link station can:

1. Initiate a transaction instance and enter the first exchange. The data received over the sub-link becomes the exchange message. The slave link station waits for a response message from one of the TST stations in the exchange routing list. Once a response message is received at the slave link station, the slave link station sends the response message header and data back to the master link station over the sub-link. The message returned by the slave to the master is a new message. It contains data supplied by the slave transaction. This message is not related to the exchange message sent by the master link station when it initiated the transaction.

   If the completed exchange is the last exchange in the slave transaction (or if the response message says to) the slave transaction instance is closed.

   If the exchange is not the last exchange, then the slave link station waits for the next link message from the master station.

2. Continue the transaction instance at the proper exchange. In this case, the slave link station uses the link message data to generate the exchange message for the new exchange. Then it waits for a response message from one of the TST stations in the exchange routing list and continues as outlined above.
3. Abort the currently open transaction.

6.1.5.1 Response Messages Sent to the Slave Link Station

A transaction initiated by a slave link station must send exactly one response message per exchange to the originating station.

The following response message types can be sent from a slave link station are: The text following the type identifier indicates the effect this routine has on both the master and slave transactions:

1. ABORT – The slave transaction is aborted.
2. STPRPT – Slave goes to next defined exchange. No effect on master transaction.
3. CLSTRN – Slave closes transaction. Sublink is released by master link station.
4. PRCEED – Slave continues according to transaction definition. No effect on master transaction.
5. TRNSFR – Slave continues according the transaction specified in call to TRNSFR routine. No effect on master transaction.

NOTE

Do not attempt to send any response message type other than those listed above. You cannot send a REPLY response message from a slave transaction.

When the response message is received at the slave link station, the entire response message is sent from slave to master. If the slave transaction is closed, the master releases the sublink automatically.

The message received back at the master link station contains an eight-byte header (placed there by the library routine that sends the response message) followed by the response message data from the slave transaction.

The 8 byte header in the response message sent by the slave is place in the first 8 bytes of the exchange message sent to the next TST in the exchange route list.

The received message does not contain any length indication. If variable-length data is being returned, the length must be included as part of the data.

The following table shows the response messages that can be sent by a slave transaction, and the data contained in the header:

Response Message	Message Type Code	Code Area Contents
ABORT	RA	Reply Number in third word.
PRCEED	CP	1 in third word.
STPRPT	CH	1 in third word.
CLSTRN	CC	1 in third word.
TRNSFR	FX	ASCII name of next exchange.

The following sections define typical COBOL and BASIC-PLUS-2 data structures to receive the header data in the master transaction.

COBOL Response Message Header Structure:

```
01  EXCHANGE-MESSAGE
    02  RESPONSE-MESSAGE-FROM-SLAVE
    03  MESSAGE-TYPE-CODE                  PIC XX.
    03  CODE-WORDS.
        05  CODE-WORD-1                    PIC S9(4).
        05  CODE-WORD-2                    PIC S9(4).
        05  CODE-WORD-3                    PIC S9(4).
    03  NEXT-EXCHANGE REDEFINES CODE-WORDS.
        05  EXCHANGE-NAME                  PIC X(6).
```

BASIC-PLUS-2 Response Message Header Structure:

```
\ MSGMAP                                                    &
            EM.SLAVE.RESP.MSG$ = 8                          &
\ MSGMAP                                                    &
            EM.MSG.TYP.CODE$ = 2                            &
    ,       EM.MSG.CODE.WORD1%                              &
    ,       EM.MSG.CODE.WORD2%                              &
    ,       EM.MSG.CODE.WORD3%                              &
\ MSGMAP                                                    &
            EM.FILLER$ = 2                                  &
    ,       EM.EXCHNG.NAME$ = 6
```

6.2 TRAX/3271-TL

TRAX/3271-TL allows a TRAX transaction processor to initiate processing in an IBM system running under CICS/DS or CICS/VS.

To use TRAX/3271-TL, you must define a master link station in the TRAX transaction processor. In the master link station definition, you must specify the IBM line number connected to this master link station.

TRAX always initiates processing across the link. Processing begins when a master transaction is invoked on the TRAX side. Typically, processing involves the following set of operations:

1. The master transaction is invoked. Processing proceeds in the same manner as a normal transaction. A master link station is defined in an exchange routing list in the master transaction.
2. In the TST preceding a master link station in the routing list, you assign values to the message header data structure. The message header must be located in the first 12 characters of the exchange message data structure.
3. When an exchange message is placed at a master link station, the header is used to transmit the data buffer to the IBM system.
4. The IBM system sees only the message data and the control characters in character positions 10 through 12.
5. After the IBM system completes processing of that exchange data, it sends a message across the link to the master link station.
6. The master link station places this message in the exchange message area and passes it to the next TST in the master (TRAX) exchange routing list for subsequent processing.

The application programmer must be concerned with three different message states:

1. The exchange message containing a header and a data buffer that is placed at the master link station.
2. The data buffer received by the IBM system. This data must be structured for the IBM program processing the data.
3. The message data sent by the IBM system back to the master link station. This message serves as the exchange message for subsequent TSTs in the current exchange of the master transaction.

6.2.1 Master Link Stations

The IBM system receiving the link message has no exchange structure. Reserving a sub-link to an IBM system reserves that resource so that future exchanges are processed upon receipt at the master link station.

A master link station can perform several types of processing. The character preceding each description in the following list must be specified in the message type code field of the exchange message header received by the master link station.

1. I (INITIATE-RELEASE) – The line is released upon receipt of a message back from the IBM system.
2. R (INITIATE-RESERVE) – The master link station connects the line when it sends data to the IBM system. This line is retained until a type E message is received at the master link station. (See code E below.)
3. C (CONTINUE) – The master link can continue sending exchange message data through a previously reserved line to an IBM system. Do not specify C if you are sending data to the last exchange in an IBM slave transaction.
4. E (RELEASE) – The message for the last exchange (or the last exchange that you want resources reserved for) must contain this code to explicitly release the line at the conclusion of this exchange.

The exchange message created by the master link station when TRAX/3271-TL is down consists of a two-character status message (SL), which replaces the original exchange message.

6.2.2 Preparing the Exchange Message for the Master Link Station

In a transaction that initiates a link using a master link station, the master station appears in the routing list of the exchange exactly as if it were a TST station.

In a TST preceding (usually the TST immediately before) the master link station in the routing list, you must prepare an exchange message in the format expected by the master link station.

Data routed to an IBM station must be in the format expected by the IBM system. The message must begin with an attention identifier (AID) followed by a two-character cursor address. If the IBM terminal that TRAX is emulating requires any type of formatting support, then the TRAX application program must be prepared to supply and receive 3277 format protocols. The master link station does not examine or supply any such data.

Data sent by TRAX to an IBM system may consist of printable graphic ASCII characters that can be translated into EBCDIC. All other byte values are translated to EBCDIC spaces before they are sent to IBM. EBCDIC data received in a TRAX system that translates to non-graphic ASCII characters is converted to ASCII spaces.

6.2.3 COBOL Master to Slave Message Format

In a COBOL TST, the following is an example of the format required for a message that is submitted to a master link station for transmission to an IBM system using TRAX/3271-TL.

```
LINKAGE SECTION.
01  EXCHANGE-MESSAGE.
    02  LINK-HEADER.
        03  MESSAGE-TYPE-CODE          PIC X.
        03  FILLER                     PIC X.
        03  FILLER                     PIC X(6).
        03  LINK-MESSAGE-LENGTH        PIC S9(4) COMP.
        03  ATTENTION-IDENTIFIER       PIC X.
        03  IBM-CURSOR-ADDRESS         PIC XX.
    02  LINK-EXCHANGE-MESSAGE.
        03  CUSTOMER-NUMBER            PIC X(6).
        03  CUSTOMER-NAME              PIC X(30).
```

If you use an exchange message similar to the one shown above, you must supply the header data in a TST preceding the master link station in the exchange routing list. The exchange message received from the initiating station must not place data in the header area. The exchange message data sent by the initiating station must begin in the 13th character position.

After you define the link message header in the LINKAGE SECTION of a TST, you must perform the following operations in that TST's PROCEDURE DIVISION:

- Move the appropriate message type code (I,R,C or E) to the exchange message.
- Move the length of user data to be sent across the link into the exchange message.
- Move the attention identifier and cursor address into the header. In addition, any terminal format protocols must be specified in the data buffer.

NOTE

All filler fields in the exchange message header are reserved. Any data you place here is destroyed. These fields need not be initialized in the master transaction.

6.2.4 BASIC-PLUS-2 Master to Slave Message Format

In a BASIC-PLUS-2 TST, the following is an example of the format required in the MSGMAP statement immediately preceding the master link station in the exchange routing list.

```
600 \ MSGMAP   EM.MSG.TYPE.CODE$    =   1      &
      ,        EM.FILLER$           =   7      &
      ,        EM.LINK.MSG.LEN%                &
      ,        EM.ATTN.ID&          =   1      &
      ,        EM.CURSOR.ADDR$      =   2      &
      ,        EM.CUSTOMER.NUMBER   =   6      &
      ,        EM.CUSTOMER.NAME$    =  30
```

If you use an exchange message similar to the one shown above, you must supply the header data in a TST preceding the master link station in the exchange routing list. The exchange message

received from the initiating station must not place data in the header area. The exchange message data sent by the initiating station must begin in the 13th character position.

After you define the link message header in the MSGMAP statement, you must perform the following operations:

- Assign the appropriate message type code (I,R,C or A) to the type code field in the exchange message header.
- Calculate the length of user data to be sent across the link and assign this value to the message length field in the exchange message header.
- Assign the attention identifier and cursor address to the appropriate fields. If the IBM system requires terminal formatting protocols, these must be included in the message data beginning in the 13th character position.

The master link station takes a previously formatted exchange message, converts the data in the message from ASCII to EBCDIC and transmits the data to the IBM system. The data in the data buffer must conform with the structures required by the IBM system.

The flow of information through the TRAX/3271-TL interface is the following:

1. A master transaction is initiated. An exchange in this master transaction has a master link station in its routing list.
2. The TST preceding the master link station formats an exchange message in the format required for TRAX/3271-TL.
3. The exchange message is passed to the master link station which sends it to the slave transaction processor residing in an IBM 360 or 370 system running under CICS/0S or CICS/VS.
4. When the IBM transaction has completed, it sends a message back to the master link station.
5. The master link station places this response data into the exchange message and passes it to the next TST in the routing list for subsequent processing.

6.2.4.1 Handling Responses from IBM Transactions

There are no specific requirements on the formatting of the data portion of messages built by IBM transactions and returned to TRAX. The message as received by the line driver must begin with STX (start text), ESC (escape), command code, and write-control-character. This, however, is enforced by the CICS control program; the application program is only responsible for supplying the user data after the write control character (WCC).

The messages received by the TRAX master link station are in the following format:

```
COBOL Reply Format from IBM
01  EXCHANGE-MESSAGE.
    02  HEADER.
        03  IBM-COMMAND-CODE          PIC X.
        03  IBM-WRITE-CONTROL-CHAR    PIC X.
        03  FILLER                    PIC X (6).
    02  MESSAGE-DATA,
        03  ●●●
```

The command code values are:

"1" = WRITE.
"5" = ERASE/WRITE.
"?" = ERASE ALL UNPROTECTED

The command code, WCC, and data are all translated from EBCDIC to ASCII. Otherwise, they are not inspected or altered. If the command code is "?" (Erase All Unprotected), the WCC and user data are meaningless.

BASIC-PLUS-2 Reply Format from IBM

```
600 MSGMAP                                      &
    ,   EM.COMMAND.CODE$        =  1            &
    ,   EM.WRT.CTRL.CHAR$       =  1            &
    ,   EM.FILLERS$             =  6            &
```

The command code values are:

"1" = WRITE.
"5" = ERASE/WRITE.
"?" = ERASE ALL UNPROTECTED

The command code, WCC, and data are all translated from EBCDIC to ASCII. Otherwise, they are not inspected or altered. If the command code is "?" (Erase All Unprotected), the WCC and user data are meaningless.

CHAPTER 7
TST DEBUGGING AND TESTING FACILITIES

7.1 COMPILING TSTs

After you write a TST, use the DEC Editor to enter the source statements into a file.

The next step in the development process is compilation. The COBOL and BASIC-PLUS-2 compilers are equipped with a /TST switch. This switch must be specified in the command line when you are compiling TST source statements.

The /TST switch causes the compiler to create object files that conform to a TST's specialized structural requirements.

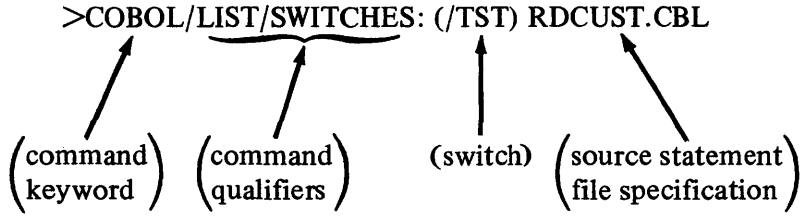
NOTE

Stand-alone programs designed to run in the TRAX support environment should not be compiled using the /TST switch. Procedures for support environment program development are outlined in the TRAX Support Environment User's Guide.

The following examples show how the /TST switch is specified at compile time.

7.1.1 Compiling a COBOL TST

If you specify the following DCL command line, the compiler processes the source file RDCUST.CBL. Since the /LIST command qualifier is specified, the listing file RDCUST.LST is also produced. If no fatal errors are detected by the COBOL compiler, it produces the object file RDCUST.OBJ.



7.1.2 Compiling a BASIC-PLUS-2 TST

In the following terminal sequence, typing the DCL BASIC command invokes the BASIC-PLUS-2 compiler. The BASIC environment comes up, and has its own set of commands to process the source file. The OLD command reads the source file RDCUST.B2S. The COMPILE command causes the compiler to process the statements in RDCUST.B2S. The /TST switch identifies the source file as a

TST. The /DEBUG switch indicates that you want the BASIC-PLUS-2 symbolic debugger included in the TST object module.

```
>BASIC
  Basic2 V01.5
  OLD RDCUST
  Basic2
  COM/TST/DEBUG
  Basic2
  EXIT
  >
```

In the example above, the TST compiled successfully. The object module RDCUST.OBJ was produced by the BASIC-PLUS-2 compiler, and included the BASIC symbolic debugger.

7.2 LINKING TSTs – THE TSTBLD UTILITY

Before a TST can run, the object module (created by the language compiler) must be linked to the system and language libraries, including the library of TRAX routines, and merged into an executable file called a task image. TRAX imposes specific structural requirements upon TST task-image files.

The TSTBLD utility simplifies the process of constructing a task-image file from object modules and insures that required linkages are performed to TRAX operating system resources. TSTBLD converts your compiler output object modules into acceptable TST task image files.

TSTBLD creates command files from arguments that you supply in an interactive dialog. It then invokes the TRAX Linker to process the input object modules into suitable TST task image files.

You invoke the TSTBLD utility from a support environment terminal by typing:

```
@[1,2]TSTBLD
```

The TSTBLD utility is invoked, and enters a dialog that begins with the question:

```
Object Module Names <EXIT>?
```

Using standard RMS file specifications, identify the object modules that comprise this TST. If more than one object module is required, separate the specifications with a plus sign (+).

```
filespec[+filespec. . . .+filespec]
```

The filename of the first module becomes the name of the task image and map files.

Table 7-1 shows the defaults that TSTBLD applies to the object module specifications.

Table 7-1
TSTBLD Specification Defaults

dev: [group,member] filename,typ;ver

Field	*Default*
dev:	Initially SY0:; however, each explicit device name becomes the default until the next time you explicitly specify the device in a file specification.
[group,member]	Initially the current UIC; however, each explicit UIC becomes the default UIC until the next time you specify a UIC in a file specification.
filename	None, you must always specify a filename.
.type	.OBJ
;ver	The highest version number

The next question asks you the name of the language compiler that created the object modules. COBOL is the assumed default. BASIC-PLUS-2 and MACRO can also be specified as responses.

Language <COBOL>?

If you specify a high-level language (COBOL and BASIC-PLUS-2), the next question asks if that language is installed with a shared object time system (OTS). If an OTS is present, answer, Y, YE, or YES, and the modules in the OTS are linked to the task image. Linking a TST to a shared OTS greatly reduces the size of the task image.

Is there a shared OTS for this language <YES>?

When you are debugging a TST, different modules must be included in the task image. TSTBLD has a loop of three questions to specify the kind of debugging you want to perform the task image. This loop begins by asking:

Debug mode <NO>?

If you press the RETURN key, the dialog for this TST ends, and you are returned to the question "Object module names <EXIT>?". If you answer Y, YE, or YES, then debugging support is included in the task image and TSTBLD continues by asking:

Transaction processor debug <NO>?

If you want a support environment terminal assigned to a TST for debugging purposes in the transaction processing environment, answer Y, YE or YES to this question. TSTBLD then asks you to specify:

Debugging terminal ?

You respond with the device name of the support environment terminal you want assigned to the TST. When a TST is built with a debugging terminal, you can use language debugging facilities such as the BASIC-PLUS-2 debugger and the COBOL ACCEPT and DISPLAY statements to examine and modify values in the TST.

After you answer this question, the dialog returns to the question "Object module names <EXIT>?".

If you want to debug the TST in the support environment using the DEBUG utility, answer NO to the transaction processing debug question by pressing the RETURN key. TSTBLD then asks for the:

Initializing module name <NONE>?

You specify the name of an object module used to initialize files and common data structures before execution of the TST. This initializing module must have an entry point name of TSTINI. In all other respects, the initializing module must conform to the structural requirements of a TST. The debugger calls and executes this module prior to executing the TST. When attempting to debug TSTs that depend upon prior execution in a transaction or exchange, the initializing module is useful for positioning files. After you answer this question the utility returns to the question: "Object Module Names <EXIT>?"

NOTE

If you include an initializing module in a BASIC-PLUS-2 TST, the Linker returns a multiply defined symbol error for module "OTSVA$". This error is informational and does not affect execution.

At this point, you can build another TST. If you press the RETURN key, accepting the default "EXIT", the utility terminates. The command file used to invoke TSTBLD executes a LINK command to build the TST task image. The TST task image and map files are built in your current UFD with the filename of the first object module specified to the TSTBLD dialog.

In certain cases, you may want to retain the command file created by TSTBLD. In this case, do not invoke TSTBLD using the command file [1,2]TSTBLD. Instead you should:

```
>RUN $TSTBLD
```

After the utility dialog ends and you exit from TSTBLD through the question "Object module names <EXIT>?" TSTBLD creates a command file in your current UFD. This command file is named TSTCOM.CMD. You can rename this command file or copy it into another directory. When you want to build a TST task image, simply type the command string:

```
>LINK @filename
```

NOTE

You should not attempt to edit the command file created by TSTBLD. Doing so will cause unpredictable and/or fatal results.

7.2.1 Examples of TSTBLD Usage

Figure 7-1A shows the TST specification sheet describing the TSTBLD parameters to construct a task image of the COBOL TST RDCUST to be used with the support environment DEBUG utility.

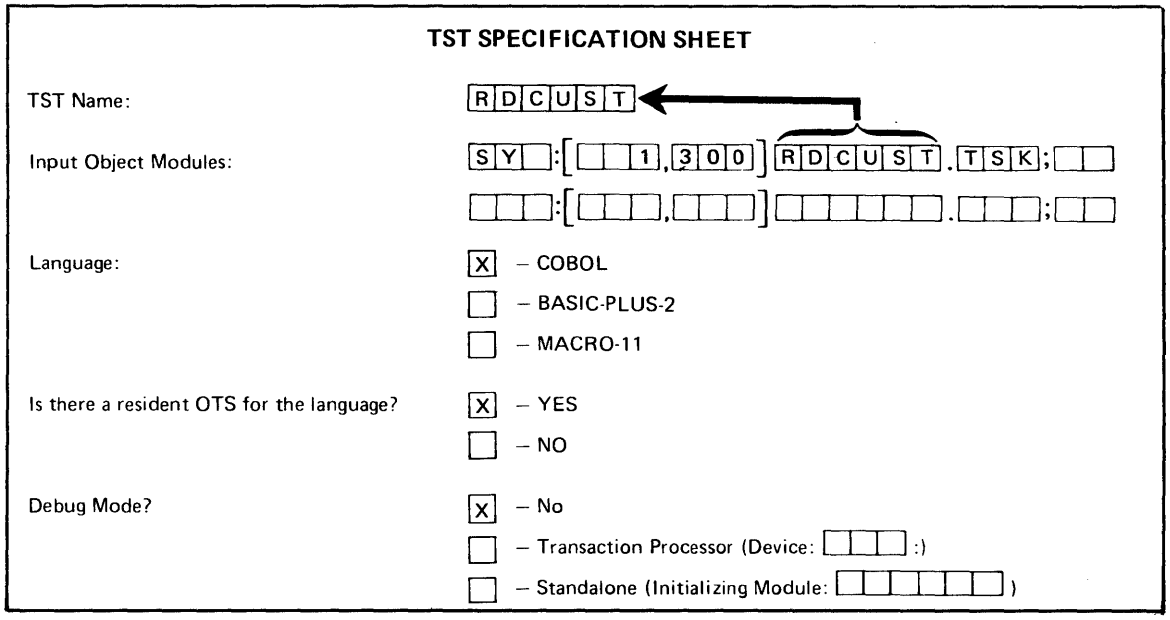


Figure 7-1a TST Specification Sheet for COBOL TST RDCUST.

Figure 7-1B shows the terminal dialog corresponding to the specifications of Figure 7-1A:

```
>
>@[1,2]TSTBLD (RET)
>RUN $TSTBLD

TSTBLD V1.0

Links TSTs for testing or final use.

Object module names <EXIT>? RDCUST (RET)

Language <COBOL>?(RET)

Is there a shared OTS for this language <YES>?(RET)

Debug mode <NO>? YES (RET)

Transaction Processing Debug <NO>?(RET)

Initializing module name <NONE>?(RET)

Object module names <EXIT>?(RET)
>LIN @TSTCOM
>
>DELETE TSTCOM.CMD;*
>@ <EOF>
>
```

Figure 7-1b TSTBLD Dialog to Build "RDCUST" For Debugging

Figure 7-2A shows the TST specification describing the TSTBLD parameters to construct a task image of the BASIC-PLUS-2 TST RDCUST that can be used in transaction processor debugging from a support environment terminal TT2:

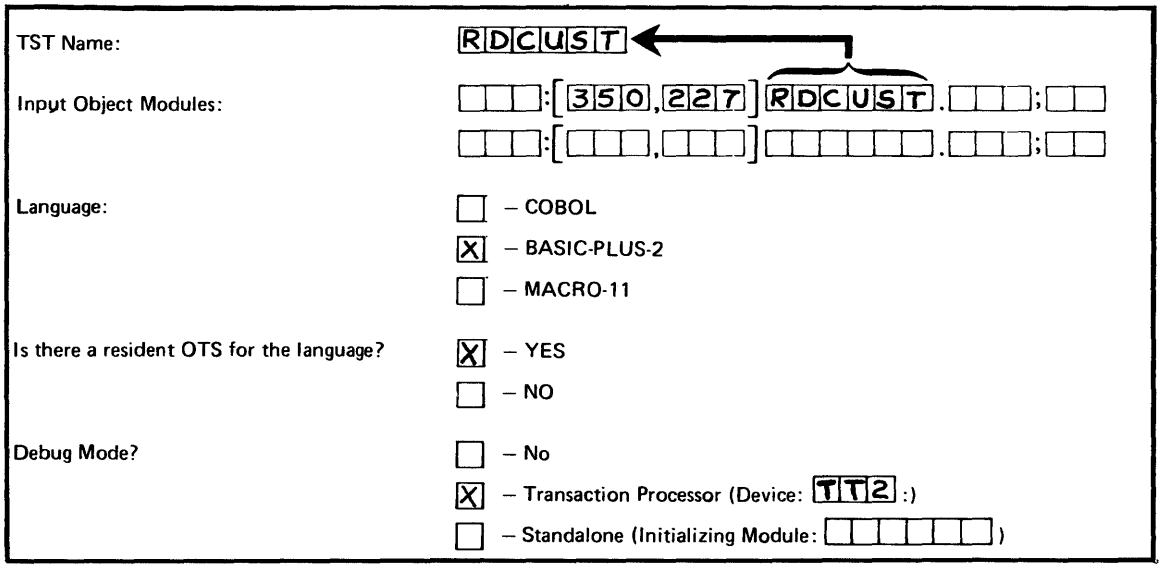


Figure 7-2a TST Specification Sheet for BASIC TST RDCUST.

Figure 7-2B shows the terminal dialog corresponding to the specification in Figure 7-2A.

```
>@[1,2]TSTBLD (RET)
>RUN $TSTBLD

TSTBLD V1.0

Links TSTs for testing or final use.


Object module names <EXIT>? [350,227]RDCUST (RET)

Language <COBOL>? BASIC (RET)

Is there a shared OTS for this language <YES>? (RET)

Debug mode <NO>? YES (RET)

Transaction Processing Debug <NO>? YES (RET)

Debugging terminal? TT2: (RET)

Object module names <EXIT>? (RET)
>LIN @TSTCOM
>
>DELETE TSTCOM.CMD;*
>@ <EOF>
```

Figure 7-2b TSTBLD Dialog to Build "RDCUST" for Debugging

7.3 TST DEBUGGING IN THE SUPPORT ENVIRONMENT

After you have coded a TST, created a source language statement file, compiled the source statements, corrected the syntax errors, and used the TSTBLD utility to create an executable task image, the next step in the TST development process is debugging.

Debugging TSTs begins in the support environment. A TRAX utility program, DEBUG, allows you to debug TSTs in an interactive manner from a support environment terminal. You may also include system debugging aids such as the BASIC-PLUS-2 debugger.

7.3.1 DEBUG–The TST Debugging Utility

This utility assists you in debugging a TST by simulating its operating environment.

The DEBUG utility simulates a transaction processor, allowing your TST to execute and access system resources in the same way as a TST installed in a transaction processor. The TST operating under DEBUG can access:

- The TRAX system library routines
- A simulated exchange message
- A simulated transaction workspace
- Application data files

The simulated exchange message and workspace data are provided by the programmer.

All calls to the system library and I/O routines are logged in the logging file (or device) together with the parameters specified in the call. When the TST exits, the contents of the simulated workspace are written to the logging device.

7.3.2 Using the DEBUG Utility

You invoke the DEBUG utility from a support environment terminal by typing:

```
RUN $DEBUG
```

The program issues an identifying message, and begins an interactive dialog starting with the question:

```
TST filename?
```

Specify a TST task image file. DEBUG assumes the current system device and your default UFD, the highest version number, and the filetype .TSK as default values. In every case, you must specify a TST filename. The TST task image you specify must be created by the TSTBLD utility, with support for debug mode.

The DEBUG utility executes this task image file.

Logical file name <DONE>?

Specify the logical filename used in the TST. If you answer this question with a 1- to 6-character logical filename, DEBUG responds with the prompt:

File specification "?

You respond with the RMS file specification for the logical file you named in the preceding question. The dialog continues to prompt for logical file name until you press the RETURN key indicating you are done.

You may press RETURN in response to the "Logical file name" question, without specifying any logical filenames. In this case, DEBUG prompts you during execution for the RMS file specification of any logical files opened in the TST.

The logical filename and file specification are used to access physical files when the TST being debugged attempts to access the logical file.

Logging device<TI:>?

The dialog continues by asking you to specify the file and/or device where the system and I/O calls are logged by the DEBUG utility. The default is your terminal (TI0:). If you respond with a different file or device specification, DEBUG writes output to that file or device.

Message file?

The message file is the simulated exchange message. It can be a file you create using the DEC Editor, or it can be a terminal device specification such as TI:. If exchange message data is brief such as a record key, then specifying the terminal device as the exchange message source is appropriate. If the exchange message data runs to any length, then a file may be more appropriate. DEBUG assumes a filetype of .TXT for any exchange message data files.

Since exchange messages often run longer than the 80-character width of a video terminal, DEBUG recognizes the hyphen (-) followed by a carriage return as a continuation character in exchange message and transaction workspace data. The following example illustrates the way you can use the DEC Editor to create a set of four exchange messages, each consisting of 36-characters.

```
>EDIT EMRDCUST.TXT (RET)
*I (RET)
000001(RET)
000012(RET)
000100(RET)
000000(RET)
(CTRL/Z)
*EX(RET)
>
```

```
Workspace file<TI:>?
```

The workspace file is the simulated transaction workspace. It can be a file you create using the DEC Editor, or it can be a terminal device specification such as TI:. If the workspace data is brief such as a record key, then specifying the terminal device as the source of workspace data is appropriate. If the workspace is a large area such as a record data structure, then creating a file is more appropriate. DEBUG assumes a filetype of .TXT for any transaction workspace data files.

If the TST you are debugging does not have a defined workspace, then you must respond to this question with the RETURN key.

Since the transaction workspace often exceeds the 80-character width of a video terminal, DEBUG recognizes the hyphen (-) followed by a carriage return as a continuation character in exchange message and transaction workspace data. The following example illustrates the way you can use the DEC Editor to create a set of four empty transaction workspaces, to initialize this structure for subsequent use by a TST. Each empty workspace is a 205-character area, initialized with zeros.

```
>EDIT EMPTY.TXT
*I (RET)
ØØØØØØØØØØØØØØØØØØØØØØØØØØØØØØØØØØØØØØØØØØØØØØØØØØØØØØØØØØØØ-(RET)
ØØØØØØØØØØØØØØØØØØØØØØØØØØØØØØØØØØØØØØØØØØØØØØØØØØØØØØØØØØØØ-(RET)
ØØØØØØØØØØØØØØØØØØØØØØØØØØØØØØØØØØØØØØØØØØØØØØØØØØØØØØØØØØØØ-(RET)
ØØØØØØØØØØØØØØØØØØØØØØØØØ(RET)
(CTRL/Z)
*COPY 10:40 %TO %END(RET)
*COPY 10:40 %TO %END(RET)
*COPY 10:40 %TO %END(RET)
*EX(RET)
>
```

After you specify the workspace data, DEBUG loads the specified TST task image and begins executing.

If you have specified TI: in response to the "Message file?" or "Workspace file?" questions, DEBUG prompts you for this data before it begins to execute the TST task image.

```
Input Message Data?
Input Workspace Data?
```

If you end a line with a hyphen (-), DEBUG reissues the same prompt, and continues to accept data from the terminal. DEBUG stops prompting when it detects the RETURN character without a preceding hyphen.

As the TST starts to open application data files the DEBUG utility intercepts the file access calls. In the case where you did not supply a logical file name or physical file specification in the dialog that initiated DEBUG, the following prompt is issued when an open call is issued for a logical file name:

File specification "XXXXXX"?

In this case, the DEBUG utility found a logical filename "XXXXXX" that requires a corresponding physical file specification. You must specify at least the filename and filetype of this application data file. The defaults assumed by DEBUG are the system device, your default UFD, and the current version. On subsequent accesses to the same logical filename the same physical file is read.

The TST then executes to completion and the debugging session (using the available language debugging tools) takes place. When the TST exits, the message file is read again, and if another message is found, then DEBUG initializes the message, TST workspace, and the logical file assignments before beginning another execution of the TST.

After the last message has been processed, the debugging session ends.

When the TSTs are debugged satisfactorily, they can be integrated into a 'system test' environment to complete the application debugging. This technique is discussed in Chapter 14.

Figure 7-3a shows the debugging output from the COBOL version of "RDCUST."

```
>RUN $DEBUG (RET)
DEBUG   01-01

TST Debug Utility

TST filename?RDCUST (RET)

Logical file name<DONE>?(RET)

Logging device<TI:>?(RET)

Message file?TI: (RET)

Workspace file?EMPTY (RET)


Input Message Data ? 000004 (RET)
File specification "CUSTOM"?[111,7]CUSTOM.DAT (RET)

  ***          Subroutine Name  $OPEN     -     TST File Name  RDCUST

               File Name  CUSTOM

                    Completion Status(STS)  1  -  Status Value(STV)  0

  ***          Subroutine Name  $CONNECT     -     TST File Name  RDCUST

               File Name  CUSTOM

                    Completion Status(STS)  1  -  Status Value(STV)  0

  ***          Subroutine Name  $FIND     -     TST File Name  RDCUST

               File Name  CUSTOM

                    Completion Status(STS)  1  -  Status Value(STV)  0
               Specified Key
                    000004

  ***          Subroutine Name  $GET     -     TST File Name  RDCUST

               File Name  CUSTOM

                    Completion Status(STS)  1  -  Status Value(STV)  0
               Specified Key
                    000004
               Specified Record
                    000004Mesa Stamp Center             1 R Street
     Mesa, AR
                                                  852016026991347N K Nentakian
 0000000500000000000000000000000000000000
                    30001

  ***          Subroutine Name  PRCEED     -     TST File Name  RDCUST

               Message Data
                              000004Mesa Stamp Center             1 R Street
                Mesa, AR
                                                  852016026991347N K Nenta
kian             500.00
```

Figure 7-3a DEBUG Utility Output for COBOL "RDCUST" TST

```
***     Workspace Data
              000004Mesa Stamp Center              1 R Street
 Mesa, AR
                                                852016026991347N K Nentakian       0000
00050000000000000000000000000000000000
              30001

  ***     TST Exit


Input Message Data ? 000100 (RET)

  ***           Subroutine Name  $OPEN      -      TST File Name  RDCUST

                File Name  CUSTOM

                     Completion Status(STS)  1  -  Status Value(STV)  0

  ***           Subroutine Name  $CONNECT      -      TST File Name  RDCUST

                File Name  CUSTOM

                     Completion Status(STS)  1  -  Status Value(STV)  0

  ***           Subroutine Name  $FIND      -      TST File Name  RDCUST

                File Name  CUSTOM

                     Completion Status(STS)  -1472  -  Status Value(STV)  0
                Specified Key
                     000100

  ***           Subroutine Name  REPLY      -      TST File Name  RDCUST

                               Reply Number  2

                Message Data
                               No Record Exists under that Key
                               File Status Word: 23


***     Workspace Data
              00000000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000
              00000000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000
              00000

  ***     TST Exit


Input Message Data ?  (SP)  (SP)  (SP)  (RET)

  ***           Subroutine Name  $OPEN      -      TST File Name  RDCUST

                File Name  CUSTOM

                     Completion Status(STS)  1  -  Status Value(STV)  0

  ***           Subroutine Name  $CONNECT      -      TST File Name  RDCUST

                File Name  CUSTOM

                     Completion Status(STS)  1  -  Status Value(STV)  0

  ***           Subroutine Name  REPLY      -      TST File Name  RDCUST
```

Figure 7-3a (cont.) DEBUG Utility Output for COBOL "RDCUST" TST

```
                                Reply Number  2

                Message Data
                                You Specified an Invalid Customer ID #


***     Workspace Data
                000000000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000
                000000000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000
                00000

  ***     TST Exit

Input Message Data ? ^Z (CTRL/Z)

>
```

Figure 7-3a (cont.) DEBUG Utility Output for COBOL "RDCUST" TST

CHAPTER 8
USING THE DEFINITION UTILITIES

The TRAX system includes a set of utilities for defining the processing paths and components of a transaction processor. These programs are called transaction processor definition utilities. You run the definition utilities from an interactive terminal connected to the TRAX support environment. Each utility leads you through a dialog, asking a set of questions. You answer these questions using information supplied by the application designer. Chapters 9 through 13 describe each of the definition utilities.

8.1 UTILITY DESCRIPTIONS

The information required to install a transaction processor resides in a set of definition data files. Each definition utility modifies one specific definition file. Table 8-1 provides a brief functional description of each definition utility.

Table 8-1
Definition Utility Functions

Chapter	Utility	Function
9	TPDEF	Sets dimensions for transaction processor components. Defines the processor's common data areas. Creates the definition data files and transaction processor record.
10	STADEF	Defines stations referenced by the transaction processor.
11	TRADEF	Defines the transactions that can be executed by the transaction processor. Specifies the exchanges, exchange routing lists, and subsequent actions taken by the system.
12	FILDEF	Specifies application data files accessed by the transaction processor.
13	WORDEF	Defines work classes and associates transaction names with each work class.
13	AUTDEF	Assigns identifiers, passwords, and work classes to individual users.

8.2 TRAX UTILITY DIALOG CONVENTIONS

TRAX Utility dialogs follow these conventions:

Help Text

If a question does not give enough information for you to answer, type a question mark followed by the RETURN key (? RET). The utility responds with an explanation and repeats the question.

```
Transaction processor name? ? RET

Enter the name of an existing transaction processor, 1- to 6-character
alphanumeric string.

Transaction processor name?
```

RETURN Key

Pressing the RETURN key (RET) terminates your response.

```
Station type? TERMINAL RET
Station name?
```

Responses

Each question indicates the type of response expected by the utility. The utility checks the input as you enter it. If the input is incorrect, the utility returns an error message and repeats the question.

Defaults

If the system assumes a default value as a response to a question, that value is shown in angle brackets following the question.

```
Station priority <128>
```

You can accept the default value by pressing the RETURN key. For example,

```
Station priority <128> RET
```

To specify a different value, type the value followed by a carriage return.

```
Station priority <128> 124 RET
```

YES/NO

If a question requires a YES or NO answer, you can respond by typing Y, YE, YES, N, or NO followed by the RETURN key.

```
Stage updates <YES>? N RET
Repeat <NO>? YE RET
```

Numeric Data

Any question that requires a numeric reply expects decimal input, unless the question indicates otherwise.

```
Exchange time limit [minutes]? 5 RET
```

[text]

Some questions include text within square brackets to clarify the entity or units to be specified.

```
Maximum size of exchange message [bytes]?
```

Abbreviation

If a question asks you to specify an item from a known set (utility commands, for example), you need to type only the characters required to uniquely identify the selected item within the set.

```
Command? P (RET)
```

P stands for PRINT, one command from a set that includes ADD, DELETE, EDIT, EXIT, INDEX, PRINT, and SHOW.

ESCAPE Key

Utilities ask questions in a specific order. To reverse the order, that is, to return to the last question asked, press the ESCAPE key (ESC). In a loop of questions, pressing the ESCAPE key returns you to the first question in that loop.

```
Exchange label? ACNT (RET)
Form name? (ESC)
Exchange label?
```

The following 3 questions deal with the subsequent action of the exchange.

```
Wait <YES>? (ESC)
Destinations command <DONE>?
```

EXIT Command

If you type the keyword EXIT in response to the "Command?" question in any utility, the utility exits normally.

CTRL/Z to Exit

You may make an orderly exit from a TRAX utility by typing (CTRL/Z) in response to a question.

CHAPTER 9

TRANSACTION PROCESSOR DEFINITION: THE TPDEF UTILITY

The TPDEF utility creates the definition data files and transaction processor common areas that define the parameters of a transaction processor.

When you run TPDEF to create or modify an existing transaction processor, the following files and data structures are created or changed.

You must run TPDEF to create a new transaction processor. The other TRAX definition utilities cannot be run to define components of that transaction processor until after a transaction processor is defined.

- The transaction processor's common data area.
- A terminal management task image file.
- Six definition data files in the UFD [1,300], which are subsequently processed by a set of definition utilities.
- An entry for the named transaction processor in the transaction processor definition file [1,1] TPDEF.TPF.

9.1 TRANSACTION PROCESSOR DATA STRUCTURES

When you create a transaction processor using TPDEF, it enters a dialog to gather the transaction processor parameters. Once the dialog finishes, TPDEF generates and executes a set of DCL commands to create the transaction processor. The system displays each command at your terminal as the command executes.

When you run TPDEF using a command that modifies or recreates definition structures, the DCL commands that make the changes are also displayed and executed at your terminal.

9.1.1 The Transaction Processor File Record

Each defined transaction processor has a record in the system file [1,1] TPDEF.TPF. TPDEF creates this record when you run the utility to define a new transaction processor. The record contains the name of the processor, a unique identifying label, and other data needed by the system for monitor and control purposes. TPDEF sets up the record fields. Some fields are filled later by the system each time you run a definition utility or install the transaction processor.

9.1.2 The Terminal Management Task and Common Data Area

The terminal management task contains data the contributes to the processor's handling of application terminals. The common data area contains data that relates to every aspect of the processor.

More specifically, it reflects your responses to TPDEF questions, that cover the following general topics:

- Transaction types and instances
- Transaction Step Tasks (TSTs)
- Network communications
- Communications with batch processing
- Stations
- Application data files
- Transaction slots
- Crash recovery

When you install a transaction processor, the system loads all required tables and tasks into memory. These tables and task image areas remain memory resident until you remove the transaction processor.

9.1.3 The Definition Data Files

The TPDEF utility creates six definition data files that are subsequently populated by six corresponding utilities. Table 9-1 gives the names of each file, a brief description of its contents, and the utility provided to process it.

Table 9-1
The Definition Data Files

File	Contents	Utility
tpname.STA	Records that define each statement within the transaction processor.	STADEF
tpname.TRA	Records that define each transaction type used by the processor.	TRADEF
tpname.FIL	Records that describe each application data file.	FILDEF
tpname.WOR	Records that describe each work class (a specific set of transaction types).	WORDEF
tpname.AUT	Records that determine the work class or classes that individual users can transact.	AUTDEF
tpname.FDF	Records that contain compiled form definitions.	ATL

(The ATL Utility is described in the TRAX ATL Language Reference Manual.)

The system information files are described fully in the chapters that discuss the corresponding utilities.

9.2 THE TPDEF UTILITY

The following sections describe the questions posed by the TPDEF utility dialog. The order that the questions are asked by the utility depends on the command you specify and the answers you supply to various questions. Several examples of TPDEF usage are shown in this section.

You should read the following paragraphs to familiarize yourself with the information required by the TPDEF utility.

The responses you make to TPDEF dialog questions should reflect the maximum configuration required by the final system. For example, if the transaction processor will eventually use 15 TSTs to execute the defined transactions, but only 7 are currently operational, you should specify 15 when responding to the question:

Maximum number of TSTs?

Anticipating the configuration and requirements of the final system avoids editing the transaction processor definition every time a new component is added to the processor. Each time you run TPDEF, a number of components are recompiled and relinked. Planning for expansion during the system design and definition process can save considerable programmer time during the implementation phase of an application.

9.2.1 Invoking the TPDEF Utility

You invoke the TPDEF utility by logging on to a support environment terminal and typing:

@[1,2]TPDEF

The command file you invoked issues the command:

RUN $TPDEF

The TPDEF utility responds by issuing the identifying text:

TPDEF V1.0

Transaction processor definition utility

TPDEF then begins by asking the question:

Command <EXIT>?

You answer this question by specifying one of the eight command keywords. Typing a carriage return in response to this question causes the TPDEF utility to exit. Any generated commands are executed immediately following the utility exit.

The command keywords you may specify in response to the TPDEF "Command" question are listed below, with a brief description of their function.

- CREATE – Creates a new transaction processor definition including the common data area, the terminal management task, the six definition data files, and the processor record in TPDEF.TPF.
- COPY – Creates an identical copy of an existing processor and gives the copy a new name. The new processor includes renamed duplicates of the related processor definition files.
- DELETE – Deletes an existing processor's related processor definition files, and the transaction processor record on [1,1]TPDEF.TPF.
- EDIT – TPDEF modifies the transaction processor definition record, and issues commands to construct the related processor tables and tasks. EDIT mode does not affect the definition data files.
- INDEX – Displays on your terminal the names of all defined transaction processors.
- PRINT – Lists the parameters of the named transaction processor on the line printer.
- RENAME – Changes the name of an existing transaction processor and its related processor definition files.
- SHOW – Displays at your terminal (TI:), the parameters of the named transaction processor.

The TPDEF dialog varies according to the command keyword you specify. The TPDEF command keywords fall into five different functional groupings:

1. The CREATE and EDIT Commands are discussed in Section 9.2.2.
2. The INDEX Command is discussed in Section 9.2.3.
3. The PRINT and SHOW Commands are discussed in Section 9.2.4.
4. The COPY and RENAME Commands are discussed in Section 9.2.5.
5. The DELETE Command is discussed in Section 9.2.6.

9.2.2 Creating or Editing a Transaction Processor Definition

The CREATE Command
The EDIT Command

You must create a transaction processor before you use any of the other TPDEF commands to modify that transaction processor. After you answer the "Command ?" question with the keyword CREATE, TPDEF enters a dialog to gather the parameters needed to create a transaction processor.

The EDIT command lets you modify the definition of an existing transaction processor. The EDIT command asks you to review the existing transaction processor definition, changing parameter values as required.

The CREATE and EDIT commands use the same dialog.

After you respond to each question, the utility validates your response and automatically issues the next question.

The dialog begins by asking for:

New transaction processor name?

TPDEF asks you to name the processor. The name you specify must be a 1- to 6-character alphanumeric string.

If you are editing a transaction processor definition, TPDEF uses the name you specify to retrieve the corresponding transaction processor definition record from the file.

If you are creating a transaction processor definition, this name also becomes the file name for the definition datat files created by TPDEF. For example, if you name a processor SAMPLE, TPDEF creates the following six files in UFD [1,300] :

SAMPLE.STA	(the station definition file)
SAMPLE.TRA	(the transaction definition file)
SAMPLE.FIL	(the application data file definition file)
SAMPLE.WOR	(the work class definition file)
SAMPLE.AUT	(the user authorization file)
SAMPLE.FDF	(the forms definition file)

Once the transaction processor definition record is created (retrieved), the dialog continues with the question:

Maximum number of transaction types?

You must specify the maximum number of transactions that can be defined. Each transaction type has a corresponding transaction definition record in the file tpname.TRA. A transaction processor can support up to 64 transaction types. The dialog continues by asking for the:

Maximum number of concurrent transaction instances?

A transaction instance is an executing transaction. The number you specify limits the number of transaction instances that the processor can execute at one time. The number must be less than or equal to 64. If a situation occurs where more than this number of transaction instances are attempting to execute concurrently, the requests to invoke the additional transaction are queued and are executed when currently executing transaction instances conclude and free system resources.

The next parameter you are asked to supply is the:

Maximum number of application terminals?

TPDEF needs to know the maximum number of deviced defined as application terminals. A transaction processor can support up to 64 application terminals.

The next question is:

Maximum number of user TSTs?

TPDEF asks you to specify the maximum number of transaction step tasks (TSTs) to be included in the transaction processor. Every TST within a processor has a unique station. Defining this unique station makes the associated TST part of the transaction processor. A transaction processor definition can specify up to 256 TST stations.

The TRAX system allows transaction processors to communicate among themselves, and with transaction processors in IBM systems. The communicating transaction processors can run on the same computer or they can run in separate computers. You must define special link stations that send and receive data between transaction processors to support this transaction processor link facility.

There are two kinds of link stations, master and slave.

A master link station receives exchange messages from local transaction instances and forwards the messages to a specific remote transaction processor or IBM system. The TPDEF dialog question to define master link station is:

Maximum number of master link stations?

This value is the total number of other transaction processors that the transaction processor you are defining expects to communicate with. A transaction processor definition can specify up to 10 master link stations. A master station in one active transaction processor communicates with a slave link station in another active transaction processor in the same or a remote system. See the description of the TRAX/TL facility in Chapter 6 for more information about link stations.

After you have defined master link stations, TPDEF asks you to specify:

Maximum number of slave link stations?

A slave link station receives exchange messages from a master link station. The master link station initiating the slave transaction resides in a remote transaction processor. Each message received causes an exchange to be initiated, with the slave link as the source station. The number of slave link stations you specify corresponds to the total number of concurrent transactions that can be initiated by the other transaction processors linked to the transaction processor you are defining.

You can define up to 64 slave stations in a single transaction processor. If you specify a receive link message, the TPDEF dialog asks for the:

Maximum size of a receive link message?

A receive link message is the exchange message received by a slave link station in the local transaction processor. In order to manage its resources, the transaction processor needs to know the maximum size in bytes of any such message forwarded by a remote master link station. The largest message size you can specify is 512 bytes.

After defining the link stations, you are asked to specify the number of interfaces to the batch processor. These include submit and slave batch stations. The first question asks for the:

Number of submit batch stations?

For a transaction instance to submit a batch command file, a submit batch station must be defined in both the transaction processor and station definitions. A transaction processor can have only one submit batch station.

The second question asks for the:

Number of slave batch stations?

If your transaction processor permits batch jobs to initiate transactions, you must define one or more batch slave stations. The number of batch-initiated transaction instances that can run concurrently is equal to the number of defined slave batch stations. A transaction processor can support up to 16 batch slave stations.

The dialog next asks you to specify:

Maximum number of mailbox stations?

A mailbox station is used to store messages that can be retrieved by other transaction instances. A set of system library routines have been supplied to allow TSTs to store and retrieve data at mailbox stations.

A processor can support up to 10 mailbox stations.

The TPDEF dialog next asks you to specify:

Maximum number of application data files?

TPDEF asks you to specify the maximum number of files that can be defined in the definition data file tpname.FIL. These application files include files that contain the application-specific data related to transaction processing, as well as work files created, opened, closed, or erased dynamically by TSTs. See Chapter 12 for a description of the file tpname.FIL and FILDEF, its corresponding utility. The number of application data files must be less than or equal to 64.

The next question asks you to define the:

Maximum size of transaction slot [64 byte blocks]?

For each transaction instance, the transaction processor allocates a portion of memory called a transaction slot. Each slot consists of three parts:

1. An area that contains the current exchange message
2. A transaction workspace used in turn by every TST that executes on behalf of the transaction instance
3. A system workspace used by the transaction processor to store data for staged file updates and exchange recovery during the life of the transaction instance.

The size of the transaction slot areas varies from one transaction type to another. The TPDEF utility asks you to specify the maximum number of 64-byte blocks required for a transaction slot by any transaction defined for the current transaction processor.

When you are running TRADEF to define a transaction type, you can calculate the maximum slot size using the individual space requirements of the exchange message, transaction workspace, and system workspace sizes.

The formula for calculating the system workspace is explained in Figure 11-1 as part of the TRADEF description.

The TPDEF dialog concludes with the question:

Automatic crash recovery?

When a transaction processor includes this option, it can recover from a software crash and continue execution, aborting in progress transactions and completing any unstaging operations in progress.

After you answer this question, the TPDEF utility exits and automatically spawns several DCL command lines. The operating system processes this set of commands and creates the definition data files and the terminal task and the transaction processor common data areas.

For illustration purposes, a transaction processor called"SAMPLE" has been designed, and is used throughout the discussions in the manual. Figure 9-1a shows a transaction processor specification sheet prepared for the transaction processor "SAMPLE".

Figure 9-1b shows the terminal listing of the dialog used to CREATE a transaction processor definition for "SAMPLE". Following the terminal dialog, the system issues a number of DCL commands to create the definition data files required to build a transaction processor.

9.2.3 Listing the INDEX of Defined Transaction Processors

The INDEX Command

The INDEX command lists all defined transaction processors. The action is sutomatic once you specify INDEX in response to the TPDEF in response to the TPDEF "Command ?" question.

The 3-character transaction processor identifier shown in the INDEX listing is the suffix of the various transaction processor tasks that are active when a transaction processor is installed. For example, if the transaction processor "SAMPLE" has an ID of "AAG", the terminal task for "SAMPLE" is known to the system as the task TIMAAG.

Once the INDEX is printed on your terminal, the TPDEF utility returns with the "Command?" question, allowing you to invoke some other command or to exit from the utility. Figure 9-2 shows the TPDEF terminal dialog listing for the INDEX command.

TRANSACTION PROCESSOR SPECIFICATION SHEET

Transaction Processor Definition: SAMPLE

Transaction Processor Name: SAMPLE

Maximum number of transaction types:	(0–64)	4
Maximum number of concurrent transaction instances:	(0–64)	4
Maximum number of application terminals:	(0–64)	4
Maximum number of user TSTs	(0–256)	10
Maximum number of master link stations:	(0–10)	Ø
Maximum number of slave link stations:	(0–64)	Ø
Maximum size of receive link message:	(0–512)	
Maximum number of submit batch stations:	(0–1)	Ø
Maximum number of slave batch stations:	(0–16)	Ø
Maximum number of mailbox stations:	(0–10)	Ø
Maximum number of application data files:	(0–64)	1
Maximum transaction slot size:	(1–1022)	8 blocks
Automatic crash recovery:	☐ – YES	☒ – NO

Figure 9-1a Transaction Processor Specification Sheet for "SAMPLE"

```
>@[1,2]TPDEF (RET)
>RUN $TPDEF

TPDEF V1.0
Transaction processor definition utility

Command <EXIT>? CREATE (RET)

New transaction processor name? SAMPLE (RET)

Maximum number of transaction types? 4 (RET)

Maximum number of concurrent transaction instances? ? (RET)

Specify the maximum number of transaction instances (<=64) that can be active at
one time.

Maximum number of concurrent transaction instances? 4 (RET)

Maximum number of application terminals? 4 (RET)

Maximum number of user TSTs? 10 (RET)

Maximum number of master link stations? 0 (RET)

Maximum number of slave link stations? 0 (RET)

Maximum number of submit batch stations? 0 (RET)

Maximum number of slave batch stations? 0 (RET)

Maximum number of mailbox stations? 0 (RET)

Maximum number of application data files? 1 (RET)

Maximum size of a transaction slot [64 byte blocks]? 8 (RET)

Automatic crash recovery <NO>? (RET)



>
>RUN $FDFBLD
>RENAME [1,300]NONE.FDF [1,300]SAMPLE.FDF
>
>MACRO/OBJECT:TEMP [1,1]RMSMAC/LI+[1,300]SAMPLE+[24,10]TPSCOM
>MACRO/OBJECT:TEMP.CTX [1,300]SAMPLE+[24,10]TIMCTX
>LINK @TEMP.LNK
>SET PROTECTION [1,300]SAMPLE.* (SYSTEM:RWED,OWN:RWED,GROUP:RWED,WORLD:RWED)
>DELETE TEMP.*;*
>@ <EOF>
>
```

Figure 9-1b Terminal Listing of CREATE Command Dialog

```
>RUN $TPDEF (RET)

TPDEF V1.0
Transaction processor definition utility

Command <EXIT>? INDEX (RET)



NAME                ID

ABCDEF              AAA
1                   AAB
BATCH1              AAC
APP001              AAD
TEST1               AAE
FNT002              AAF
BACKUP              AAG
CREDIT              AAH
APGEXA              AAI



Command <EXIT>? (RET)
```

Figure 9-2 Terminal Listing of INDEX Command Dialog

9.2.4 Printing or Showing a Transaction Processor Definition

The PRINT Command
The SHOW Command

The PRINT command prints a hard copy listing of a transaction processor definition on the system device assigned to the logical device (CL0:). This device is generally the line printer. Consult your system manager for information regarding which device he has assigned to CL0:.

The SHOW command lists the transaction processor definition on your terminal. This command is useful for checking your work after creating or editing a transaction processor definition.

After invoking the PRINT or SHOW command, TPDEF asks you to supply the name of the existing transaction processor whose definition is to be printed. The question is:

Old transaction processor name?

Once you enter the transaction name, TPDEF reissues the "Command?" question after the definition is spooled to the printer. You can invoke another command, or enter a carriage return to exit from TPDEF.

Figure 9-3 shows the terminal dialog listing as an example of the TPDEF utility SHOW command. The PRINT command dialog is identical, except the listing prints on logical device (CL0).

```
>
>RUN $TPDEF (RET)

TPDEF V1.0
Transaction processor definition utility

Command <EXIT>? SHOW (RET)

Old transaction processor name? SAMPLE (RET)



Definition of Transaction Processor - SAMPLE


Maximum number of transaction types                              4
Maximum number of concurrent transaction instances               4
Maximum number of application terminals                          4
Maximum number of user TSTs                                      10
Maximum number of master link stations                           0
Maximum number of slave link stations                            0
Maximum size of receive link message                             0
Maximum number of submit batch stations                          0
Maximum number of slave batch stations                           0
Maximum number of mailbox stations                               0
Maximum number of application data files                         1
Maximum size of a transaction slot [64 byte blocks]              8
Automatic crash recovery                                         NO
```

Figure 9-3 Terminal Listing of SHOW Command Dialog

9.2.5 Copying or Renaming an Existing Transaction Processor

The COPY Command
The RENAME Command

The COPY RENAME commands duplicate or rename an existing transaction processor definition. These commands are most useful when you are testing a new version of a transaction processor, but need to retain the old version in its existing state for security and backup purposes.

The terminal dialog for the COPY and RENAME commands consists of two questions:

Old transaction processor name?

You must supply the name of the transaction processor that you want to copy (rename). This is the 1- to 6-character alphanumeric name that you assigned to the transaction processor when it was originally created, renamed, or copied.

New transaction processor name?

TPDEF asks you to specify the new name of the processor; the name must be a 1- to 6-character alphanumeric string. This name appears in the TPDEF.TPF record that corresponds to this processor. The name also becomes the filename for the definition data files created by TPDEF. For example, if you copy (rename) a processor called STDAPP giving it a new name SAMPLE, TPDEF copies (renames) the existing definition data files for STDAPP, calling the new files:

SAMPLE.STA
SAMPLE.TRA
SAMPLE.FIL
SAMPLE.WOR
SAMPLE.AUT
SAMPLE.FDF

Once you specify an old and new transaction processor name, TPDEF exits and executes a number of DCL COPY (RENAME) commands to assign new names to the definition files.

Figure 9-4 shows the TPDEF terminal dialog for the COPY command. The RENAME command is similar, except that the old transaction process no longer exists.

```
>@[1,2]TPDEF (RET)
>RUN $TPDEF

TPDEF V1.0
Transaction processor definition utility

Command <EXIT>? COPY (RET)

Old transaction processor name? SAMPL1 (RET)

New transaction processor name? SAMPLE (RET)

>
>COPY [1,300]SAMPL1.MAC [1,300]SAMPLE.MAC
>COPY [1,300]SAMPL1.FDF [1,300]SAMPLE.FDF
>COPY/CONTIGUOUS [1,300]SAMPL1.TSK [1,300]SAMPLE.TSK
>COPY/CONTIGUOUS [1,300]SAMPL1.TIM [1,300]SAMPLE.TIM
>MERGE [1,300]SAMPL1.STA/INDEXED/KEY:NUMBER:1 [1,300]SAMPLE.STA/INDEXED
>MERGE [1,300]SAMPL1.TRA/INDEXED/KEY:NUMBER:1 [1,300]SAMPLE.TRA/INDEXED
>MERGE [1,300]SAMPL1.FIL/INDEXED/KEY:NUMBER:1 [1,300]SAMPLE.FIL/INDEXED
>MERGE [1,300]SAMPL1.WOR/INDEXED/KEY:NUMBER:1 [1,300]SAMPLE.WOR/INDEXED
>MERGE [1,300]SAMPL1.AUT/INDEXED/KEY:NUMBER:1 [1,300]SAMPLE.AUT/INDEXED
>SET PROTECTION [1,300]SAMPLE.* (SYSTEM:RWED,OWN:RWED,GROUP:RWED,WORLD:RWED)
>DELETE TEMP.*;*
>@ <EOF>
```

Figure 9-4 TPDEF Utility COPY Dialog Listing

9.2.6 Deleting A Transaction Processor Definition

The DELETE Command

To delete an existing transaction processor definition, you use the TPDEF utility, invoking the DELETE command. The DELETE command dialog consists of one question:

Old transaction processor name?

You must supply the name of the transaction processor that you want tc delete. This is the 1- to 6-character alphanumeric name that you assigned to the transaction processor when it was created, renamed, or copied.

Once you specify the name of the trnasaction processor, the utility generates a series of DCL commands, and executes them, deleting the transaction processor and associated definition files. The TPDEF utility also deletes the transaction processor record in [1,1 TPDEF.TPF].

Figure 9-5 shows the terminal dialog to delete a transaction processor definition.

```
>@[1,2]TPDEF (RET)
>RUN $TPDEF

TPDEF V1.0
Transaction processor definition utility

Command <EXIT>? DELETE (RET)

Old transaction processor name? SAMPLE (RET)



>
>DELETE [1,300]SAMPLE.MAC;*
>DELETE [1,300]SAMPLE.FDF;*
>DELETE [1,300]SAMPLE.TSK;*
>DELETE [1,300]SAMPLE.TIM;*
>DELETE [1,300]SAMPLE.STA;*
>DELETE [1,300]SAMPLE.TRA;*
>DELETE [1,300]SAMPLE.FIL;*
>DELETE [1,300]SAMPLE.WOR;*
>DELETE [1,300]SAMPLE.AUT;*
>DELETE TEMP.*;*
>@ <EOF>
>
```

Figure 9-5 Terminal Listing of DELETE Command Dialog

CHAPTER 10
STATION DEFINITION

10.1 STATION CONCEPTS

A *station* receives messages for a component of a transaction processor. Stations are supported by system software within a transaction processor. In order to properly support the number and types of stations used by a transaction processor, you must first define the stations using the STADEF utility.

The TRAX operating system allows you to specify stations for seven different classes of transactior processor components. Stations may be defined for:

1. A Transaction Step Task (TST) that serves as an application program.
2. A terminal station that controls execution of an application terminal.
3. A master link station that sends messages to other transaction processors.
4. A slave link station that initiates transactions upon receipt of link messages from other transaction processors.
5. A submit batch station that initiates a batch job from a transaction instance.
6. A slave batch station that initiates a transaction instance when it receives a message from a batch job.
7. A mailbox station that stores temporary data in a location accessible to all transaction processors.

A station definition consists of a set of station parameters that make up a station definition record. The definition data file [1,300] tpname.STA contains the set of station definitions for a transaction processor. For example, all station definition records for the transaction processor SAMPLE are contained in the file [1,300] SAMPLE.STA.

10.2 THE STADEF UTILITY

The STADEF utility program adds, modifies, displays, and deletes station definition records contained in [1,300] tpname.STA.

10.2.1 Invoking the STADEF Utility

The STADEF utility is invoked from a support environment terminal by typing the command:

```
>RUN $STADEF
```

STADEF responds with a version number and an identification line, followed by the first dialog question:

```
STADEF V1.0
Station Definition Utility

Transaction Processor name?
```

You must answer with a 1- to 6-character name of an existing transaction processor.

STADEF then opens the station definition file for that transaction processor, [1,300] tpname.STA. Once the station definition file is opened, STADEF asks you to invoke one of six commands:

Command?

You can respond by typing one of the following command keywords:

- ADD – The ADD command defines the attributes of a station and creates a station definition record.
- DELETE – The DELETE command removes an existing station definition record from the .STA file.
- EDIT – The EDIT command examines and modifies the attributes of a previously defined station definition record.
- INDEX – This command displays the names of all stations defined for the transaction processor .STA file that you are using. The listing appears on your terminal (TI:).
- PRINT – The PRINT command lists the attributes of one or all station definitions on the console listing device (CL0:). This command is useful for documenting the components of a transaction processor for reference.
- SHOW – The SHOW command lists the attributes of one or all stations on your terminal (TI:).

After you specify a command keyword, the STADEF dialog continues. The order of the dialog questions depends on the command you have invoked. The remainder of this chapter is organized by the four major STADEF functions:

1. The ADD and EDIT commands, Section 10.2.2
2. The INDEX command, Section 10.2.3
3. The PRINT and SHOW commands, Section 10.2.4
4. The DELETE command, Section 10.2.5

10.2.2 Adding or Editing a Station Definition

The ADD Command
The EDIT Command

If you respond to the command question by typing ADD or EDIT, the dialog begins by asking you for the station name:

Station name?

You respond by typing the 1- to 6-character name of the station you are defining (editing).

With the EDIT command, STADEF shows the named station's type. If you are ADDing a station definition, STADEF asks you to specify:

Station type?

You can respond to this question with one of seven station type keywords:

TST
TERMINAL
MASTER LINK
SLAVE LINK
SUBMIT BATCH
SLAVE BATCH
MAILBOX

At this point, the STADEF dialog breaks into seven different processing paths by the station type you specify.

The remainder of this section presents the dialogue required to ADD or EDIT a station definition according to the type of station you wish to define. To review the STADEF dialog, refer to the section describing the station type you specified.

10.2.2.1 Terminal Stations – A terminal station holds messages sent from a TST to an application terminal. In addition, the definition of a terminal station determines attributes of the associated physical device in the context of the transaction processor (whether the terminal can display system messages, for example).

Most application terminals at an installation perform identical functions in a transaction processing environment. STADEF provides a facility to create a single station definition record for a group of identically functioning terminals.

To define a group of similar terminal stations, you must specify a station name that has four alphanumeric characters and ends in two asterisks (**). For example:

Station name? TERM**

When STADEF receives a name of this type and the station type specification TERMINAL, it continues the dialog by asking:

Number of stations in group?

You respond with the number of terminals (1-64) you wish to define using the group name TERM**. The remainder of the dialog for a group of terminal stations is the same as the dialog for a single station.

The individual station names for a group of terminal stations are derived from the characters preceding the asterisks (the group name) and a 2-character number. For example, if the group name TERM contains 10 terminals, STADEF generates 10 stations named TERM01, TERM02, TERM03, and so on up to TERM10.

If you are specifying a terminal station name other thatn a group name, the name must by 1 to 6 ASCII characters and the last character must be a letter in the range A through Z.

After the name, type, and group number specifications are made, the ADD and EDIT dialogs continue by asking:

Name of device?

You respond with the logical terminal name of the associated application terminal. Every terminal is assigned a logical terminal name at system generation. When you define a group of stations, this question is repeated for every station in the group.

If you are defining a terminal station that supports a VT62 interactive terminal, specify the logical terminal name assigned to that terminal.

If you are defining a terminal station for an LA-180P DECPRINTER, you specify the logical terminal name of the VT62 that drives the LA-180P.

When you are defining an LA-180S DECPRINTER, specify the logical terminal name assigned to the LA-180S during the system generation process.

The logical terminal names are assigned at during system generation (See the TRAX System Generation Manual).

Once you specify the logical name (or names for a group definition), the STADEF dialog continues by asking the question:

Input or output device <BOTH>?

TRAX supports two types of terminal stations: interactive terminals that perform both input and output, and hard-copy terminals that can be used for output-only.

You can answer this question with the keyword BOTH, specifying that the terminal can perform input and output, or the keyword OUTPUT, which tells STADEF that the terminal is an output-only device.

Enable system massages <YES>?

If you want an interactive application terminal to display system messages, answer Y, YE, or YES. If you don't want system messages displayed on application terminals, answer N or NO. System messages cannot be received by output-only terminal stations.

Work class name <NONE>?

If you wish to associate a particular work class with a terminal station (or group of stations), you can specify that work class name in response to this question. The name must be a 1- to 6-character name that is also defined using the WORDEF utility. See Chapter 13 for a complete discussion of work classes and application security tools.

Will the terminal run a dedicated transaction <NO>?

A terminal station can be dedicated, or allowed to perform only one specific type of transaction. If you answer YES to this question, you must end the dialog with the name of the dedicated transaction in response to the question:

Transaction name?

The dialog ends, and you are returned to the "Station name?" question.

However, if you answer NO to the "dedicated transaction" question, the STADEF dialog continues by asking you to specify:

Form name?

This is the name of a form (the "initial form") that is displayed when the terminal is not actively processing. The name should identify a transaction selection form.

After you answer this question, STADEF adds or updates the station definition record to the file and returns to the question:

Station name?

allowing you to continue ADDing or EDITing station definitions.

Figure 10-1a shows a Terminal Station Specification Sheet that specifies the parameters used to define a group of four terminal stations with identical attributes.

Figure 10-1b lists the STADEF terminal dialog that adds a group of terminal stations to the station definition file [1,300] SAMPLE.STA.

TERMINAL STATION SPECIFICATION SHEET

Transaction Processor Name: SAMPLE

Station Name	Device Name	Device Type	System Messages	Work Class	Run A Dedicated Transaction?
TERM**	ØØ1V62 TO ØØ4V62	[X] – BOTH [ ] – OUTPUT	[X] – YES [ ] – NO	SIGNON	[ ] Yes – Transaction Name: [X] No – Initial Form Name: SELECT
		[ ] – BOTH [ ] – OUTPUT	[ ] – YES [ ] – NO		[ ] Yes – Transaction Name: [ ] No – Initial Form Name:
		[ ] – BOTH [ ] – OUTPUT	[ ] – YES [ ] – NO		[ ] Yes – Transaction Name: [ ] No – Initial Form Name:
		[ ] – BOTH [ ] – OUTPUT	[ ] – YES [ ] – NO		[ ] Yes – Transaction Name: [ ] No – Initial Form Name:
		[ ] – BOTH [ ] – OUTPUT	[ ] – YES [ ] – NO		[ ] Yes – Transaction Name: [ ] No – Initial Form Name:
		[ ] – BOTH [ ] – OUTPUT	[ ] – YES [ ] – NO		[ ] Yes – Transaction Name: [ ] No – Initial Form Name:
		[ ] – BOTH [ ] – OUTPUT	[ ] – YES [ ] – NO		[ ] Yes – Transaction Name: [ ] No – Initial Form Name:
		[ ] – BOTH [ ] – OUTPUT	[ ] – YES [ ] – NO		[ ] Yes – Transaction Name: [ ] No – Initial Form Name:
		[ ] – BOTH [ ] – OUTPUT	[ ] – YES [ ] – NO		[ ] Yes – Transaction Name: [ ] No – Initial Form Name:
		[ ] – BOTH [ ] – OUTPUT	[ ] – YES [ ] – NO		[ ] Yes – Transaction Name: [ ] No – Initial Form Name:

Figure 10-1a Terminal Station Specification Sheet – "SAMPLE"

```
>
>RUN $STADEF (RET)

STADEF  V1.0
Station Definition Utility

Transaction Processor name? SAMPLE (RET)

Command? ADD (RET)

Station name? TERM** (RET)

Station type? TERMINAL (RET)

Number of stations in group < 1 >? 4 (RET)

Name of device? 001V62 (RET)

Name of device <>? 002V62 (RET)

Name of device <>? 003V62 (RET)

Name of device <>? 004V62 (RET)

Input or output device <BOTH>? (RET)

Enabled for system messages <YES>? (RET)

Work class name <NONE>? SIGNON (RET)

Will terminal run a dedicated transaction <NO>? (RET)

Form name? SELECT (RET)

Record (TERM**) ADDed.

Station name? (CTRL/Z)

>
```

Figure 10-1b Listing of Terminal Dialog to Add a Terminal Station

10.2.2.2 TST Stations – You incorporate a TST into a transaction processor by defining an associated TST station. When a message arrives at a TST station, the processor activates the TST so it can process the message. A TST cannot be initiated in any other way.

See Chapter 1 and Chapter 11 for a description of routing lists, that tell the processor which TST stations receive each exchange message in a given transaction.

You can define up to the maximum number of TST stations allowed by the limit specified in the transaction processor definition.

If you specify TST is response to the "Station type?" question, the following dialog gathers parameters that describe the TST station.

Station priority <128>?

You must specify a priority value for the station. The priority of a TST station is relevant only when more than one TST is waiting to be brought into memory. When memory becomes available, the processor chooses the TST with the highest priority to run first, as long as that TST can fit into the space that has been freed. If the highest priority TST is too large, the processor selects the next highest priority TST that will fit. The default priority for a TST is 128, but you can specify any number between 1 and 250, with 1 being low priority and 250 being high. The dialog continues by asking you to specify the:

Task image file specification?

You respond with the complete specification of the TST's task-image file, which is the output file produced by the TSTBLD utility (See Chapter7).

STADEF assumes certain default values. Unless you specify otherwise, the task image file is assumed to reside on the system device in UFD [1,300], with a filetype of .TSK, and it is assumed to be the highest existing version number. If you accept the assumed defaults, you can answer this question by simply specifying a 1- to 9-character filename.

Maximum number of active copies?

You are asked to specify the maximum number of TST copies that can be active concurrently. Normally, if a TST is active when a message arrives at its station, the message must wait for processing until the TST has completed. However, if you specify that multiple TST copies can be active at the same time, the processor activates a copy of the associated TST as soon as another message arrives at the TST's station. The processor can activate the number of copies that you specify in response to this question.

TST serially reusable <NO>?

If you specify a TST as serially reusable, the transaction processor can rerun a memory-resident copy of a serially reusable TST, rather than having to retrieve a new copy from disk.

Figure 10-2a shows a TST Station Specification Sheet used by the change customer transaction that specifies the parameters needed to define the TST stations RDCUST, VALIDC, and REWRIT.

Figure 10-2b lists the STADEF terminal dialog that adds the three TST stations in Figure 10-2a to the station definition file [1,300]SAMPLE.STA.

TST STATION SPECIFICATION SHEET

Transaction Processor Name: SAMPLE

Station Name	Station Priority	Task Image File Specification	No. Active Copies	Serially Reusable?
RDCUST	128	SYØ:[1,3ØØ]RDCUST.TSK;Ø	4	[X] – YES [ ] – NO
VALIDC	128	SYØ:[1,3ØØ]VALIDC.TSK;Ø	1	[X] – YES [ ] – NO
REWRIT	128	SYO:[1,300]REWRIT.TSK;Ø	4	[ ] – YES [ ] – NO
		:[,] . ;		[ ] – YES [ ] – NO
		:[,] . ;		[ ] – YES [ ] – NO
		:[,] . ;		[ ] – YES [ ] – NO
		:[,] . ;		[ ] – YES [ ] – NO
		:[,] . ;		[ ] – YES [ ] – NO
		:[,] . ;		[ ] – YES [ ] – NO
		:[,] . ;		[ ] – YES [ ] – NO
		:[,] . ;		[ ] – YES [ ] – NO
		:[,] . ;		[ ] – YES [ ] – NO
		:[,] . ;		[ ] – YES [ ] – NO
		:[,] . ;		[ ] – YES [ ] – NO
		:[,] . ;		[ ] – YES [ ] – NO

Figure 10-2a TST Station Specification Sheet – "SAMPLE"

```
>RUN $STADEF (RET)

STADEF  V1.0
Station Definition Utility

Transaction Processor name? SAMPLE (RET)

Command? ADD (RET)

Station name? RDCUST (RET)

Station type? TST (RET)

Station priority < 128 >? (RET)

Task image file specification? [1,300]RDCUST.TSK (RET)

Maximum number of active copies < 96 >? 4 (RET)

TST serially reusable <NO>? YES (RET)

Record (RDCUST) ADDed. (RET)

Station name? VALIDC (RET)

Station type? TST (RET)

Station priority < 128 >? (RET)

Task image file specification? [1,300]VALIDC.TSK (RET)

Maximum number of active copies < 96 >? 1 (RET)

TST serially reusable <NO>? YES (RET)

Record (VALIDC) ADDed.

Station name? REWRIT (RET)

Station type? TST (RET)

Station priority < 128 >? 128 (RET)

Task image file specification? [1,300]REWRIT.TSK (RET)

Maximum number of active copies < 96 >? 4 (RET)

TST serially reusable <NO>? YES (RET)

Record (REWRIT) ADDed. (RET)

Station name? (ESC)

Command? EXIT (RET)
>
```

Figure 10-2b Listing of Terminal Dialog to Add a TST Station

10.2.2.3 Master Link Stations – A master link station forwards messages to an associate remote transaction processor or IBM system.

Each master link station can handle up to 64 sublinks, where each sublink corresponds to a potential slave transaction instance. The master link maintains each sublink until the corresponding slave transaction instance is completed.

The maximum number of master link stations allowed for each transaction processor is determined by a parameter defined in the TPDEF utility (See Chapter 9).

To define a master link station, you answer the following questions:

Connected to?

The identity of the remote host connected by the master link. There are three possible types of identity.

- If the link is to a transaction processor in another computer, you must specify the node name of that system. The name must identify the host system rather than the transaction processor within that system; a node name is a 1- to 6-character alphanumeric string defined at system generation.
- If the master link connects to a transaction processor located within the same host computer, you respond by typing the keyword LOCAL.
- If the master link connects to an IBM system, you must specify IBM.

Number of sublinks?

The number of sublinks that the master link can maintain simultaneously. For each sublink, there must be an available slave link station in the remote processor or an available line number in the remote IBM system. A master link can support up to 64 simultaneous sublinks.

Slave transaction processor name?

You must specify the name of the remote transaction processor (if applicable). The name is the 1- to 6-character alphanumeric string assigned to the processor by the TPDEF utility (see Chapter 9.).

IBM line number?

If you are communicating with an IBM system, you must specify an IBM line number for each sublink (if applicable). The STADEF utility asks this question as many times as required to obtain one line number for every sublink.

Figure 10-3a shows a Master Link Station Specification Sheet that specifies the parameters needed to define a master link station in a local transaction processor.

Figure 10-3b lists the STADEF terminal dialog that adds a master link station to the station definition file [1,300] SAMPLE.STA.

MASTER LINK STATION SPECIFICATION SHEET

Transaction Processor Name: SAMPLE

Station Name	Connected to Slave Link Type	Number of Sublinks	
MASLNK	☐ – Node Name:		Slave TP Name
	☒ – Local	01	Slave TP Name TEST01
	☐ – IBM		Line Number
	☐ – Node Name:		Slave TP Name
	☐ – Local		Slave TP Name
	☐ – IBM		Line Number
	☐ – Node Name:		Slave TP Name
	☐ – Local		Slave TP Name
	☐ – IBM		Line Number

Figure 10-3a Master Link Station Specification Sheet – "SAMPLE"

```
>RUN $STADEF (RET)

STADEF  V1.0
Station Definition Utility


Transaction Processor name? SAMPLE (RET)

Command? ADD (RET)

Station name? MASLNK (RET)

Station type? MASTER LINK (RET)

Connected to? LOCAL (RET)

Number of sublinks? 1 (RET)

Slave transaction processor name? TEST01 (RET)

Record (MASLNK) ADDed.

Station name? (CTRL/Z)

>
```

Figure 10-3b Listing of Terminal Dialog to Add a Master Link Station

10.2.2.4 Slave Link Stations – A slave link station receives exchange messages forwarded by a master link station. When a slave link station receives a message it initiates a transaction instance.

Each processor supports the maximum number of slave link stations specified in the TPDEF utility (see Chapter 9). The STADEF dialog to define a slave link station consists of the station name and type. No other parameters are required.

10.2.2.5 Submit Batch Station – Each transaction processor can include one submit station. The submit batch station provides a means for your transaction processor to submit a batch job for processing in the TRAX environment.

When defining the submit batch station, you only need to specify the station name and type. No other parameters are required. See Chapter 5 for the procedure required to submit a batch job from a transaction instance.

10.2.2.6 Slave Batch Stations – Slave batch stations perform the reverse function of the submit batch station; they allow batch programs to access the transaction processor. A batch job can initiate a single-exchange transaction by sending an exchange message to a batch slave station. The transaction processor then initiates a transaction instance. The slave batch station is the source station for that transaction instance.

A transaction processor can support up to the maximum number of batch slave stations specified for the processor through the TPDEF utility. (See Chapter 9.)

The dialog required to define a slave batch station consists of the station name and station type. No other parameters are required.

10.2.2.7 Mailbox Stations – To define a mailbox station, the STADEF utility asks you to specify a station name and station type. If you are defining a mailbox station, STADEF then asks you to specify:

Maximum message size [bytes]? <64>

You can press the RETURN key <RET> to accept the default value of 64 bytes per message, or you may specify a value in the range 64 to 8192 bytes.

When defining the transaction processor, you specify the maximum number of mailbox stations used by the transaction processor (See Chapter 9.).

Figure 10-4a shows a Special Purpose Specification Sheet that specifies the parameters to define a mailbox station named "OVRCRL".

Figure 10-4b lists the STADEF terminal dialog that adds a mailbox station to the station definition file [1,300] SAMPLE.STA

SPECIAL PURPOSE STATION SPECIFICATION SHEET

Transaction Processor Name: SAMPLE

Station Name	Station Type
OVRCRL	[X] – MAILBOX – Max. Message Size: [ ][ ]64 bytes (Must be 64–8192).
S/B OVRCRL	[ ] – SLAVE LINK
	[ ] – SUBMIT BATCH
	[ ] – SLAVE BATCH
[ ][ ][ ][ ][ ][ ]	[ ] – MAILBOX – Max. Message Size: [ ][ ][ ][ ] bytes (Must be 64–8192).
	[ ] – SLAVE LINK
	[ ] – SUBMIT BATCH
	[ ] – SLAVE BATCH
[ ][ ][ ][ ][ ][ ]	[ ] – MAILBOX – Max. Message Size: [ ][ ][ ][ ] bytes (Must be 64–8192).
	[ ] – SLAVE LINK
	[ ] – SUBMIT BATCH
	[ ] – SLAVE BATCH

Figure 10-4a Special Purpose Station Specification Sheet – "SAMPLE'

```
>RUN $STADEF (RET)

STADEF  V1.0
Station Definition Utility


Transaction Processor name? SAMPLE (RET)

Command? ADD (RET)

Station name? OVRCRL (RET)

Station type? MAILBOX (RET)

Maximum message size [bytes] <64>? (RET)

Record (OVRCRL) ADDed.

Station name? (ESC)

Command? EXIT (RET)
>
```

Figure 10-4b Listing of Terminal Dialog to Add a Mailbox Station

10.2.3 Listing the INDEX of Defined Stations

The INDEX Command

When you specify the INDEX command keyword, the STADEF utility displays all station names and types in the .STA file associated with the transaction processor name that you supplied when you invoked the utility. This listing appears on the support environment terminal where you are running the STADEF utility.

Figure 10-5 shows the STADEF terminal dialog that lists the INDEX of station definitions in the station definition file [1,300] SAMPLE.STA.

```
Command? INDEX (RET)
Station : MASLNK        Type : MASTER LINK        Repeat count :  1

Station : OVRCRL        Type : MAILBOX            Repeat count :  1

Station : RDCUST        Type : TST                Repeat count :  1

Station : REWRIT        Type : TST                Repeat count :  1

Station : VALIDC        Type : TST                Repeat count :  1

Station : TERM**        Type : TERMINAL           Repeat count :  4


Command? EXIT (RET)
>
```

Figure 10-5 STADEF Utility Listing of INDEX Command Terminal Dialog

10.2.4 Printing or Showing a Station Definition

The PRINT Command
The SHOW Command

The SHOW and PRINT commands perform the same operations but direct their output to different devices. SHOW lists the attributes of a station definition at your support environment terminal, while PRINT provides a hard-copy listing on the system device assigned to logical unit CL0:. This is usually the line printer. The terminal dialog for these commands is a single prompt:

Which station <ALL>?

If you provide a station name in response to the prompt, the SHOW and PRINT commands display information for that station only. Pressing the RETURN key <RET> produces a listing of the station attributes for the transaction processor name specified when you invoked the STADEF utility.

Figure 10-6 lists the STADEF terminal dialog that prints station definitions in the station definition file [1,300] SAMPLE.STA on the system device (CL0:).

```
>RUN $STADEF (RET)

STADEF  V1.0
Station Definition Utility


Transaction Processor name? SAMPLE (RET)

Command? PRINT (RET)

Which station <ALL>? (RET)

Command? (CTRL/Z)

>
```

Figure 10-6 STADEF Utility Listing of PRINT Command Terminal Dialog

```
Transaction Processor Name: SAMPLE
Station name: ALL

Station : MASLNK        Type : MASTER LINK          Repeat count :  1

                        Node : LOCAL     Transaction processor : TEST01
                        Number of sublinks :  1


Station : OVRCRL        Type : MAILBOX              Repeat count :  1

                        Maximum message size:  64  bytes

Station : RDCUST        Type : TST                  Repeat count :  1

                        Task image file : SY:[1,300]RDCUST.TSK;0
                        Maximum number of active copies :  4
                        Station priority :  128
                        TST is serially reusable

Station : REWRIT        Type : TST                  Repeat count :  1

                        Task image file : SY:[1,300]REWRIT.TSK;0
                        Maximum number of active copies :  4
                        Station priority :  128
                        TST is serially reusable

Station : VALIDC        Type : TST                  Repeat count :  1

                        Task image file : SY:[1,300]VALIDC.TSK;0
                        Maximum number of active copies :  1
                        Station priority :  128
                        TST is serially reusable

Station : TERM**        Type : TERMINAL             Repeat count :  4

                        Devices :  4

                                              Terminal ID
                                              001V62
                                              002V62
                                              003V62
                                              004V62

                        Allowed operations : INPUT/OUTPUT
                        Enabled for system messages
                        Work class name : SIGNON
                        Initial FORM : SELECT
```

Figure 10-6 (cont.) Line Printer Output from STADEF PRINT Command.

10.2.5 Deleting a Station Definition

The DELETE command

The DELETE command removes a station definition from the .STA file named by the transaction processor specified when you invoked STADEF. To delete a station definition, simply type the station name you wish to delete.

Figure 10-7 lists the STADEF terminal dialog used to delete all terminal station definitions in the group TERM** form the station definition file [1,300] SAMPLE.STA.

```
>
>RUN $STADEF (RET)

STADEF  V1.0
Station Definition Utility


Transaction Processor name? SAMPLE (RET)

Command? DELETE (RET)

Station name? TERM** (RET)

Record (TERM**) DELETED. (RET)

Station name? (ESC)

Command? EXIT (RET)
>
```

Figure 10-7 STADEF Utility Listing of DELETE Command Terminal Dialog

CHAPTER 11
TRANSACTION DEFINITION

11.1 TRANSACTION CONCEPTS

A transaction is a pre-defined unit of data processing performed by a transaction processor. Before you can invoke a transaction, it must be defined. A transaction definition includes:

- The names of all exchanges and associated forms presented to the user of the transaction type
- The initial order in which exchanges are presented.
- The stations which process the information for each exchange
- Actions to be taken upon completion of a transaction instance

You define transaction types using TRADEF, a utility that adds, edits, displays and deletes transaction definition records in the system information file [1,300] tpname.TRA.

11.1.1 The TRADEF Utility

Before you can define a transaction, you must first run the TPDEF utility to create a transaction processor definition. TPDEF creates the system information file [1,300] tpname.TRA, which is read and updated by the TRADEF utility.

You run TRADEF from a support environment terminal. TRADEF uses an interactive dialog to question you about the structure and parameters of the transactions you are defining. Your responses to these questions are used to create a transaction definition record, in the transaction definition file.

Each transaction type you define is identified by a unique 1- to 6-character transaction name.

Once you create a set of transaction definitions, you can use TRADEF to modify individual transaction definitions or to list information about the records in the transaction definition file.

NOTE

This utility updates information in the file [1,2] TPDEF.TPF. If a new version of either the TPDEF.TPF or tpname.TRA file is created, you must run TRADEF, specify the transaction processor name, and exit before attempting to install that transaction processor.

11.1.2 Invoking the TRADEF Utility

You invoke the TRADEF Utility from a support environment terminal by typing:

```
>RUN STRADEF
```

TRADEF responds with an identifying message, and begins the interactive dialog. The first question asks:

```
TRADEF V1.0
Transaction Definition Utility

Transaction processor name?
```

You reply by specifying the name of the transaction processor that uses the transaction you are defining. If you have not defined the transaction processor, you must exit from TRADEF, and run the TPDEF utility. (See Chapter 9 for a description of the TPDEF utility.)

Once you have specified the name of an existing transaction processor, TRADEF asks you to specify one of the six command keywords:

```
Command?
```

The command keywords recognized by TRADEF are:

- ADD – A transaction definition record is added to the file tpname.TRA
- EDIT – An existing transaction definition record may be modified.
- DELETE – An existing transaction definition record may be deleted.
- INDEX – The names of all existing transaction types are displayed on your terminal (TI0:).
- PRINT – The definition parameters of one or all existing transaction types are listed on the system listing device (CL0:).
- SHOW – The definition parameters of one or all existing transaction definitions are displayed on your terminal (TI0:).

The dialog questions asked by TRADEF differ depending on the command you specify. In the remainder of this chapter, the TRADEF utility dialog is explained according to function in four sections:

1. The ADD and EDIT Commands are described in Section 11.1.2
2. The INDEX command is described in Section 11.1.3
3. The PRINT and SHOW commands are described in Section 11.1.4
4. The DELETE command is described in Section 11.1.5

11.1.3 Adding or Editing a Transaction Definition

The ADD Command
The EDIT Command

If you invoke the ADD command, you enter a dialog that asks you to completely specify a new transaction definition. TRADEF begins the ADD dialog by asking you to specify the:

```
Transaction name?
```

The transaction name is a 1- to 6-character alphanumeric string that identifies the transaction you are defining.

The transaction name you specify must be unique within the structure of the transaction processor you named in the first question of the TRADEF dialog.

Once you have specified a transaction name, the dialog starts to question you about the general attributes of the transaction. First, it asks you if you want this transaction to have:

Exchange Recovery?

If you specify exchange recovery, the transaction processor executive copies each exchange message to disk before any processing takes place. If an error occurs during processing, a TST in the exchange can call the RESTRT library routine (See Chapter 4) and restart processing at the beginning of the exchange using the copy of the exchange message previously saved on disk.

If you include exchange recovery in a transaction definition, you must also allocate a system workspace for use by the transaction processor.

Log exchange messages <NO>?
Log other station messages <NO>?

TRADEF allows you to request the logging of station messages associated with the transaction. These include exchange messages derived from the initiator of the transaction and other station messages (mailbox, response, and report messages,) generated by TSTs. If a transaction definition enables message logging, the processor copies the specified message types to the journal file. (See Chapter 3.)

The TRAX system includes a utility called SHOLOG that selectively displays logged messages. SHOLOG is described in the *TRAX System Management and Operations Guide.*

For each active transaction instance, the processor allocates space in memory for a transaction slot. The slot contains space for the current exchange message, a transaction workspace, and a system workspace.

The TRADEF dialog has three questions that determine how much space the processor must allocate for each data structure in the slot.

The space allocated for the exchange message must be large enough to accommodate the largest exchange message created by any exchange in the current transaction type. The dialog to define the exchange message area is:

Maximum size of exchange message [bytes] < 64 >?

The sum of the largest exchange message size added to the size of the transaction workspace cannot exceed 8064 bytes.

The smallest exchange message you can specify is 1 byte. All values you specify are rounded up to the nearest multiple of 64 by TRADEF.

Transaction workspace size [bytes] < 0 >?

The transaction workspace allows TSTs executing as part of the same transaction instance to process information across exchanges. The transaction workspace is optional. The sum of the transaction workspace size added to the size of the largest exchange message in the transaction type cannot exceed 8064 bytes.

System workspace size [64 byte blocks] <0>?

The system workspace is used by the processor to store crash and exchange recovery information, and information needed to stage file updates. The formula for calculating the system workspace requirements is shown in Figure 11-1.

You must specify the system workspace size parameter as a number of 64-byte blocks. Figure 11-1 shows the worksheet used to calculate the system workspace for the change customer transaction.

Note that a question in the TPDEF utility (see Chapter 9.) asks you to specify the maximum size of the transaction slot as a whole. This value must be at least 64 bytes, and cannot exceed 8096 bytes. The sum of the three individual sizes you specify to TRADEF cannot exceed the maximum slot size set when the current transaction processor was defined.

11.1.3.1 Exchange Definition Parameters – After receiving appropriate answers to the questions described above, the TRADEF utility concerns itself with the definition of individual exchanges.

A transaction type can consist of up to 32 exchanges. Each exchange must be given a name, have a form associated with it (for terminal-initiated exchanges) have a routing list specified for it, and have timing and subsequent action parameters declared for it.

TRADEF performs exchange definition by entering a loop of questions that begin with the prompt:

Exchange command <DONE>?

You respond with one of six commands:

- ADD – Create an exchange definition. If other exchange definitions already exist, TRADEF adds the new definition to the end of the transaction's list of exchanges.
- DELETE – Delete an existing exchange definition by specifying the exchange label.
- EDIT – Modify the parameters of an existing exchange.
- INSERT – Create an exchange definition and insert the definition before a named exchange in the transaction's exchange list.
- LIST – Display at your terminal (TI0:) a list of the exchange labels defined for the transaction.
- DONE – Exit from the exchange definition loop. TRADEF writes the current transaction definition to [1,300]tpname.TRA, and returns to the question "Transaction name?."

As you define a set of exchanges, TRADEF builds up a list of exchange definitions containing up to 32 entries.

The ADD, INSERT, and EDIT commands use the same exchange definition dialog. When you define an exchange using ADD, TRADEF appends the resultant definition to the end of the list.

SYSTEM WORKSPACE WORKSHEET

Transaction Name: CHGCUS

You must complete this worksheet when your transaction is defined with staged files and/or exchange recovery. Use this procedure to calculate the system workspace size.

Bytes

1. If the transaction is defined with exchange recovery,enter the number of channels that will be connected.For example if the transaction opens file A on channel 01 and channel 02 and opens file B on channel 03 then enter 3.

   1 / No. connected channels * 34 bytes = 34

2. If the transaction uses files that are staged enter the number of times data records from these files are added,deleted,and updated. For example if a data record is added then subsequently updated in the same transaction instance then enter 2.

   1 / No. staged records * 18 bytes = 18

3. If the record that is staged is the result of an update then enter the length of the updated record and the total number of staged updates. For example if a 205 bytes record is updated twice enter 2*205=410. Added and deleted record contents are not stored in the system workspace.

   1 / No. updated records * 205 / Length bytes = 205

4. If the transaction is defined with exchange recovery then enter the number of mailbox messages retrieved by the transaction.

   — / No. mailbox messages * 8 bytes = ____

   Subtotal 257

   Add 16 bytes 16

   TOTAL 273

5. Divided the TOTAL by 64,round the result up to the next whole number,then add 1 block. The result is the system workspace size in blocks.

   SYSTEM WORKSPACE SIZE 6

Figure 11-1 System Workspace Worksheet for "CHGCUS"

The INSERT command, on the other hand, allows you to define an exchange and insert it before an existing exchange definition. The EDIT command allows to modify an existing exchange definition without changing its place in the exchange order. Before entering the exchange definition dialog, INSERT asks you to locate the new exchange with the question:

INSERT before which exchange [exchange label]?

The exchange label that you specify tells TRADEF that the new exchange is to be inserted before the specified exchange.

The EDIT exchange command allows you to change an exchange label. For this reason, it precedes the editing pass with the question:

EDIT which exchange [exchange label]?

The ADD, EDIT, and INSERT commands begin their common dialog at this point with the question:

Exchange Label ?

An exchange label is a 1- to 6-character alphanumeric string that uniquely identifies the exchange within the transaction's exchange list. Once the exchange name is specified, the dialog continues by asking you to specify the name of the form associated with this exchange.

Form Name<XXXXXX>?

A form name is a 1- to 6-character alphanumeric string that identifies a form definition in the file [1,300] tpname.FDF.

Note that TRADEF does not check for the existence of the form definition in the forms definition file. You must ensure that a form definition has been written and processed through the ATL utility (see the TRAX *ATL Language Reference Manual*). You only need to supply a form name if the transaction being defined must be able to run from an application terminal.

TRADEF defaults the form name to the string that defines the exchange label.

Forms provide the interface between the transaction processor and the user at the terminal. When transactions are initiated by a batch program, by a TST, with input data on the transaction selection form, or by a remote system, such an interface is unncessary. These types of sources must supply an exchange message already formatted for processing by an exchange routing list. For exchanges initiated from other than terminal stations, and for the first exchange following an exchange message created by a transaction selection form, you may specify NONE in response to this question.

To create the routing list of station names for the exchange, the TRADEF utility uses another loop of questions. Like the exchange definition loop, the routing list loop has its own set of commands called:

Destinations command <DONE>?

The five valid destinations commands are:

- ADD – Add a TST station (destination) to the end of the routing list.
- DELETE – Delete a destination from the routing list.
- INSERT – Insert a destination at a specific point within the routing list.
- LIST – Display the list of TST station names, with corresponding sequence numbers, at your terminal (TI:).
- DONE – Exit from the loop and proceed to the first subsequent action question (see below).

As you specify or delete each destination, TRADEF creates a list of the station names and assigns a sequence number to each entry. When you ADD an entry, TRADEF places the new destination at the bottom of the list.

When you INSERT an entry, TRADEF places the new destination in the routing list immediately prior to the designated sequence number. All destinations with the same or higher sequence number have their sequence number increased by one.

When destinations have been INSERTed or DELETEd, TRADEF resequences the remaining destinations accordingly.

If you invoke the DELETE or INSERT destination command, TRADEF asks you to specify a sequence number. To verify the number that you want to specify, enter the destination command LIST to obtain a current listing of all destinations and corresponding sequence numbers.

Destination sequence number?

When you are ADDing, or INSERTing a destination, TRADEF asks you to specify the name of the TST station to be included in the list. TRADEF does not check that the station named, a 1- to 6-character alphanumeric string, has been defined by the STADEF utility (see Chapter 10 for a description of station definition.)

Destination station name?

When TRADEF has completed the ADD, INSERT, DELETE, or LIST operation, it always returns to the question "Destinations command?". Enter DONE to proceed to the subsequent action questions in the exchange loop.

11.1.3.2 Subsequent Action Parameters – The processor checks the exchange definition to determine the next action to take. The exchange definition contains three subsequent action parameters that describe the transition from one exchange to another or the actions that take place at the close of a transaction.

Note that a response message from a TST overrides the subsequent action parameters specified in the exchange definition.

You are asked to answer a set of three questions regarding subsequent action.

Wait <YES>?

If you respond to this question by pressing only the RETURN key (which indicates YES), the transaction processor cannot close the exchange you are defining until a response message is received at the initiating station.

If you respond with N or NO, the transaction processor will immediately initiate the exchange specified as subsequent action in the transaction definition.

Repeat <NO>?

If you press the RETURN key <RET> which indicates NO, the transaction goes on to the exchange specified by the subsequent action parameter. If you respond with Y,YE, or YES, the current exchange is repeated until a stop repeat order is received from the terminal or from a TST.

Subsequent action <NEXT>?

You must specify one of the following actions:

- NEXT – When the exchange terminates, proceed to the next exchange defined for this transaction type.
- FIRST – When the exchange terminates, close the transaction instance, and start a new instance beginning at the first exchange of the current transaction type.
- INITIAL – When the exchange terminates, close the transaction instance, and return the station to its initial state. In the case where the transaction is initiated by a terminal station, this means the initial form specified in that terminal's station definition is displayed on the terminal screen.

The final parameter in an exchange definition is a question that asks you to specify:

Exchange time limit [minutes]?

The time limit is expressed in minutes. If the exchange is not completed within the time specified by this parameter, the transaction processor aborts the entire transaction instance. If for some reason the exchange being defined should not be limited, you can reply 0 (zero) to this question. The time limit refers to the time allowed for exchange processing in the transaction processor. The time spent entering data at the terminal has no meaning to the system.

Once you have answered this question, the exchange definition is finished. TRADEF reissues the "Exchange command <DONE>?" question, allowing you to perform additional exchange processing, or to exit from the exchange loop.

Figure 11-2a is a specification sheet describing a transaction type "CHGCUS" from the transaction processor "SAMPLE".

Figure 11-2b is the TRADEF terminal dialog used to implement the specification in Figure 11-2a.

TRANSACTION SPECIFICATION SHEET

Transaction Processor Name: SAMPLE

Transaction Name: CHGCUS

Exchange Recovery? [X] – YES [ ] – NO

Log Exchange Messages? [ ] – YES [X] – NO

Log Other Station Messages? [ ] – YES [X] – NO

Maximum Size of Exchange Message: 173 bytes

Transaction Workspace Size: 205 bytes

System Workspace Size
(Calculate according to formula on worksheet – "Calculating the system workspace".)
9 (64-byte blocks)

Transaction Slot Size Calculation:

Divide Exchange Message Size by 64 and round up:	3	blocks
Divide Transaction Workspace Size by 64 and round up:	4	blocks
Enter System Workspace Size:	5	blocks
Add to find Transaction Slot Size:	12	blocks

NOTE: A Transaction Exchange Definition should be prepared for each exchange associated with the transaction you have just defined.

TRANSACTION EXCHANGE DEFINITIONS

Exchange Label	Form Name	Destination Station List	Wait	Repeat	Subsequent Action	Time Limit
CHGEX1	CHCUS1	RDCUST	[X] WAIT [ ] NOWAIT	[ ] REPEAT [X] NOREPEAT	[ ] INITIAL [ ] FIRST [X] NEXT	2 MINS
GHGEX2	CHCUS2	VALIDC REWRIT	[X] WAIT [ ] NOWAIT	[ ] REPEAT [X] NOREPEAT	[ ] INITIAL [X] FIRST [ ] NEXT	4 MINS

Figure 11-2a Transaction Specification Sheet for "CHGCUS"

```
>RUN $TRADEF (RET)

TRADEF  V1.0
Transaction Definition Utility

Transaction Processor name? SAMPLE (RET)

Command? ADD (RET)

Transaction name? CHGCUS (RET)

Exchange recovery <NO>? YES (RET)

Log exchange messages <NO>? (RET)

Log other station messages <NO>? (RET)

Maximum size of exchange message [bytes] < 64 >? 173 (RET)

Size of transaction workspace [bytes] < 0 >? 205 (RET)

Maximum size of system workspace [64 byte blocks] < 0 >? 5 (RET)

Exchange command <DONE>? ADD (RET)

Exchange label? CHGEX1 (RET)

Form name <CHGEX1>? CHCUS1 (RET)

Destinations command <DONE>' ADD (RET)

Destination station name? RDCUST (RET)

Destinations command <ADD>? DONE (RET)

The following 3 questions deal with the subsequent action of this exchange. (RET)

Wait <YES>? (RET)

Repeat <NO>? (RET)

Subsequent action <NEXT>? (RET)

Exchange time limit [minutes] < 0 >? 2 (RET)

Exchange command <ADD>? (RET)

Exchange label? CHGEX2 (RET)

Form name <CHGEX2>? CHCUS2 (RET)

Destinations command <DONE>? ADD (RET)

Destination station name? VALIDC (RET)

Destinations command <ADD>? (RET)

Destination station name? REWRIT (RET)

Destinations command <ADD>? DONE (RET)
```

Figure 11-2b TRADEF Utility Listing of Add Command Terminal Dialog

```
The following 3 questions deal with the subsequent action of this exchange.
Wait <YES>? (RET)
Repeat <NO>? (RET)
Subsequent action <NEXT>? FIRST (RET)
Exchange time limit [minutes] < 0 >? 4 (RET)
Exchange command <ADD>? DONE (RET)
Record (CHGCUS) ADDed. (RET)
Transaction name? (ESC)
Command? (CTRL/Z)
```

Figure 11-2b (cont.) TRADEF Utility Listing of Add Command Terminal Dialog

11.1.4 Listing the INDEX of Defined Transactions

The INDEX Command

The INDEX command displays an INDEX of all transactions definitions in the file [1,300]tpname.TRA. (tpname is the transaction processor name specified when you invoked TRADEF.) The INDEX listing is displayed on your support environment terminal. Once the listing is printed, TRADEF reissues the "Command?" question.

Figure 11-3 lists the TRADEF terminal dialog that lists the INDEX of all transaction definitions currently in the transaction definition file [1,300] SAMPLE.TRA.

```
>RUN $TRADEF (RET)

TRADEF  V1.0
Transaction Definition Utility


Transaction Processor name? SAMPLE (RET)

Command? INDEX (RET)

Transaction : CHGCUS          Number of exchanges : 2
```

Figure 11-3 TRADEF Utility Listing of INDEX Command Terminal Dialog

11.1.5 Printing or Showing a Transaction Definition

The PRINT Command
The SHOW Command

The SHOW and PRINT commands perform the same operation but direct thier output to different devices. SHOW lists the attributes of a transaction definition at your support environment terminal, while PRINT provides a hard-copy listing on the system device assigned to logical unit CL0:. (This is usually the line printer or the LA-36 console terminal.) The terminal dialog for these commands cosists of a single prompt:

Which Transaction <ALL>?

If you provide a transaction name in response to the prompt, the SHOW and PRINT commands will display information for that transaction only. Responding to this prompt by pressing the RETURN key <RET> produces a complete listing of all transaction definitions currently in the transaction definition file [1,300] tpname.TRA. (tpname is the transaction processor name specified when you invoke the TRADEF utility.)

Figure 11-4 lists the TRADEF terminal dialog that displays all transaction definitions currently in the transaction definition file [1,300] SAMPLE.TRA on your terminal.

11.1.6 Deleting a Transaction Definition

The DELETE command

The DELETE command allows you to remove a transaction definition from the .TRA file named by the transaction processor specified when you invoked TRADEF. To delete a transaction definition, simply type the transaction name you wish to delete.

Figure 11-5 lists the TRADEF terminal dialog used to delete the transaction definition "BROWSE" from the transaction definition file [1,300] SAMPLE.TRA.

```
>
>RUN $TRADEF (RET)

TRADEF  V1.0
Transaction Definition Utility


Transaction Processor name? SAMPLE (RET)

Command? SHOW (RET)

Which transaction <ALL>? CHGCUS (RET)

Transaction : CHGCUS          Number of exchanges : 2

                      Exchange label    Form name    Number of destinations: 1

                              CHGEX1      CHCUS1              RDCUST

                              Subsequent action : WAIT,NOREPEAT,NEXT
                              Exchange time limit :  2  minutes

                      Exchange label    Form name    Number of destinations: 2

                              CHGEX2      CHCUS2              VALIDC
                                                              REWRIT

                              Subsequent action : WAIT,NOREPEAT,FIRST
                              Exchange time limit :  2  minutes

                              Maximum exchange message size : 192  bytes
                              TST workspace : 256  bytes
                              System workspace : 0  bytes

Command? EXIT (RET)
>
```

Figure 11-4 TRADEF Utility Listing of PRINT Command Terminal Dialog

```
Command? DELETE (RET)

Transaction name? CHGCUS (RET)

Record (CHGCUS) DELETED.

Transaction name? (CTRL/Z)

>
```

Figure 11-5 TRADEF Utility Listing of DELETE Command Terminal Dialog

CHAPTER 12
APPLICATION DATA FILE DEFINITION

12.1 FILE CONCEPTS

Two different kinds of application data files can be defined for use with a transaction processor:

- Permanent data files support the transaction processors ongoing data requirements. They form the data base that is modified by the transaction processor. Permanent data files are created by support environment programs and/or DCL commands. These files may have indexed, relative, or sequential organization. Permanent data files permit full record locking support, and every permanent data file defined for a transaction processor is opened automatically when that transaction processor is started, and remain open until the transaction processor is stopped, at which time they are closed.
- Work files on the other hand, provide transient data storage. Work files are dynamically created and populated by the current transaction processor. Work files can be dynamically opened, closed, created, or erased while a transaction processor is active. Work files that exist prior to the time a transaction processor is installed, are not opened until a TST explicitly opens them. Work files can be organized as relative files, or sequential files. Indexed work files are not allowed.

Files used by a transaction processor are created by one of three means:

- Any program (which creates the file).
- The DCL CREATE command.
- The RMSDEF utility program.

Information on how to write a support environment program is supplied in the Language Reference Manual and Users Guide for the language (COBOL or BASIC-PLUS-2) that you have selected. The DCL commands and RMS utilities are documented in the TRAX Support Environment User's Guide.

12.2 THE FILDEF UTILITY

The File Definition utility program FILDEF, provides the means for you to specify the attributes of both permanent data files, and work files that must be accessed by the transaction processor. FILDEF allows you to add, delete, and manipulate records in the transaction processor's File Definition [1,300] (tpname.FIL) file.

When you define a file using FILDEF, you must specify a logical name for the file. This logical name is used by all TSTs that reference the file, and by the transaction processor to access the file definition record.

A transaction processor's data management software can reference a file from a TST only by a logical name which has a corresponding record defined in the tpname.FIL file.

NOTE

The FILDEF utility updates information in the file [1,2] TPDEF.TPF. If a new version of the TPDEF or tpname.FIL file is created, you must run FILDEF, specify the transaction processor name, and exit.

12.3 INVOKING THE FILDEF UTILITY

You invoke the FILDEF utility from a support environment terminal by typing:

```
>RUN $FILDEF
```

FILDEF responds with an identifying line, followed by a question:

```
FILDEF V1.0
File Definition Utility

Transaction Processor name?
```

You respond to this first question by typing the name of the transaction processor that will access the application data files that you are defining.

The 1- to 6-character name supplied in response to this question is used to reference a file in UFD [1,300] known as tpname.FIL. This file is the File Definition File for the transaction processor you have just named.

Once you have supplied a valid transaction processor name, the FILDEF utility asks you to specify a command name which governs the subsequent utility dialog.

```
Command?
```

The following commands are valid responses to this question:

- ADD – Lets you create a definition record for an application data file and insert it in the .FIL file defined for the applicable transaction processor.
- DELETE – Allows you to delete the definition record of a previously defined application data file.
- EDIT – Allows you to modify the attributes of an existing definition record.
- EXIT – Lets you exit from the FILDEF utility.
- INDEX – Lets you display at your terminal (TI:), the logical names of all files defined in the file tpname.FIL.
- PRINT – Prints the attributes of one or all existing file definitions on the console listing device (CL0:). (CL0: is usually the line printer.)
- SHOW – Displays the attributes of one or all existing file definitions at your terminal (TI:).

The order of questions in the FILDEF utility dialog depends upon the command you invoke. The following section discusses the dialog sequence for:

1. The ADD and EDIT Commands are described in Section 12.3.1.
2. The INDEX Command is described in Section 12.3.2.

3. The PRINT and SHOW Commands are described in Section 12.3.3.
4. The DELETE Command is described in Section 12.3.4.

Each question is presented in the order it appears on the terminal when you are running the FILDEF utility.

12.3.1 Adding or Modifying a File Definition Record

The ADD and EDIT Commands

The ADD and EDIT commands are invoked by typing the command keyword in response to the Command? prompt. For example:

```
Command? ADD (RET)
         or
Command? EDIT (RET)
```

The ADD and EDIT Commands follow the same dialog sequence. When EDITing a file definition record, the currently specified value is displayed in angle brackets <> immediately to the left of the question mark. When ADDing a file definition, assumed responses are also enclosed in angle brackets.

The ADD (or EDIT) dialog begins by asking you to specify the:

```
Logical file name?
```

The logical file name is a unique 1- to 6-character alphanumeric string that you assign to an application data file. Your application program (TST) makes all subsequent references to a file using the logical file name.

Because this logical name is the transaction processor's only means of identifying a file, you must ensure that the logical names used in TSTs and the logical names supplied to FILDEF refer to the correct physical files.

```
File specification?
```

You are asked to supply the RMS file specification that identifies the file you want to associate with the logical file name.

You should supply the standard RMS specification for the file. A brief description of the information required is contained in this section. A more detailed discussion of file specifications is contained in Chapter 1 of this manual.

The standard form of an RMS File Specification is the following:

```
device: [group,member] filename.filetype;version
```

where:

device: is the name of the device where the volume on which the file resides is mounted. The device name takes the form: ddn: ; where dd is a 2-character device name

(DR or DB, for example), and n is the unit number. The trailing colon (:) must always be specified (MM0:, for example).

[group member] is the User File Directory (UFD) specification that identifies the directory used to hold the complete file specification. The required inputs for the UFD are the owner's group and member numbers, surrounded by brackets [ ] and separated by a comma.

filename is a 1- to 9-character alphanumeric file name.

.filetype is a 3-letter file type that identifies the general class describing the contents of the file (.DAT for data, for example).

;version is the octal version number from 1 to 77777 that distinguishes different versions of the same file.

Examples:

```
DB2: [300,221]PAYROL.DAT;35
DM1: [222,200]CUSTOM.DAT
```

When FILDEF asks for a file specification, you must supply all parts of the specification except the device and the version number. If they are omitted, the device name defaults to SY0:, while the version defaults to the highest existing version of the specified file. When specifying a file in the FILDEF utility, you must always supply the UFD, filename, and filetype.

Work file <NO>?

If you answer YES to this question, the definition of a work file requires no parameters other than those described to this point in the dialog. FILDEF writes the work file definition record to the .FIL file, then reissues, the Command? question.

If the file is not a work file, you should answer NO to the question about work files. FILDEF will then proceed to question you about the organization of the file being defined.

TRAX uses the RMS data management system that supports three types of file organization:

- Indexed
- Sequential
- Relative

Depending on the organization of the file being defined, you must answer YES or NO to the question:

Is the organization of the file indexed <NO>?

If you answered NO, FILDEF then proceeds to question you about concurrent file access. If you indicate that the file is indexed, FILDEF asks you to provide the following additional information:

Number of keys defined?

You must supply the number of keys defined for a record in the current file. Once you supply this information, the dialog continues with FILDEF asking you to specify the largest key size defined in the file:

Maximum key length?

This parameter is the length (in bytes) of the longest record key defined for the current file.

Once you specify the file organization characteristics, FILDEF continues with a series of questions related to multi-user file access rights.

Maximum concurrent file accesses < 1 >?

This parameter refers to the maximum number of different transaction instances you allow to access the file at any given time. If a transaction instance attempts to access a file, and that access exceeds the value you specified for concurrent accesses, then the transaction processor delays execution of that transaction instance until some other transaction instance releases the file. Normally, the user is not aware that such a delay has occurred.

Read only <NO>?

Although a TRAX system can have more than one transaction processor or support environment task active at the same time, only one such task may have write access to a particular file (given that your system is configured with sufficient memory resources). The different tasks and processors cannot have full access rights to each other's application data files. If one transaction processor or support environment task has write access to records within a file, other active tasks and transaction processors are allowed only read access to that file.

If you want to deny write access to this file for the transaction processor, you must answer YES to this question.

Fast deletions <YES>?

When a transaction instance requests that a record be deleted, the processor deletes the record in one of two ways. The record can be physically deleted immediately (on line), or it can be marked as deleted. Marking a record for deletion, called fast deletion, means that the deletion is a logical deletion; the record remains physically in the file. Logical deletions are explicitly faster than physical deletions.

Records marked for deletion remain in the file until you use the DCL COPY, MERGE, and/or CREATE commands.

Interval between attempts to access locked records[SECONDS]?

When a transaction instance applies a lock to a record, the record remains locked until that instance explicitly or implicitly releases the record lock.

When a subsequent transaction instance attempts to:

- Read from a file that does not permit read access,
- Read a record that has a staged lock applied,

the subsequent operation is placed in a lock wait queue. If the first lock is released before the time limit specified in this question, the waiting transaction instance is removed from the queue and given the record.

If the second transaction instance is still in the lock wait queue when the time limit expires, a locked record error is returned to the TST.

The value you specify in response to this question determines the interval a transaction will spend in the lock wait queue when attempting to access locked records in the current file. (See Chapter 3 for more information on record locking.)

Allow read access to locked records <YES>?

In some cases, denying any kind of access to a locked record is impractical. The FILDEF utility gives you the option of allowing read-only access to a locked record. In other words, the option allows another transaction instance to read a record that has been locked.

If you answer this question with a carriage return, read access is allowed to locked record in the file, except when the file is staged.

The final set of questions asked by FILDEF involves the recovery procedures you want enabled for the current file.

Journal after images <YES>?

Journalling provides a means of recovering application data files that have been corrupted in some manner. When you enable journalling for a file, the processor copies the transaction slots (after images) of all transaction instances that perform updates to that file. (The copies are made to a file reserved for journalling; see Section X.X.X. – Recovery procedures.) The transaction slots include all the information needed to recreate the corresponding transaction instances. In the event of a disk failure, for example, you can run RECOVR, a TRAX utility that uses the journal to restore the contents of a corrupted file to their condition prior to the transaction instance that caused the file to be incorrectly altered. (See the description of the RECOVR utility in the TRAX System Management and Operations Guide.)

Answering YES to the journalling question automatically assumes that you have specified staged file updates. If you answer NO, you are next asked whether or not you want staged updates to the file.

Stage updates <YES>?

Staging file updates allows you to protect a file from updates by a transaction instance that fails at some future point in its processing.

To stage updates, the processor delays the execution of file update requests until the transaction has completed successfully. The processor holds the requests in the system workspace portion of the transaction slot. If the transaction is successful, the processor performs the updates. If the transaction fails, the processor simply erases the update requests.

If you have enabled journalling for the file, FILDEF automatically specifies that updates to the file be staged. The information stored in the system workspace in order to stage file updates contains all the information the system needs to journal the file.

A file cannot be journalled unless the processor stages its updates.

After you have completed all questions in the FILDEF dialog, the file definition record will be added or updated to the file definition file. After the operation is complete, FILDEF stays in the ADD (or EDIT) command, and reissues the logical file name question. If you want to continue the current mode of operation (ADD or EDIT), simply specify the new file name. If you want to change to another command, type ESCAPE to reissue the Command question. For example:

```
Logical file name? <ESC>
Command?
```

Figure 12-1a shows a File Definition Specification Sheet for the customer file record used by the transaction processor "SAMPLE".

Figure 12-1b shows the terminal listing of the FILDEF dialog used to implement the specification shown in Figure 12-1a.

12.3.2 Listing the INDEX of File Definitions

The INDEX Command

Once the INDEX command is specified, its operation is automatic. After you type INDEX in response to the Command? question, FILDEF prints an index of file definitions on your support environment terminal. The Command prompt is then reissued, allowing you to EXIT, or to specify another command. Figure 12-2 shows the INDEX Command dialog from FILDEF.

12.3.3 Printing or Showing a File Definition Record

The PRINT and SHOW Commands

The PRINT and SHOW commands are invoked by typing the command keyword in response to the Command? prompt. For example:

```
Command? PRINT (RET)
         or
Command? SHOW (RET)
```

Both commands ask you to specify the logical file name after they are invoked. The question asked is:

```
Which file <ALL>?
```

FILE DEFINITION

Part One

Transaction Processor Name: SAMPLE

Logical Filename: CUSTOM

RMS File Specification: [ ]:[350,227]CUSTOM.DAT;[ ]

Work File?
- [ ] – Yes (Go to Part Two)
- [X] – No (Continue with next question)

Is This an Indexed File?
- [X] – Yes: No. of Keys [2]
  Maximum Key Length [30]
- [ ] – No: Sequential or Relative File

Maximum Concurrent File Accesses? [4]

Read-Only?
- [ ] – Yes
- [X] – No

Fast Deletions?
- [ ] – Yes
- [X] – No

Lock Interval [2] seconds

Read Access to Locked Records?
- [ ] – Yes
- [X] – No

Journal?
- [ ] – Yes (Go to Part Two)
- [X] – No (Continue with next question)

Staged File Updates?
- [ ] – Yes
- [X] – No

Part Two

File Channel Assignment

Description of File Contents: CUSTOMER FILE RECORD

Assigned I/O Channel Number [3]

Figure 12-1a File Definition Specification for "CUSTOM"

```
>RUN $FILDEF (RET)

FILDEF  V1.0
File Definition Utility

Transaction Processor name? SAMPLE (RET)

Command? ADD (RET)

Logical file name? CUSTOM (RET)

File specification? [350,227]CUSTOM.DAT (RET)

Work file <NO>? (RET)

Is the organization of this file INDEXED <NO>? Y (RET)

Number of keys defined? 2 (RET)

Maximum key length? 30 (RET)

Maximum concurrent file accesses < 1 >? 4 (RET)

Read only <NO>? (RET)

Fast deletions <YES>? NO (RET)

Interval between attempts to access locked record [SECONDS]? 2 (RET)

Allow read access to locked records <YES>? NO (RET)

Journal after images <YES>? NO (RET)

Stage updates <YES>? NO (RET)

Record (CUSTOM) ADDed.
```

Figure 12-1b FILDEF Utility Listing ADD Command Terminal Dialog

If you answer with a carriage return (RET), every file definition record contained in the current tpname.FIL file is displayed. If the PRINT command has been specified, the display takes the form of a listing sent to the console listing device (CL0:). If the SHOW command is specified, the file information is displayed directly on your support environment terminal.

If you specify a previously defined logical file name in response to this question, only that file definition record is displayed.

```
Command? INDEX (RET)
Logical name : CUSTOM          Specification : [350,227]CUSTOM.DAT
```

Figure 12-2 FILDEF Utility Listing of INDEX Command Dialog

After displaying the requested file definition record, FILDEF returns you to the Command? question.

Command?

You can respond by typing EXIT, or specify the name of a FILDEF command.

Figure 12-3 shows the terminal dialog used to display a file definition on a support environment terminal.

```
Command? SHOW (RET)
Which file <ALL>? CUSTOM (RET)
Logical name : CUSTOM          Specification : [350,227]CUSTOM.DAT
                               Organization is indexed
                               Number of keys defined : 2
                               Maximum key length : 30
                               Maximum concurrent file accesses : 4
                               Locked record retry interval : 2  seconds
                               Access rights : READ/WRITE
                               After images are not journalled
                               Updates are not staged
                               File is not a WORK FILE
                               Fast deletions will not be done
                               Read access to locked records is not allowed
```

Figure 12-3 FILDEF Utility Listing of SHOW Command Dialog

12.3.4 Deleting a File Definition Record

The DELETE Command

The DELETE command is invoked by typing the command keyword DELETE in response to the Command? prompt. For example:

Command? DELETE (RET)

After you have invoked the DELETE command, a single question is asked:

Logical file name?

The logical file name is the unique 1- to 6-character alphanumeric string that you assigned to an application data file. After you specify a defined logical file name in response to this question, the FILDEF utility deletes the corresponding file definition record from [1,300] tpname.FIL. The physical file previously assigned to this logical file name remains unaltered. Any executing TSTs that refer to a deleted logical file name will cause a software error to be logged.

After the deletion has been performed, the logical file name question is reissued. Should you wish to change to another command, or modify the file definitions of a different transaction processor, simply press the ESCAPE key (ESC) until you return to the desired question.

Figure 12-4 illustrates the terminal dialog used to DELETE a file definition record.

```
Command? DELETE (RET)
Logical file name? CUSTOM (RET)
Record (CUSTOM) DELETED.
Logical file name?(ESC)
Command? EXIT (RET)
>
```

Figure 12-4 FILDEF Utility Listing of DELETE Command Dialog

CHAPTER 13
APPLICATION SECURITY TOOLS

TRAX permits you to govern user access to a transaction processor through three major security tools. The WORDEF utility allows you to establish work class names that encompass a number of transaction types. The AUTDEF utility allows you to assign user authorization identifiers and passwords, and to associate them with one or more work class names. The SIGNON and SIGNOF transactions are supplied with TRAX. If you want to require users to sign on an application terminal prior to accessing a transaction processor, you can specify these transactions as part of your application. This chapter has three major sections. Each section explains one component of the TRAX security system. The sections explain:

1. The Work Class Definition Utility, WORDEF.
2. The User Authorization Definition Utility, AUTDEF.
3. The transaction, file, and station definitions required to include the SIGNON/SIGNOF transactions into an existing application.

13.1 WORK CLASSES

- An application terminal can be defined with a specific work class assignment. Using the STADEF utility (see Chapter 10), you can specify a work class name as part of the terminal station definition. The associated application terminal can then process only those transaction types grouped under the assigned work class.
- Work class names can be associated with user authorization identifiers using the AUTDEF utility. Individual users can be allowed access to one or more work classes as part of the user authorization. This permits you to restrict the set of transaction types these users can execute after signing onto an application terminal. (Assigning work classes to individual users applies only to installations that require users to sign on and off application terminals.)

13.1.1 The WORDEF Utility

Each work class has a corresponding definition record, keyed by the unique work class name, stored in the system information file tpname.WOR. These records are created and manipulated by the WORDEF utility.

13.1.1.1 Invoking the WORDEF Utility – To invoke the WORDEF utility, you issue the command line:

```
>RUN $WORDEF
```

from a support environment terminal. The utility then identifies itself, and asks you to supply the name of the transaction processor where the transactions requiring work classes are defined.

```
WORDEF V1.0
Work Class Definition Utility
Transaction Processor name?
```

After you specify the previously defined transaction processor name, the WORDEF dialog asks you to enter one of six command modes:

- ADD – Add a work class definition record to the file tpname.WOR.
- DELETE – Delete an existing work class definition record.
- EDIT – Change the list of transaction names in an existing work class definition record.
- INDEX – List the names of all defined work classes at your terminal (TI:).
- PRINT – List the transaction names included in one specific work class or in all defined work classes on the console listing device (CL0:).
- SHOW – List the transaction names included in one specific work class or in all defined work classes on your terminal (TI:).

Once you have selected a command mode, WORDEF continues its dialog. Since the questions vary according to the command specification, the remainder of this discussion of the WORDEF utility breaks into four different sections:

1. The ADD and EDIT Commands
2. The INDEX Command
3. The PRINT and SHOW Commands
4. The DELETE Command

13.1.1.2 Adding or Modifying a Work Class Definition

The ADD and EDIT Commands

If you specified ADD or EDIT in response to the Command? question, the WORDEF dialog continues by asking the question:

Work class name?

A work class name is a 1- to 6-character alphanumeric string that uniquely identifies a work class.

If you are ADDing a new work class name, you must supply a list of one or more transaction names to be included in that work class. To control the list of transaction names, WORDEF enters a loop of questions, beginning with the prompt:

Subcommand?

You must enter one of two possible subcommand keywords:

- ADD – Adds a transaction name to the list of transactions for the current work class.
- DELETE – Deletes a transaction name from the list of transactions for the current work class.

After you type the subcommand keyword ADD or DELETE, WORDEF asks you to specify the transaction name to be added to or deleted from the list of transactions for the current work class.

Transaction name <DONE>?

WORDEF continues to prompt for transaction names until you press only the RETURN key. This action accepts the DONE keyword, and tells WORDEF that you are finished with the current operation on this record. WORDEF exits from the transaction name loop, and returns you to the question "Work class name?." The transaction names that you supply to the WORDEF utility are 1- to 6-character names that represent transaction types defined in the TRADEF utility (See Chapter 11). You can specify a maximum of 64 transaction names in a work class definition.

Figure 13-1a shows a Work Class Specification Sheet completed by the application designer. The information on this sheet can be converted into work class definitions.

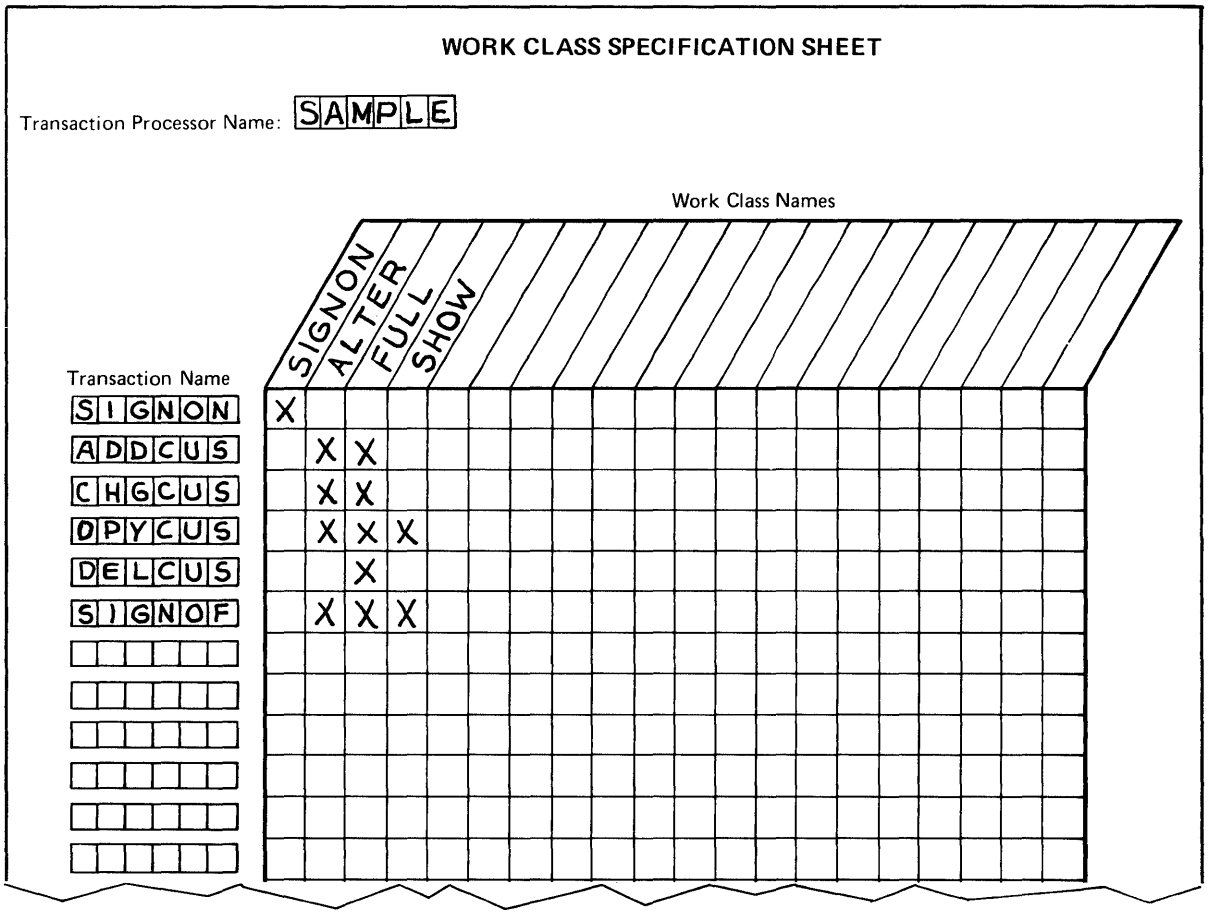


Figure 13-1a Work Class Specification Sheet for "SAMPLE"

Figure 13-1b shows the terminal dialog listing that results when the information in Figure 13-1a is submitted to the WORDEF utility.

```
>RUN $WORDEF (RET)

WORDEF  V1.0
Work Class Definition Utility
Transaction Processor name? SAMPLE (RET)

Command? ADD (RET)

Work class name? SIGNON (RET)

Sub command? ADD (RET)

Transaction name <DONE>? SIGNON (RET)

Transaction name <DONE>? (RET)

Record (SIGNON) ADDed. (RET)

Work class name? ALTER (RET)

Sub command? ADD (RET)

Transaction name <DONE>? ADDCUS (RET)

Transaction name <DONE>? CHGCUS (RET)

Transaction name <DONE>? DPYCUS (RET)

Transaction name <DONE>? SIGNOF (RET)

Transaction name <DONE>? (RET)

Record (ALTER) ADDed.

Work class name? SHOW (RET)

Sub command? ADD (RET)

Transaction name <DONE>? DPYCUS (RET)

Transaction name <DONE>? SIGNOF (RET)

Transaction name <DONE>? (RET)

Record (SHOW) ADDed.

Work class name? (ESC)

Command? EXIT (RET)
>
```

Figure 13-1b WORDEF Terminal Dialog Listing for ADD Command

13.1.1.3 Listing the INDEX of Work Class Definitions

The INDEX Command

If you answer the Command? question by typing INDEX, WORDEF prints an index of defined work class names on your terminal (TI:). Once the index listing is printed, WORDEF reissues the Command?

Figure 13-2 shows the WORDEF terminal dialog used to obtain an index of work class definitions.

```
>
>RUN $WORDEF (RET)

WORDEF  V1.0
Work Class Definition Utility


Transaction Processor name? SAMPLE (RET)

Command? INDEX (RET)

Work class name : ALTER        Number of associated transactions : 4
Work class name : FULL         Number of associated transactions : 5
Work class name : SHOW         Number of associated transactions : 2
Work class name : SIGNON       Number of associated transactions : 1
```

Figure 13-2 WORDEF Terminal Dialog Listing for INDEX Command

13.1.1.4 Printing or Displaying Work Class Definitions – If you select the PRINT or SHOW command mode, WORDEF asks you to specify the work class name that you wish to examine:

Which work class <ALL>?

If you respond to this question with a previously defined work class name, that work class is the only record displayed by the utility. If you simply type a carriage return, all work class definition records are displayed on the device specified by the command selection. In either case, WORDEF outputs a listing of the work class name and its associated transaction types. Once the listing is completed, WORDEF reissues the Command? question, allowing you to either exit, or to specify a valid command name.

Figure 13-3 shows the WORDEF terminal dialog used to display all work class definitions on your support environment terminal. The PRINT command provides identical output to the system print device CL0:.

13.1.1.5 Deleting a Work Class Definition
The DELETE Command

If you specify the DELETE command, the WORDEF utility continues its dialog by asking you to specify:

Work class name?

You respond by typing the 1- to 6-character name of the work class you want deleted. After the WORDEF utility deletes the record for that work class from the tpname.WOR file, it reissues the Work class name? question. If you want to invoke another command from this question, simply press the escape <ESC> key and the utility return to the Command? question. question.

Figure 13-4 shows the WORDEF terminal dialog used to delete a work class definition.

```
Command? SHOW. (RET)

Which work class <ALL>? (RET)

Work class name : ALTER        Number of associated transactions : 4

                               Associated transaction names

                                         ADDCUS
                                         CHGCUS
                                         DPYCUS
                                         SIGNOF

Work class name : FULL         Number of associated transactions : 5

                               Associated transaction names

                                         ADDCUS
                                         CHGCUS
                                         DPYCUS
                                         DELCUS
                                         SIGNOF

Work class name : SHOW         Number of associated transactions : 2

                               Associated transaction names

                                         DPYCUS
                                         SIGNOF

Work class name : SIGNON       Number of associated transactions : 1

                               Associated transaction names

                                         SIGNON
```

Figure 13-3 WORDEF Terminal Dialog Listing for SHOW Command

```
Command? DELETE (RET)

Work class name? SHOW (RET)

Record (SHOW) DELETED.

Work class name? (CTRL/Z)
```

Figure 13-4 WORDEF Terminal Dialog Listing for DELETE Command

13.2 USER AUTHORIZATIONS

TRAX allows you to control access to certain types of transactions by specifying a set of user authorization records that assign users access rights to certain groups of work classes. In conjunction with the SIGNON and SIGNOF transactions, user authorization provides a means to control user access to certain transaction types that might process confidential or controlled information. (See Section 13.3 for a description of how the SIGNON and SIGNOF transactions supplied with the TRAX system can be incorporated into your application.)

In many cases, the display at an application terminal will indicate that a user sign on before executing certain transactions. In order to successfully sign on as instructed, a user must have a corresponding user authorization record in the system information file tpname.AUT. A user authorization record contains a user identifier, a password, and one or more assigned work classes.

To sign onto an application terminal, a user selects the SIGNON transaction, then enters his or her unique identifer and the correct password for that identifier. The transaction types that are named in that user's assigned work classes then determine which transactions the user can execute. (When a user signs onto a terminal, his or her assigned work classes override any work class assigned to the terminal's station.)

13.2.1 The AUTDEF Utility

The AUTDEF utility creates and manipulates the records in tpname.AUT that define each user's authorization.

13.2.1.1 Invoking the AUTDEF Utility – After invoking the utility from a support environment terminal, you must supply the name of the transaction processor that will be accessed by the user or users whose authorization records are being added, deleted, or edited. The transaction processor whose name is supplied in response to the prompt:

Transaction Processor name?

must have been previously defined by running the TPDEF utility. (See Chapter 4 for further information on TPDEF.) The AUTDEF utility then asks you to select one of seven command modes by issuing the prompt:

Command?

- ADD – Add user authorization records to the file tpname.AUT.
- DELETE – Delete an existing user authorization record.
- EDIT – Modify the information in an existing user authorization record.
- INDEX – List at your terminal (TI:) the user identifiers and passwords of all authorized users.
- PRINT – List the information in one or all existing user authorization records on the system console listing device (CL:).
- SHOW – List at your terminal (TI:) the information in one or all existing user authorization records.

13.2.1.2 Adding or Editing a User Authorization Definition

The ADD and EDIT Commands

To ADD or EDIT a user authorization record, you must supply a unique user identifier, a password, and one or more work class names.

User identifer?

A User identifier is a unique 1- to 20-character alphanumeric string. This identifier cannot be duplicated in any other record within the file tpname.AUT; the identifier is the key to the record.

Password?

The password is a 1- to 6-character alphanumeric string. Each user must enter the password associated with his or her identifier in order to sign on successfully.

After you have supplied or examined the user identifier and password, the AUTDEF utility asks you to supply the one or more work classes that the user can access. The names you specify should correspond to work class names that have been defined by the WORDEF Utility.

Unless you specify one or more work class names, the user cannot execute any transactions after signing on.

To control the list of work class names, AUTDEF enters a loop of questions, starting with the prompt:

Subcommand?

You then reply with one of two subcommands:

- ADD – Assign a work class to the user's authorization record.
- DELETE – Delete a work class from the user's authorization record.

When you have specified either ADD or DELETE, AUTDEF asks you to name a work class. You respond by typing a 1- to 6-character work class name. (The default answer is DONE; i.e. you have defined all work classes for this application.) AUTDEF continues to prompt you for work class names until you respond by pressing the RETURN key.

Work class name <DONE>?

When you are DONE with the list of work class names, AUTDEF then returns to the question "User identifier?"

If you are EDITing a user authorization record, you must specify either ADD or DELETE in response to the "Subcommand?" prompt. (If you are not changing any work classes, simply enter ADD, and then hit a carriage return in response to the "Work Class <DONE>? prompt.) If you wish to change work classes, you must DELETE incorrect names, and then ADD the desired work class names.

Figure 13-5a shows a user authorization specification sheet that is implemented by the AUTDEF utility.

Figure 13-5b shows the AUTDEF terminal dialog required to implement the specifications shown in Figure 13-5a.

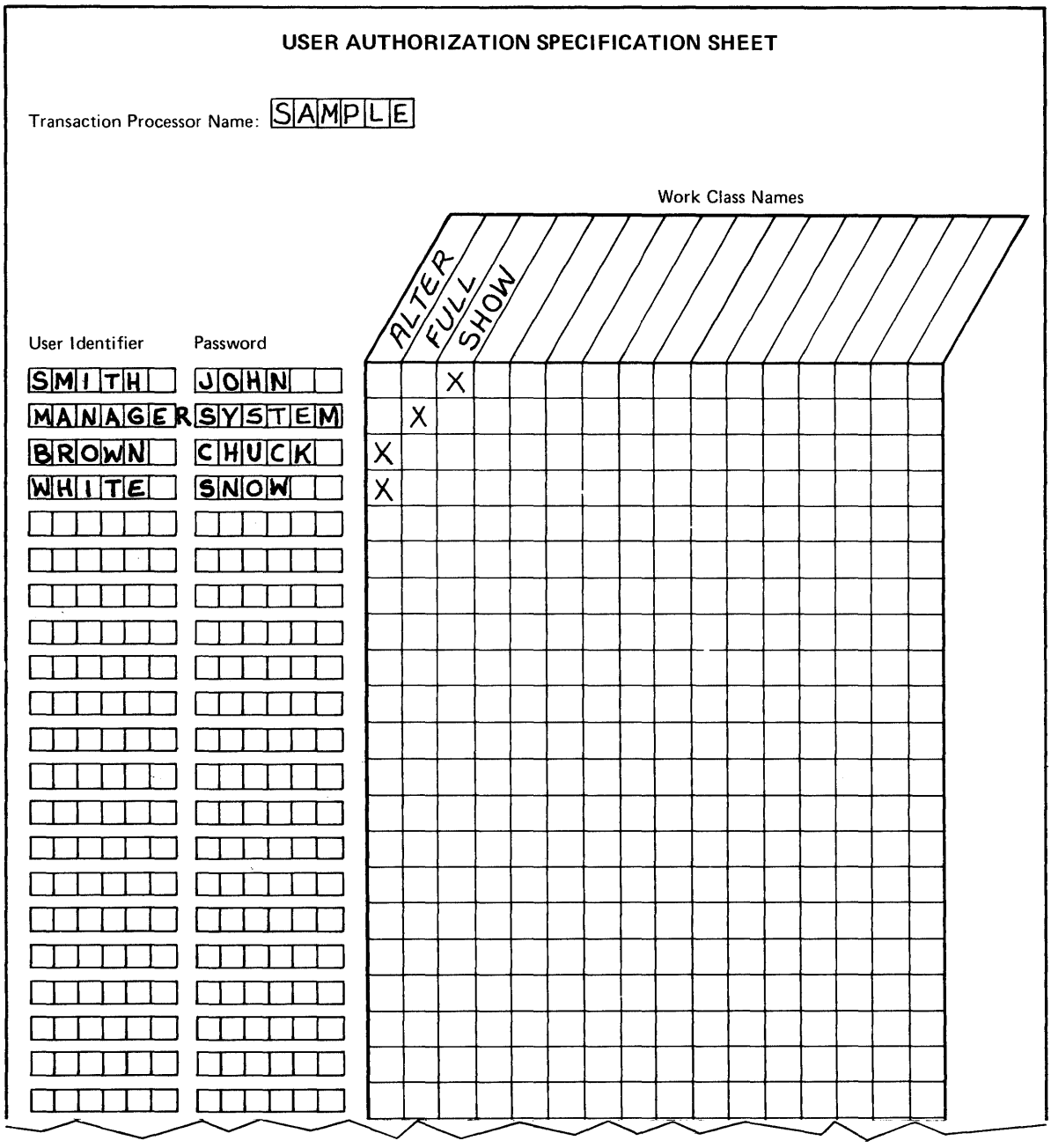


Figure 13-5a User Authorization Specification Sheet for "SAMPLE"

```
>
>RUN $AUTDEF (RET)

AUTDEF  V1.0
User Authorization Definition Utility

Transaction Processor name? SAMPLE (RET)

Command? ADD (RET)

User identifier? SMITH (RET)

Password? JOHN (RET)

Sub command? ADD (RET)

Work class name <DONE>? SHOW (RET)

Work class name <DONE>? (RET)

Record (SMITH) ADDed.

User identifier? MANAGER (RET)

Password? SYSTEM (RET)

Sub command? ADD (RET)

Work class name <DONE>? FULL (RET)

Work class name <DONE>? (RET)

Record (MANAGER) ADDed.

User identifier? BROWN (RET)

Password? CHUCK (RET)

Sub command? ADD (RET)

Work class name <DONE>? ALTER (RET)

Work class name <DONE>? (RET)

Record (BROWN) ADDed.
```

Figure 13-5b AUTDEF Terminal Dialog Listing for ADD Command

13.2.1.3 Listing the INDEX of User Authorizations

The INDEX Command

This command requires no further terminal dialog. It displays the identifiers and passwords of all user authorizations defined in the .AUT file associated with the transaction processor name that you supplied when you invoked the utility. This listing appears on the support environment terminal where you are currently running the AUTDEF utility.

Figure 13-6 lists the AUTDEF terminal dialog that lists the INDEX of all user authorization definitions currently in the user authorization definition file [1,300] SAMPLE.AUT.

```
Command? INDEX (RET)

User identifier : BROWN                    Password : CHUCK
User identifier : MANAGER                  Password : SYSTEM
User identifier : SMITH                    Password : JOHN
User identifier : WHITE                    Password : SNOW
```

Figure 13-6 AUTDEF Utility Listing of INDEX Command Terminal Dialog

13.2.1.4 Printing or Showing a User Authorization Definition

The PRINT Command
The SHOW Command

The SHOW and PRINT commands perform the same operation but direct their output to different devices. SHOW lists the attributes of a user authorization definition at your support environment terminal, while PRINT provides a hard-copy listing on the system device assigned to logical unit CL0:. (This is usually the line printer or the LA-36 console terminal.) The terminal dialog for these commands consists of a single prompt:

Which user authorization <ALL>?

If you provide a user authorization name in response to the prompt, the SHOW and PRINT Commands will display information for the user authorization only. Responding to this prompt by pressing the RETURN key <RET> produces a complete listing of the user authorization attributes for the transaction processor name that you specified when you invoked the AUTDEF utility.

Figure 13-7 lists the AUTDEF terminal dialog that lists all user authorization definitions currently in the user authorization definition file [1,300] SAMPLE.AUT on your terminal.

```
Command? SHOW (RET)
Which user identifier <ALL>? BROWN (RET)
User identifier : BROWN                    Password : CHUCK
                              Number of associated work classes : 1
                              Associated work class names
                                        ALTER
```

Figure 13-7 AUTDEF Utility Listing of PRINT Command Terminal Dialog

13.2.1.5 Deleting a User Authorization Definition

The DELETE Command

The Delete command allows you to remove a user authorization definition from the .AUT file named by the transaction processor specified when you invoked AUTDEF. To delete a user authorization definition, simply type the identifier you wish to delete.

Figure 13-8 lists the AUTDEF terminal dialog used to delete the user authorization "SMITH" from the authorization definition file [1,300] SAMPLE.AUT.

```
Command? DELETE (RET)
User identifier? BROWN (RET)
Record (BROWN) DELETED.
User identifier?
```

Figure 13-8 AUTDEF Utility Listing of DELETE Command Terminal Dialog

13.2.2 The SIGNON and SIGNOF Transactions

The SIGNON and SIGNOF transactions consist of forms and TSTs supplied as TRAX system software. When the forms and TSTs are incorporated into a transaction processor using the definition utilities, the SIGNON and SIGNOF transactions are created. These transactions allow you to establish authorization procedures for application terminal users.

13.2.2.1 Incorporating the SIGNON/SIGNOF Transaction Into Your Application – If the application designer selects SIGNON/SIGNOF control for a transaction processor, the transaction processor definition utilities must first be run to incorporate the definitions, TSTs, forms and files of the SIGNON and SIGNOF transaction. The steps in this process are:

1. Run the TPDEF utility to EDIT the transaction processor definition. The EDIT should add the following resources to the existing definition:

   a. Two transaction types
   b. Two user TSTs
   c. Two application data files

   Figure 13-9 shows the modifications needed to include the SIGNON facility in a transaction processor definition.

TRANSACTION PROCESSOR SPECIFICATION SHEET

Transaction Processor Definition: SAMPLE	Edit	
Transaction Processor Name: SAMPLE		
Maximum number of transaction types:	(0–64)	□□+2
Maximum number of concurrent transaction instances:	(0–96)	□□
Maximum number of application terminals:	(0–256)	□□□(SAME)
Maximum number of user TSTs	(0–256)	□□□+2
Maximum number of master link stations:	(0–10)	□□(SAME)
Maximum number of slave link stations:	(0–64)	□□(SAME)
Maximum size of receive link message:	(0–4096)	□□□□
Maximum number of submit batch stations:	(0–1)	□(SAME)
Maximum number of slave batch stations:	(0–16)	□□□(SAME)
Maximum number of mailbox stations:	(0–10)	□□□(SAME)
Maximum number of application data files:	(0–64)	□□□ +2
Maximum transaction slot size:	(1–8192)	□□□□ blocks
Automatic crash recovery:	□ – YES	□ – NO SAME

Figure 13-9 Transaction Processor Specification Sheet

2. Run the TRADEF utility to ADD the SIGNON and SIGNOF transactions. Figure 13-10a and 13-10b list the transaction definitions for SIGNON and SIGNOF respectively.

TRANSACTION SPECIFICATION SHEET

Transaction Processor Name: SAMPLE

Transaction Name: SIGNOF

Exchange Recovery? [ ] – YES [X] – NO

Log Exchange Messages? [ ] – YES [X] – NO

Log Other Station Messages? [ ] – YES [X] – NO

Maximum Size of Exchange Message: 64 bytes

Transaction Workspace Size: ____ bytes

System Workspace Size
(Calculate according to formula on worksheet – "Calculating the system workspace".) ____ (64-byte blocks)

Transaction Slot Size Calculation:

Divide Exchange Message Size by 64 and round up: 1 blocks

Divide Transaction Workspace Size by 64 and round up: Ø blocks

Enter System Workspace Size: Ø blocks

Add to find Transaction Slot Size: 1 blocks

NOTE: A Transaction Exchange Definition should be prepared for each exchange associated with the transaction you have just defined.

TRANSACTION EXCHANGE DEFINITIONS

Exchange Label	Form Name	Destination Station List	Wait	Repeat	Subsequent Action	Time Limit
SIGNOF	SIGNOF	SIGNOF	[X] WAIT	[ ] REPEAT	[X] INITIAL	[2] MINS
			[ ] NOWAIT	[X] NOREPEAT	[ ] FIRST	
					[ ] NEXT	

Figure 13-10a Transaction Specification for "SIGNON"

TRANSACTION SPECIFICATION SHEET

Transaction Processor Name: SAMPLE

Transaction Name: SIGNON

Exchange Recovery? [ ] – YES [X] – NO

Log Exchange Messages? [ ] – YES [X] – NO

Log Other Station Messages? [ ] – YES [X] – NO

Maximum Size of Exchange Message: 64 bytes

Transaction Workspace Size: ____ bytes

System Workspace Size
(Calculate according to formula on worksheet – "Calculating the system workspace".) ____ (64-byte blocks)

Transaction Slot Size Calculation:

Divide Exchange Message Size by 64 and round up:	1 blocks
Divide Transaction Workspace Size by 64 and round up:	Ø blocks
Enter System Workspace Size:	Ø blocks
Add to find Transaction Slot Size:	1 blocks

NOTE: A Transaction Exchange Definition should be prepared for each exchange associated with the transaction you have just defined.

TRANSACTION EXCHANGE DEFINITIONS

Exchange Label	Form Name	Destination Station List	Wait	Repeat	Subsequent Action	Time Limit
SIGNON	SIGNON	SIGNON	[X] WAIT	[ ] REPEAT	[X] INITIAL	1 MINS
			[ ] NOWAIT	[X] NOREPEAT	[ ] FIRST	
					[ ] NEXT	
			[ ] WAIT	[ ] REPEAT	[ ] INITIAL	MINS
			[ ] NOWAIT	[ ] NOREPEAT	[ ] FIRST	
					[ ] NEXT	

Figure 13-10b Transaction Specification for "SIGNOF"

3. Run the STADEF utility to:

   a. Add the SIGNON and SIGNOF TSTs. The TST task images reside in UFD [1,300]. The station definitions for these TSTs are shown in Figure 13-11a.

TST STATION SPECIFICATION SHEET

Transaction Processor Name: SAMPLE

Station Name	Station Priority	Task Image File Specification	No. Active Copies	Serially Reusable?
SIGNON	128	SYØ:[1,3ØØ]SIGNON.TSK;Ø	1	[X] – YES [ ] – NO
SIGNOF	128	SYØ:[1,3ØØ]SIGNOF.TSK;Ø	1	[X] – YES [ ] – NO

Figure 13-11a TST station Specification for "SIGNON" and "SIGNOF".

   b. Edit all interactive terminal station definitions to specify an initial a work class of "SIGNON". These changes are shown in Figure 13-11b.

TERMINAL STATION SPECIFICATION SHEET

Transaction Processor Name: SAMPLE

Station Name	Device Name	Device Type	System Messages	Work Class	Run A Dedicated Transaction?
TERM**	ØØ1V62 -ØØ4V62	[X] – BOTH [ ] – OUTPUT	[X] – YES [ ] – NO	SIGNON	[ ] Yes – Transaction Name: [X] No – Initial Form Name: SELECT
		[ ] – BOTH [ ] – OUTPUT	[ ] – YES [ ] – NO		[ ] Yes – Transaction Name: [ ] No – Initial Form Name:
		[ ] – BOTH [ ] – OUTPUT	[ ] – YES [ ] – NO		[ ] Yes – Transaction Name: [ ] No – Initial Form Name:
		[ ] – BOTH [ ] – OUTPUT	[ ] – YES [ ] – NO		[ ] Yes – Transaction Name: [ ] No – Initial Form Name:
		[ ] – BOTH [ ] – OUTPUT	[ ] – YES [ ] – NO		[ ] Yes – Transaction Name: [ ] No – Initial Form Name:

Figure 13-11b Terminal Station Specification with SIGNON Work Class

4. Run the FILDEF utility to ADD the logical files AUTFIL and WORFIL. These file definitions are described in Figures 13-12a and 13-12b respectively. These files are the user authorization and work class definition files, that are accessed by the SIGNON and SIGNOF transactions.

FILE DEFINITION

Part One

Transaction Processor Name: SAMPLE

Logical Filename: WORFIL

RMS File Specification: SY0:[1.300]SAMPLE.WOR;0

Work File?
- ☐ – Yes (Go to Part Two)
- ☒ – No (Continue with next question)

Is This an Indexed File?
- ☒ – Yes: No. of Keys 1
  Maximum Key Length 6
- ☐ – No: Sequential or Relative File

Maximum Concurrent File Accesses? 1

Read-Only?
- ☒ – Yes
- ☐ – No

Fast Deletions?
- ☒ – Yes
- ☐ – No

Lock Interval 1 seconds

Read Access to Locked Records?
- ☒ – Yes
- ☐ – No

Journal?
- ☐ – Yes (Go to Part Two)
- ☒ – No (Continue with next question)

Staged File Updates?
- ☐ – Yes
- ☒ – No

Part Two

File Channel Assignment

Description of File Contents: WORK CLASS DATA

Assigned I/O Channel Number ☒ N/A

Figure 13-12a Work Class File Definition Specification.

FILE DEFINITION

Part One

Transaction Processor Name: SAMPLE

Logical Filename: AUTFIL

RMS File Specification: SY0:[1.300]SAMPLE.AUT; 0

Work File?
- ☐ – Yes (Go to Part Two)
- ☒ – No (Continue with next question)

Is This an Indexed File?
- ☒ – Yes: No. of Keys 1
  Maximum Key Length 20
- ☐ – No: Sequential or Relative File

Maximum Concurrent File Accesses? 1

Read-Only?
- ☒ – Yes
- ☐ – No

Fast Deletions?
- ☒ – Yes
- ☐ – No

Lock Interval 1 seconds

Read Access to Locked Records?
- ☒ – Yes
- ☐ – No

Journal?
- ☐ – Yes (Go to Part Two)
- ☒ – No (Continue with next question)

Staged File Updates?
- ☐ – Yes
- ☒ – No

Part Two

File Channel Assignment

Description of File Contents: USER AUTHORIZATION DATA

Assigned I/O Channel Number ☒ N/A

Figure 13-12b User Authorization File Definition Specification.

5. Run the WORDEF utility to ADD a work class called SIGNON, and to EDIT all other work class definitions to ADD the SIGNOF transaction. The SIGNON work class must contain the SIGNON transaction. All other work classes must contain the SIGNOF transaction. These transactions are already included in the description of WORDEF in Section 13.1.
6. Run the AUTDEF utility to establish the user identification, passwords, and permitted work classes for each authorized user. The User Authorization description in Section 13.2 shows this procedure.

The next three steps include the SIGNON/SIGNOF facility in the forms defintion file. Details on how to code forms and the ATL Utility program are given in the *TRAX ATL Language Reference Manual.*

1. Edit the transaction selection form to include the SIGNON and SIGNOF transactions in the menu or screen display.
2. Run the ATL utility to REPLACE the transaction selection form.
3. Run the ATL utility to ADD the form defnitions for SIGNON and SIGNOF. The source statement files for these forms are [1,300] SIGNON.ATL, and [1,300] SIGNOF.ATL. Figure 13-13 shows the ATL utility dialog required to ADD the SIGNON and SIGNOF forms.

```
>RUN $ATL (RET)

ATL      V1.0
TRAX Forms Definition File Utility


Transaction processor name <"">? SAMPLF (RET)

Command <COMPILE>? ADD (RET)

Form name? SIGNON (RET)

ATL source file <SIGNON.ATL>? [1,300]SIGNON (RET)

Device type <VT62>? (RET)

List <ALL>? NONE (RET)


Command <COMPILE>? ADD (RET)

Form name? SIGNOF (RET)

ATL source file <SIGNOF.ATL>? [1,300]SIGNOF (RET)

Device type <VT62>? (RET)

List <ALL>? NONE (RET)


Command <COMPILE>? EX (RET)
```

Figure 13-13 ATL Utility Dialog to Add SIGNON and SIGNOF forms

13.2.2.2 Using the SIGNON and SIGNOF transactions – The user interface for signing on an application terminal consists of the following steps.

1. When the transaction processor is started, the user selects SIGNON from the menu of transaction types and presses the ENTER key.
2. The SIGNON form is displayed on the application terminal (See Figure 13-14a).

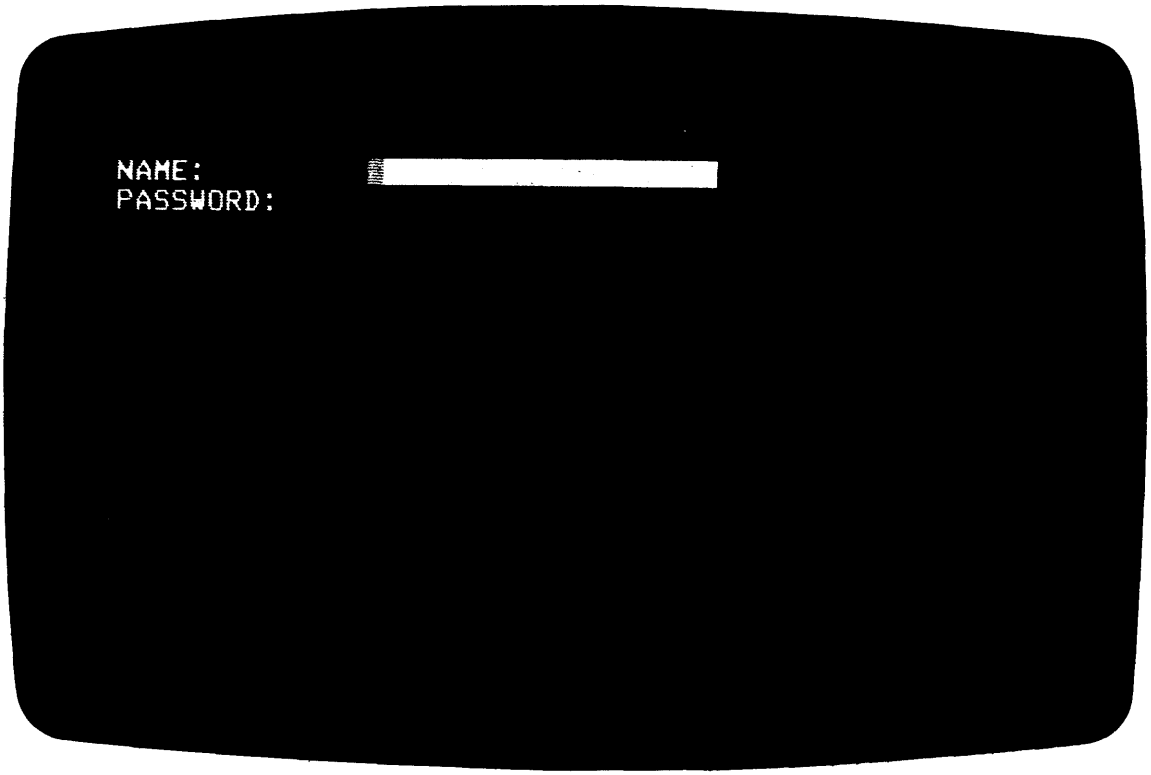


Figure 13-14a Initial Display of SIGNON Form

3. The user types in his user identification and password, and presses ENTER. (See Figure 13-14b.)

Figure 13-14b SIGNON Form after ID and password are typed.

4. The SIGNON TST executes. If the user ID or password is incorrect, a message is sent to the SIGNON form, and the user is allowed to reenter the user ID and password.
5. If the User ID and password are valid, the transaction selection form is redisplayed, with the work classes in the user authorization record now enabled for that terminal.

To sign off from an application terminal, the user must:

1. Return the terminal to the transaction selection form.
2. Select the SIGNOF transaction from the menu, and press ENTER.
3. Press ENTER is response to the SIGNOF form. (See Figure 13-15.)
4. The SIGNOF transaction disables all work classes except SIGNON, and redisplays the transaction selection form.

Figure 13-15 Initial Display of SIGNOF Form

CHAPTER 14
TRANSACTION PROCESSOR TESTING ENVIRONMENT

14.1 INSTALLING AND TESTING A TRANSACTION PROCESSOR

TRAX provides you with a sophisticated transaction testing capability that assists you during the process of debugging and testing the components of a transaction processor. The three major facilities available to you for debugging purposes are:

- A transaction processor trace facility, that records data from each transaction instance. This data can be submitted to the TPTRAC utility program, and a detailed trace of each transaction instance is produced in listing form.
- A support environment terminal that you can include in the TST task image. Using system debugging aids such as the BASIC-PLUS-2 debugger, and language terminal I/O facilities such as the COBOL DISPLAY statement, you can control the execution of the TST from the debugging terminal. (Building TSTs in DEBUG mode and including debugging terminals in a TST task image is discussed in the TSTBLD utility description in Section 7. TSTBLD.)
- A software error logger (SERLOG) that records all system and user software errors as they occur. Two utility programs, SERANL and SERDAY interpret the errors and provide valuable information that can be used to debug and correct these errors. (SERANL and SERDAY are dicusssed in the *TRAX System Manager's Guide.*)

The following sections discuss how to

- Debug transactions in the transaction processing environment.
- Enable the software error logging facility for your transaction processor.
- Control the execution of a transaction processor using the TPCTRL utility.
- Interpret the secondary error log listing.
- Run the TPTRAC utility to generate trace listings of your transaction processor.

14.2 DEBUGGING IN THE TRANSACTION PROCESSING ENVIRONMENT

Before you attempt to debug a transaction operating under a transaction processor, you should have performed the following actions:

1. Debugged all TSTs using the DEBUG utility. (See Section 7.3.)
2. Rebuilt all TST task images using TSTBLD. If you want a transaction processing debug terminal linked to the TST, specify this support in the TSTBLD dialog. (See Section 7.2.)

The remainder of this Chapter describes the steps you should follow when you debug in the transaction processing environment.

14.3 THE SOFTWARE ERROR LOGGING TASK - SERLOG

Trax provides a software error logging program that runs concurrently with transaction processors. When you are debugging in the transaction processing environment, you should always enable

the error logger. Software errors detected in TRAX system and application software modules are sent to SERLOG. SERLOG logs errors from selected transaction processors to the system error log, [1,300] SERLOG.LOG. As an option, you can also direct SERLOG to format and display error messages on a support environment terminal.

The error logging process is controlled through the SERCTL utility. If you are testing a new transaction processor for the first time, you must run SERCTL to enable error logging for the transaction processor.

The SERCTL utility is described in the *TRAX System Manager's Guide.* Figure 14-1 shows a terminal listing of the SERCTL utility dialog that enables SERLOG and directs the output to a support environment terminal TT5:.

```
>RUN $SERCTL (RET)

SERCTL  V1.0
Software error logger control utility


Command <EXIT>? TERMINAL (RET)

Device name <CO0:>? TT1: (RET)

The new  secondary logging  device will be  in effect the  next time the error
logger is run.

Command <EXIT>? SELECT (RET)

Transaction processor name <ALL>? SAMPLE (RET)

Select error monitoring <YES>? (RET)

Transaction processor name <DONE>? (RET)

The new selections will be applied.

Command <EXIT>? (RET)
>
```

Figure 14-1 SERCTL Utility Dialog to Enable Error Logging

14.4 USING THE TPCTRL UTILITY

You use the TPCTRL utility to install, start, stop, and remove a transaction processor. The examples shown here are designed to assist the application programmer who is integrating and testing a transaction processor. A complete description of the TPCTRL utility is given in the *TRAX System Manager's Guide.*

Before you install a transaction processor, you must insure that a TRAX system capable of supporting a transaction processor is installed on the computer you are using. The system manager can do this for you, or can give you instructions on how to do this. For further information refer to the SETUP and BOOT utility descriptions in the *TRAX System Manager's Guide.*

TPCTRL is run under a privileged UIC from a support environment terminal. Figure 14-2 shows the terminal dialog required to install the transaction processor "SAMPLE" and enable the trace facility.

```
>
>RUN $TPCTRL

TPCTRL V1.0
Transaction Processor Control Utility

Command <BRIEF>? INSTALL (RET)

Transaction processor name ? SAMPLE (RET)

Partition <TP1PAR>? (RET)

Trace transaction processor <NO>? YES (RET)

Write protect data base <NO>? (RET)

High-performance RMS <NO>? (RET)

Forms definition file version <LATEST>? (RET)
```

Figure 14-2 TPCTRL Terminal Dialog for the INSTALL Command

After the INSTALL command dialog is completed, TPCTRL

- Allocates the resources required by the transaction processor
- Accesses or creates the files needed for the transaction processor to run
- Builds system tables based on the data in the transaction processor definition files
- Performs consistency checks

After a transaction processor is installed, it must be started. When you issue the START command to TPCTRL, it starts an installed transaction processor and sets the start status in the transaction processor record of the TPDEF.TPF file.

If the INSTALL command was successful, you enter the START command dialog by pressing the RETURN key in response to the "Command?" question.

Figure 14-3 shows the START command dialog for "SAMPLE".

```
Command <START>? (RET)
Transaction processor name <SAMPLE>? (RET)
Enable Journalling <NO>? (RET)
Enable logging <NO>? (RET)
Command <EXIT>? (RET)
>
```

Figure 14-3 TPCTRL Terminal Dialog to START "SAMPLE"

14.5 TESTING THE TRANSACTION

Once the transaction processor is installed and started, you can begin testing. Three facilities are available to record the results of testing.

1. The transaction trace facility.
2. A debugging terminal linked to a TST.
3. The software error log.

The three facilities are independent of each other.

When the transaction processor is started, all application terminals have the initial screen displayed.

You begin by invoking a transaction from an application terminal. If you have included COBOL ACCEPT statements or the BASIC-PLUS-2 debugger in a TST, and have linked that TST to a debugging terminal, the transaction execution stops when these statements are reached. You then examine the debugging terminal (which is a support terminal, not a VT62), and interact with the executing TST. Any software errors that occur during your transaction execution written to the secondary error log.

If you are tracing a transaction processor, remember that the trace facility generates large volumes of data. Limiting debugging sessions to short periods keeps individual TPTRAC output to a manageable size.

14.6 USING THE SECONDARY ERROR LOG LISTINGS

The secondary error log is very useful for programmers during the application debugging process. When a software error is detected, SERLOG prints a message in the following form on the enable secondary logging device:

1. The date and time the error was logged.
2. The transaction name, transaction instance number, and source station ID.
3. The error message.

Figure 14-4 shows typical SERLOG output to a support environment terminal. If you want more detailed information, error log analysis is performed by two utility programs: SERDAY and SERANL.

```
 *** Software error recorded at 22-AUG-78  16:36:03
Processor SAMPLE    Source Station TERM11    Transaction ENTORD    ID 305
Abnormal tst termination

*** Software error recorded at 22-AUG-78  16:36:35
Processor SAMPLE    Source Station TERM10    Transaction ENTORD    ID 49
No modifier flag in message

*** Software error recorded at 22-AUG-78  16:36:59
Processor SAMPLE    Source Station ******    Transaction ENTORD    ID 306
Unstage error

*** Software error recorded at 22-AUG-78  16:45:56
Processor SAMPLE    Source Station TERM14    Transaction CHNSTK    ID 1513
Exchange timeout

*** Software error recorded at 22-AUG-78  17:37:04
Processor SAMPLE    Source Station TERM8     Transaction ENTORD    ID 112
Reply message after transaction closed
```

Figure 14-4 SERLOG Output Listing

The *TRAX System Manager's Guide* gives detailed information about how to use SERANL, and SERDAY.

14.7 STOPPING A TRANSACTION PROCESSOR

When you want to stop a transaction processor, you must invoke the TPCTRL utility from a privileged UIC. The STOP command stops a transaction processor.

Figure 14-5 shows the STOP and REMOVE command dialog for "SAMPLE".

```
>RUN $TPCTRL (RET)

TPCTRL V1.0
Transaction Processor Control Utility

Command <BRIEF>? STOP (RET)

Transaction processor name <ALL>? (RET)

Minutes until transaction initiation is disabled <0>? (RET)

Minutes until incomplete processing is aborted <10>? 0 (RET)

Command <BRIEF>? REMOVE (RET)

Transaction processor name <ALL>? (RET)

Command <EXIT>? (RET)
```

Figure 14-5 TPCTRL Terminal Dialog to STOP and REMOVE "SAMPLE"

After you stop a transaction processor, the REMOVE command releases the system resources used by the stopped transaction processor. The REMOVE command also resets the install status in the transaction processor record of the TPDEF.TPF file.

14.8 TRANSACTION PROCESSOR TRACE UTILITY - TPTRAC

The Transaction Processor Trace Utility, TPTRAC, allows you to decode the trace file created by an active transaction processor. When a TP is installed and the parameter TRACE is specified as part of the TPCTRL utility dialog, that TP's actions are recorded in a trace file [1,10] tpname.TRC.

When a TP is stopped, the trace file is closed. In order to interpret the data in this trace file, you must process the file through the TPTRAC utility. TPTRAC creates a detailed listing of events that occurred when the TP was active.

TPTRAC follows the standard TRAX utility dialog conventions described in Chapter 8.

In the dialog that controls the operation of TPTRAC, you may select listings of specific transaction types, TSTs, and exchange labels.

The listings may include any combination of input messages, output messages and other system calls.

After a TP has been stopped, the trace file closed, and a full support environment system installed, you can invoke TPTRAC from a support environment terminal by typing:

RUN $TPTRAC

TPTRAC identifies itself, and begins a dialog to determine the types of trace listings you require. The first question asks:

Input file?

You respond with the file specification of the trace file. Unless you specify these elements specifically, TPTRAC assumes that the file resides on the system device under UFD [1,10], with a file type of .TRC, and the current version number. You must always specify the file name in response to this question. In general, this will be the transaction processor name.

Listing device <CL:>

The next question asks you to specify the device and/or file where the listing files produced by TPTRAC should be sent. If you answer this question by pressing RETURN <RET>, the default output device is the system console listing device (CL0:). This is usually equated to the line printer when your system is generated.

If you wish to specify another device or file, simply specify a valid output device or file specification. TPTRAC assumes your UFD and a file type of .LST.

Once the input and output files have been specified, TPTRAC begins a series of questions to determine the types of operations to examine. The first of these questions asks you to specify:

Transaction name <ALL>?

If you press the RETURN key, all transaction types are listed. If you respond with a 1- to 6-character transaction name, then only instances of that transaction are examined. This question is repeated until only the RETURN key is pressed. The first time, the default will be ALL. Otherwise, the default is DONE.

TST station name <ALL>?

This question asks you to specify which TSTs are to be listed. If you press the RETURN key, I/O operations and system calls are listed for every TST invoked by the transaction processor.

If you respond with a 1- to 6-character TST station name, then only operations for that TST are listed. This question is to be repeated until only the RETURN key is pressed. The first time, the default will be ALL. Otherwise, the default is DONE.

TPTRAC enters a second set of four questions that ask you to specify the types of system calls and station messages that you want to include in the listing. The first of these asks:

Exchange messages <YES>?

If you want the listing to include exchange message data, respond by pressing the RETURN key. If you don't want to see the exchange message data, type N or NO followed by the RETURN key.

Other station messages <YES>

If you want the listing to include response messages, report messages, and mailbox messages, respond by pressing the RETURN key. If you don't want the listing to include these messages, type N or NO followed by the RETURN key.

Transaction workspace <YES>?

If you want the listing to include the transaction workspace data, respond by pressing the RETURN key. If you don't want the listings to include the workspace data, type N or NO followed by the RETURN key.

System calls <YES>

If you want the listing to include data concerning calls (other than message calls) to the TRAX system library, respond by pressing the RETURN key. If you don't want the listing to include call information, type N or NO followed by the RETURN key.

After this response is accepted TPTRAC produces the requested listing and exits.

Figure 14-6 illustrates the dialog used to produce the listings in Figure 14-7.

```
>RUN $TPTRAC (RET)

TPTRAC  V1.0
Transaction processor trace utility


Input file? SAMPLE.TRC (RET)

Listing device <CL:>? LP: (RET)

Transaction <ALL>? (RET)

TST station <ALL>? (RET)

Exchange messages <YES>? (RET)

Other station messages <YES>? (RET)

Transaction workspace <YES>? (RET)

System calls <YES>? (RET)
>
```

Figure 14-6 TPTRAC Dialog for SAMPLE

14.9 ANNOTATED TPTRAC OUTPUT LISTING

Figure 14-7 consists of a set of annotated TPTRAC listings of an instance of the DPYCUS transaction. The numbers on the computer output listing refer to the numbered paragraphs in the annotation section.

```
Transaction Trace Listing   07-Aug-78  10:21 (1)

Input file: [1,10]SAMPLE.TRC (2)

Transactions     ALL (3)

TST stations     ALL (4)

Trace will include:     Exchange messages
                        Other station messages (5)
                        Transaction workspace
                        System calls
```

Figure 14-7 TPTRAC Annotated Output Listing

1. This line identifies the listing, and the time and date it was produced.
2. [1,10] SAMPLE.TRC is the trace input file produced by the transaction processor SAMPLE.
3. The trace listing includes all transaction types.
4. The trace listing includes all TST stations.
5. The trace listings include these data structures and system calls.

```
Transaction trace from file [1,10]SAMPLE.TRC (1)

****************  Start of TST (2)

Transaction     TST             Sequence          Status wds    Name
DPYCUS (3)      FIND (4)        1    (5)

Message:
                          (6)
          ---------+---------+---------+---------+---------+
 0        P00000N K Jantakian                     mk[6mk[6mk[6mk
 50       [6mk[6mk[6mk[6
          ---------+---------+---------+---------+---------+

workspace:
                          (7)
          ---------+---------+---------+---------+---------+
 0        mk[6mk[6mk[6mk[6mk[6mk[6mk[6mk[6mk[6mk[6mk[6mk[6mk
 50       [6mk[6mk[6mk[6mk[6mk[6mk[6mk[6mk[6mk[6mk[6mk[6mk[6
 100      mk[6mk[6mk[6mk[6mk[6mk[6mk[6mk[6mk[6mk[6mk[6mk[6mk
 150      [6mk[6mk[6mk[6mk[6mk[6mk[6mk[6mk[6mk[6mk[6mk[6mk[6
 200      mk[6mk[6mk[6mk[6mk[6mk[6mk[6mk[6mk[6mk[6mk[6mk[6mk
 250      [6mk[6     mk[6mk[6mk[6mk[6mk[6mk[6mk[6mk[6mk[6mk[6
 300      mk[6mk[6mk[6mk[6mk[6mk[6mk[6mk[6mk[6mk[6mk[6mk[6mk
 350      [6mk[6mk[6mk[6mk[6mk[6mk[6mk[6mk[6
          ---------+---------+---------+---------+---------+
```

Figure 14-7 (cont.) TPTRAC Annotated Output Listing

1. Identifying line for trace output.
2. The first TST in the first exchange starts here.
3. The name of the transaction type being run.
4. The TST station name.
5. The transaction instance number defined by the system.
6. The contents of the exchange message at the start of the transaction instance. The area displayed is the maximum number of bytes defined for this transaction instance.
7. The contents of the transaction workspace at the start of the transaction instance. The area displayed is the maximum number of bytes defined for this transaction type.

```
                                        ❶
0       -5267 -18725  -5267 -18725  -5267 -18725  -5267 -18725  -5267 -18725
10      -5267 -18725  -5267 -18725  -5267 -18725  -5267 -18725  -5267 -18725
20      -5267 -18725  -5267 -18725  -5267 -18725  -5267 -18725  -5267 -18725

170     -5267 -18725  -5267 -18725  -5267 -18725  -5267 -18725  -5267 -18725
180     -5267 -18725  -5267 -18725  -5267 -18725  -5267 -18725  -5267 -18725
190     -5267 -18725

**************  Open file ❷

Transaction     IST           Sequence        Status Wds    Name
DPYCUS          FIND          1               1      1750   CUSTOM
                                                ❸            ❹

File organization is indexed ❺
Record format is variable
Maximum record size = 205
Bucket size = 1
Block size = 512

**************  Connect stream ❻

Transaction     IST           Sequence        Status Wds    Name
DPYCUS          FIND          1               1 ❼    0

Channel number = 3 ❽
```

Figure 14-7 (cont.) TPTRAC Annotated Output Listing

1. An Integer dump of the transaction workspace data.
2. The name of the operation performed by the TST.
3. The status words returned from the open file operation. The value "1" indicates success. The value "1750" is additional information provided by RMS.
4. The logical file name being opened.
5. Information about the file just opened.
6. The next operation is a connect to an I/O channel.
7. The status is success.
8. Channel #3 was connected.

```
**************  Find a record ❶

Transaction     IST           Sequence        Status was   Name
DPYCUS          FIND          1               1        Ø
                                                  ❷

Channel number = 3
Access mode is random
Reference key (if applicable) = 1 ❸
Record number (if applicable) = Ø


Key = N K Jantakian ❹



**************  Get a record ❺

Transaction     IST           Sequence        Status was   Name
DPYCUS          FIND          1               1        Ø
                                                  ❻

Channel number = 3
Access mode is sequential


Record:

        +---------+---------+---------+---------+---------+
 Ø      ØØØØØ1N K Jantakian Rare Coin Shop  Dept YT4
 5Ø                      Ø79 B Avenue                    Flag
 1ØØ    staff, AR                 86ØØ16Ø28937448N K Janta ❼
 15Ø    kian       ØØØØØØØ5ØØØØØØØØØØØØØØ777ØØØØØØØØØ777ØØ1
 2ØØ    ØØØØ1
        +---------+---------+---------+---------+---------+
```

Figure 14-7 (cont.) TPTRAC Annotated Output Listing

1. Trying to find a record.
2. Status indicates success.
3. Channel and Find operation information.
4. The key value used in the find operation.
5. Reading the next record.
6. Status and I/O information still OK.
7. Trace listing of record buffer contents.

```
                                                    (1)
0       12336  12336  12592   8270   8267  24936  29006  27469  24937   6302
10      24914  25970  17184  26991   8302  26707  28783   8224  25924  29808
-----------------/\------------------------------------------/\------------------
-----------------/\------------------------------------------/\------------------
90      12336  14135  12343  12336  12336  12336  12336  14135  12343  12592
100     12336  12336     49


***************  Send message (2)

Transaction     TST          Sequence         Status wds    Name
DPYCUS          FIND         1                1       0
                                                (3)

Proceed (4)

Message:
                                    (5)
        ---------+---------+---------+---------+---------+
0       000001N K Jantakian Rare Coin Shop  Dept YT4
50                     679 H Avenue                  Flag
100     staff, AR                  8600160289374480N K Janta
150     kian               500.00        7.77         7.77001
200     00001
        ---------+---------+---------+---------+---------+


***************  End of TST (6)

Transaction     TST          Sequence         Status wds    Name
DPYCUS          FIND         1

Message: (7)
        ---------+---------+---------+---------+---------+
0       000000N K Jantakian                      mk[6mk[6mk[6mk
50      [6mk[6mk[6mk[6
        ---------+---------+---------+---------+---------+
```

Figure 14-7 (cont.) TPTRAC Annotated Output Listing

1. Integer dump of record buffer contents.
2. TST sends a response message.
3. Status successful.
4. This is a "PROCEED" message.
5. Contents of message buffer.
6. TST exits here.
7. Show exchange message at conclusion of TST.

```
Workspace: (1)

        ---------+---------+---------+---------+---------+
 0      000001N K Jantakian Rare Coin Shop  Dept YT4
 50                  679 B Avenue                     Flag
 100    staff, AR                   8600160289374 48N K Janta
 150    kian              500.00       7.77         7.77001
 200    00001k[6mk[6mk[6mk[6mk[6mk[6mk[6mk[6mk[6mk[6mk[6mk
 250    [6mk[6     d              " Cromberg Stamp Gallery
 300                       " H K Sahoakian Coin & Stamp
 350               " H L Sehdalian Coins Co
        ---------+---------+---------+---------+---------+

                                    (2)
 0       12336   12336   12592    8270    8267   24926   29606   27489   24937    8302
 10      24914   25970   17184   26991    8302   26707   28783    8224   25924   29808
 ...
 170     21280   24948   28781    6224    8224     512      18    2561       1       0
 180        34    8264    8268   25939   25704   27745   24937    8302   28483   28265
 190      8307   28483


**************  Start of TST (3)

Transaction      TST          Sequence        Status Wds    Name
DPYCUS           NEXT         1

Message:
        (4)
        ---------+---------+---------+---------+---------+
 0      KYDOT N K Jantakian                      mk[6mk[6mk[6mk
 50     [6mk[6mk[6mk[6
        ---------+---------+---------+---------+---------+

Workspace:
        (5)
        ---------+---------+---------+---------+---------+
 0      000001N K Jantakian Rare Coin Shop  Dept YT4
 50                  679 B Avenue                     Flag
 100    staff, AR                   8600160289374 48N K Janta
 150    kian              500.00       7.77         7.77001
 200    00001k[6mk[6mk[6mk[6mk[6mk[6mk[6mk[6mk[6mk[6mk[6mk
 250    [6mk[6     d              " Cromberg Stamp Gallery
 300                       " H K Sahoakian Coin & Stamp
 350               " H L Sehdalian Coins Co
        ---------+---------+---------+---------+---------+
```

Figure 14-7 (cont.) TPTRAC Annotated Output Listing

1. Transaction workspace at end of TST.
2. Integer dump of transaction workspace area.
3. Second TST of DPYCUS starts here.
4. Exchange Message at start of TST.
5. Workspace at start of TST.

❶

```
Ø        12336  12336  12592   827Ø   8267  249Ø6  298Ø6  27489  24937   83Ø2
1Ø       24914  2597Ø  17184  26991   8372  267Ø7  28783   8224  25924  298Ø8
2Ø       22816  13396   8224   8224   8224   8224   8224   8224   8224   8224
3Ø        8224   8224   8224  14134   8249   8258  3Ø273  28261  25973   8224
4Ø        8224   8224   8224   8224   8224   8224   8224   8224  27718  26465
5Ø       29811  262Ø9  11366  16672   8274   8224   8224   8224   8224   8224
6Ø        8224   8224   8224  1388Ø  12336  13873  12848  14648  14131  13364
7Ø       2ØØ24  19232  18976  28257  24948  26987  28257   8224   8224   8224
8Ø        8224   8224   8224  136ØØ  12336  12334   824Ø   8224   8224   8224
9Ø       14112  14126   8247   8224   8224   8224  14112  14126  12343  12592
1ØØ      12336  12336  -5327 -18725  -5267 -18725  -5267 -18725  -5267 -18725
11Ø      -5267 -18725  -5267 -18725  -5267 -18725  -5267 -18725  -5267 -18725
12Ø      -5267 -18725  -5267 -18725  -5267 -18725  -5267 -18725      9     14
13Ø        484   -244      7      Ø    256   2817      1      Ø     34  29251
14Ø      28Ø15  25954  26482  2128Ø  24948  26781  182Ø8  27745  25964  31Ø9Ø
15Ø       8224   8224   8224   8224    256     2Ø   23Ø5      1      Ø     34
16Ø       8264   8267  24915  257Ø4  27489  24937   83Ø2  28483  28265   976Ø
17Ø      2128Ø  24948  28781   8224   8224    512     16   2561      1      Ø
18Ø         34   8264   8268  25939  257Ø4  27745  24937   83Ø2  28483  28265
19Ø       83Ø7  28483
```

```
****************  Send message

Transaction     IST           Sequence         Status Wds   Name
DPYCUS          NEXT          1                1      Ø     OUTSTA
```

Send a report using form PRTCUS ❷

```
Message:

        ---------+---------+---------+---------+---------+
 Ø      ØØØØØ1N K Jantakian Rare Coin Shop  Dept YT4
 5Ø                    679 B Avenue                  Flag
 1ØØ    staff, AR                   864Ø16Ø2893744ØN K Janta
 15Ø    kian              5ØØ.ØØ         7.77         7.77ØØ1
 2ØØ    ØØØØ1
        ---------+---------+---------+---------+---------+
```

Figure 14-7 (cont.) TPTRAC Annotated Output Listing

1. Integer dump of transaction workspace.
2. NEXT sends a Report Message to form "PRTCUS". Message contents shown here.

```
****************   Send message

Transaction      TST           Sequence        Status Wds    Name
DPYCUS           NEXT          1               1      0

Send reply number 2   ❶

Message:

        ---------+---------+---------+---------+---------+
 0      Hard Copy Listing of this Customer sent to LA180
 50
 100
 150
        ---------+---------+---------+---------+---------+



****************   End of TST   ❷

Transaction      TST           Sequence        Status Wds    Name
DPYCUS           NEXT          1

Message:

        ---------+---------+---------+---------+---------+
 0      KYDOT N K Jantakian                      mk[6mk[6mk[6mk
 50     [6mk[6mk[6mk[6
        ---------+---------+---------+---------+---------+

Workspace:

        ---------+---------+---------+---------+---------+
 0      000001N K Jantakian Rare Coin Shop  Dept YT4
 50                   679 B Avenue                    Flag
 100    staff, AR                  86401002893744 8N K Janta
 150    kian           500.00          7.77          7.77001
 200    00001k[6mk[6mk[6mk[6mk[6mk[6mk[6mk[6mk[6mk[6mk[6mk
 250    [6mk[6    d                  " Cromberg Stamp Gallery
 300                       " H K Sahdakian Coin & Stamp
 350                 " H L Sehdalian Coins Co
        ---------+---------+---------+---------+---------+
```

Figure 14-7 (cont.) TPTRAC Annotated Output Listing

1. This describes reply message sent to screen.
2. TST exits here. Message and Workspace contents are printed.

APPENDIX A
MACRO PROGRAMMING NOTES

A.1 WRITING A MACRO TST

A.1.1 MACRO Entry Point

Defining a MACRO Entry Point for a TST is done in exactly the same way that a MACRO Subroutine Entry Point is coded. The TST Entry Point conforms to the standard PDP-11 Linkage conventions outlined in the PDP-11 Processor Handbook.

The general form for defining the entry point in a MACRO TST required the following statements:

```
TSTEP:                  ;Standard TST name
        MOV 2(R5),R4    ;R4 now points to exchange message.
        MOV 4(R5),R3    ;R3 now points to TST workspace
```

Where:

TSTEP	Is the standard name which the TRAX operating system uses for all TSTs.
MOV 2(R5),R4	Takes the address of the exchange message from the TRAX-supplied argument list and stores it in R4.
MOV 4(R5),R3	Takes the address of the transaction workspace from the TRAX-supplied argument list and stores it in R3. If R3 contains a value of zero, then a workspace has not been defined for this transaction. Accessing address location 0 will cause unpredictable and possibly fatal results.

A.1.2 Calling a System Library Routine from a MACRO TST

You call a TRAX system library routine according to standard MACRO linkage conventions. In your TST, you must first prepare a parameter list containing the number of parameters and the addresses of each of the parameter fields. After depositing the appropriate values in these parameter fields, you call the library routine by the instructions:

```
MOV #ARGLST,R5
JSR RC,name
```

In this example, ARGLST is the symbol assigned to the first word of argument list containing the addresses of the argument fields required for the call. The "name" area of the JSR instruction should contain the mnemonic for the library routine you want to invoke.

Figure A-1 is an example of the argument list for the REPLY library routine.

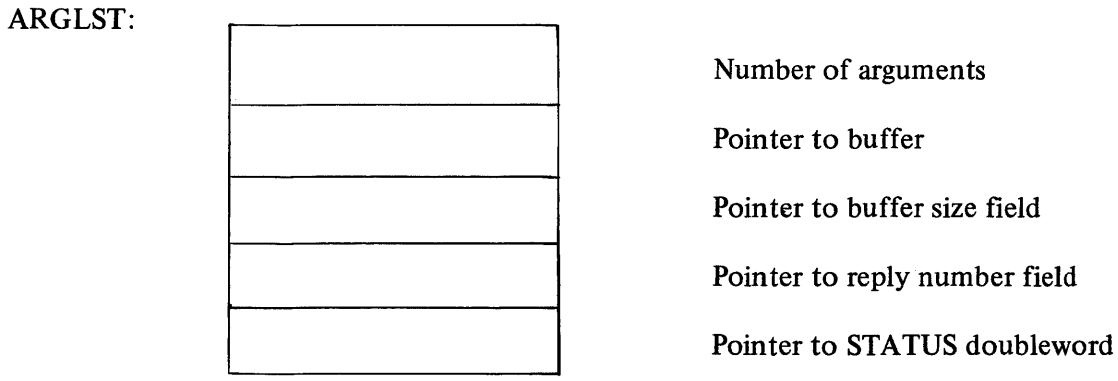


Figure A-1 Parameter List for "REPLY" Library Routine

APPENDIX B
COBOL TST EXAMPLE — RDCUST

The example in this Appendix shows the COBOL TST RDCUST. This TST is part of the CHGCUS transaction shown in Figure 1-2.

```
 COBOL  3,05   SRC:RDCUST.CBL;11          28-AUG-78  14:00:22 PAGE 001

CMD:RDCUST,RDCUST=RDCUST/TST
IDENT: 240140

  00001         IDENTIFICATION DIVISION.
  00002
  00003         PROGRAM-ID. TSTEP.
  00004         DATE-COMPILED. TODAY.
  00005
  00006
  00007             28-AUG-78 .
  00008         ENVIRONMENT DIVISION.
  00009
  00010         CONFIGURATION SECTION.
  00011             SOURCE-COMPUTER.  PDP-11.
  00012             OBJECT-COMPUTER.  PDP-11.
  00013
  00014         INPUT-OUTPUT SECTION.
  00015
  00016         FILE-CONTROL.
  00017
  00018             SELECT CUSTOM ASSIGN TO "CUSTOM.DAT"
  00019                    ORGANIZATION IS INDEXED
  00020                    ACCESS MODE IS DYNAMIC
  00021                    RECORD KEY IS CUSTOMER-NUMBER
  00022                    ALTERNATE RECORD KEY IS CUSTOMER-NAME WITH DUPLICATES
  00023                    FILE STATUS IS CUSTOMER-FILE-STATUS.
  00024
  00025
  00026         DATA DIVISION.
  00027
  00028         FILE SECTION.
  00029
  00030         FD  CUSTOM
  00031             LABEL RECORDS ARE STANDARD
  00032             VALUE OF ID IS CUSTOM-CHANNEL-NUMBER
  00033             DATA RECORD IS CUSTOMER-FILE-RECORD.
  00034
  00035         01  CUSTOMER-FILE-RECORD.
  00036
  00037             03  CUSTOMER-NUMBER                    PIC X(6).
  00038             03  CUSTOMER-NAME                      PIC X(30).
  00039             03  ADDRESS-LINE-1                     PIC X(30).
  00040             03  ADDRESS-LINE-2                     PIC X(30).
  00041             03  ADDRESS-LINE-3                     PIC X(30).
  00042             03  ADDRESS-ZIP-CODE                   PIC 9(5).
  00043             03  TELEPHONE-NUMBER                   PIC 9(10).
  00044             03  ATTENTION-LINE                     PIC X(20).
  00045             03  CREDIT-LIMIT-AMOUNT                PIC 9(10)V99.
  00046             03  CURRENT-BALANCE                    PIC 9(10)V99.
  00047             03  PURCHASES-YTD                      PIC 9(10)V99.
  00048             03  NEXT-ORDER-SEQUENCE-NUMBER         PIC 9(4).
  00049             03  NEXT-PAYMENT-SEQUENCE-NUMBER       PIC 9(4).
```

```
COBOL  3.05   SRC:RDCUST.CBL;11          28-AUG-78  14:00:22 PAGE 002

 00050
 00051        WORKING-STORAGE SECTION.
 00052
 00053        01  CUSTOM-CHANNEL-NUMBER           PIC X(11)
 00054                VALUE IS "CUSTOM/CH:3".
 00055
 00056        01  FILE-STATUS-WORD                          PIC XX.
 00057
 00058        01  CUSTOMER-FILE-STATUS                      PIC XX.
 00059
 00060        01  BUFFER-SIZE                               PIC S9999 COMP.
 00061
 00062        01  STATUS-WORDS.
 00063
 00064            03  STATUS-WORD-1                         PIC S9(4) COMP.
 00065            03  STATUS-WORD-2                         PIC S9(4) COMP.
 00066
 00067        01  REPLY-NUMBER                              PIC S9(4) COMP.
 00068
 00069        01  PROCEED-MESSAGE-BUFFER.
 00070
 00071            02  RM-CUSTOMER-FILE-RECORD.
 00072
 00073                    03  RM-CUSTOMER-NUMBER                    PIC X(6).
 00074                    03  RM-CUSTOMER-NAME                      PIC X(30).
 00075                    03  RM-ADDRESS-LINE-1                     PIC X(30).
 00076                    03  RM-ADDRESS-LINE-2                     PIC X(30).
 00077                    03  RM-ADDRESS-LINE-3                     PIC X(30).
 00078                    03  RM-ADDRESS-ZIP-CODE                   PIC 9(5).
 00079                    03  RM-TELEPHONE-NUMBER                   PIC 9(10).
 00080                    03  RM-ATTENTION-LINE                     PIC X(20).
 00081                    03  RM-CREDIT-LIMIT-AMOUNT       PIC Z,ZZZ,ZZZ.99.
 00082
 00083        01  REPLY-MESSAGE-BUFFER.
 00084            03  REPLY-MESSAGE-TEXT                    PIC X(80).
 00085            03  REPLY-FILLER                          PIC X(18)
 00086                    VALUE IS "File Status word: ".
 00087            03  REPLY-FSW                             PIC X(2).
 00088            03  REPLY-FILE-NAME                       PIC X(60).
 00089
```

```
COBOL  3.05   SRC:RDCUST.CBL;11          28-AUG-78  14:00:22 PAGE 003

 00090
 00091        LINKAGE SECTION.
 00092
 00093        01  EXCHANGE-MESSAGE.
 00094
 00095            02  EM-INPUT-FORM-CHCUS1.
 00096
 00097                    03  EM-CUSTOMER-NUMBER          PIC X(6).
 00098
 00099
 00100        01  TRANSACTION-WORKSPACE.
 00101
 00102
 00103            02  WS-CUSTOMER-FILE-RECORD.
 00104
 00105                    03  WS-CUSTOMER-NUMBER                  PIC X(6).
 00106                    03  WS-CUSTOMER-NAME                    PIC X(30).
 00107                    03  WS-ADDRESS-LINE-1                   PIC X(30).
 00108                    03  WS-ADDRESS-LINE-2                   PIC X(30).
 00109                    03  WS-ADDRESS-LINE-3                   PIC X(30).
 00110                    03  WS-ADDRESS-ZIP-CODE                 PIC 9(5).
 00111                    03  WS-TELEPHONE-NUMBER                 PIC 9(10).
 00112                    03  WS-ATTENTION-LINE                   PIC X(20).
 00113                    03  WS-CREDIT-LIMIT-AMOUNT              PIC 9(10)V99.
 00114                    03  WS-CURRENT-BALANCE                  PIC 9(10)V99.
 00115                    03  WS-PURCHASES-YTD                    PIC 9(10)V99.
 00116                    03  WS-NEXT-ORDER-SEQUENCE-NUM          PIC 9(4).
 00117                    03  WS-NEXT-PAYMENT-SEQUENCE-NUM        PIC 9(4).
 00118
```

```
COBOL  3.05   SRC:RDCUST.CBL;11          28-AUG-78  14:00:22 PAGE 004

 00119
 00120        PROCEDURE DIVISION USING EXCHANGE-MESSAGE, TRANSACTION-WORKSPACE.
 00121
 00122        DECLARATIVES.
 00123
 00124        I-O-ERROR SECTION.
 00125
 00126            USE AFTER STANDARD ERROR PROCEDURE ON CUSTOM.
 00127
 00128        CHECK-FILE-STATUS-CODE.
 00129
 00130            MOVE CUSTOMER-FILE-STATUS TO FILE-STATUS-WORD,REPLY-FSW.
 00131
 00132            IF FILE-STATUS-WORD IS EQUAL TO "10"
 00133                    MOVE "Reached End-of-File"
 00134                    TO REPLY-MESSAGE-TEXT
 00135                    GO TO SEND-REPLY-MESSAGE.
 00136
 00137            IF FILE-STATUS-WORD IS EQUAL TO "23"
 00138                    MOVE "No Record Exists under that Key"
 00139                    TO REPLY-MESSAGE-TEXT
 00140                    GO TO SEND-REPLY-MESSAGE.
 00141
 00142            IF FILE-STATUS-WORD IS EQUAL TO "92"
 00143            MOVE "The Record you wanted is
 00144        -         " locked by another user.  You may press CLOSE to exit,
 00145        -         " or you may wait and press ENTER to try again."
 00146            TO REPLY-MESSAGE-BUFFER,
 00147                    GO TO SEND-REPLY-MESSAGE.
 00148
 00149            MOVE "Transaction Aborted - I/O Error" TO REPLY-MESSAGE-TEXT,
 00150            MOVE "  Logical File Name: CUSTOM -CH3" TO REPLY-FILE-NAME.
 00151
 00152        SEND-ABORT-MESSAGE.
 00153
 00154            MOVE 164 TO BUFFER-SIZE
 00155            MOVE 2 TO REPLY-NUMBER
 00156            CALL "ABORT" USING
 00157                            REPLY-MESSAGE-BUFFER
 00158                            BUFFER-SIZE
 00159                            REPLY-NUMBER
 00160                            STATUS-WORDS.
 00161
 00162            GO TO END-ERROR-SECTION.
 00163
 00164        SEND-REPLY-MESSAGE.
 00165
 00166            MOVE 164 TO BUFFER-SIZE
 00167            MOVE 2 TO REPLY-NUMBER
 00168            CALL "REPLY" USING
 00169                            REPLY-MESSAGE-BUFFER,
 00170                            BUFFER-SIZE,
 00171                            REPLY-NUMBER,
 00172                            STATUS-WORDS.
 00173        END-ERROR-SECTION.
 00174        END DECLARATIVES.
 00175
```

```
COBOL  3.05   SRC:RDCUST.CBL;11          28-AUG-78  14:00:22 PAGE 005

00176       /
00177        MAIN-TST-ROUTINE SECTION.
00178
00179        READ-CUSTOMER-RECORD.
00180
00181            MOVE "00" TO FILE-STATUS-WORD.
00182
00183            OPEN INPUT CUSTOM.
00184            IF CUSTOMER-FILE-STATUS IS GREATER THAN "09"
00185            GO TO END-PROGRAM.
00186
00187            IF EM-CUSTOMER-NUMBER IS > "000000"
00188            GO TO KEY-OK.
00189                    MOVE 160 TO BUFFER-SIZE,
00190                    MOVE "You Specified an Invalid Customer ID #"
00191                    TO REPLY-MESSAGE-BUFFER,
00192                    MOVE 2 TO REPLY-NUMBER,
00193                    CALL "REPLY" USING REPLY-MESSAGE-BUFFER,
00194                                       BUFFER-SIZE,
00195                                        REPLY-NUMBER,
00196                                         STATUS-WORDS.
00197                    GO TO END-PROGRAM.
00198
00199        KEY-OK.
00200            MOVE EM-CUSTOMER-NUMBER TO CUSTOMER-NUMBER.
00201
00202            READ WITH LOCK CUSTOM RECORD.
00203            IF FILE-STATUS-WORD IS NOT EQUAL TO "00" GO TO END-PROGRAM.
00204            IF CUSTOMER-FILE-STATUS IS EQUAL TO "92" GO TO LOCKED-RECORD.
00205
00206            MOVE CUSTOMER-FILE-RECORD TO WS-CUSTOMER-FILE-RECORD.
00207
00208            MOVE CUSTOMER-NUMBER TO RM-CUSTOMER-NUMBER.
00209            MOVE CUSTOMER-NAME TO RM-CUSTOMER-NAME.
00210            MOVE ADDRESS-LINE-1 TO RM-ADDRESS-LINE-1.
00211            MOVE ADDRESS-LINE-2 TO RM-ADDRESS-LINE-2.
00212            MOVE ADDRESS-LINE-3 TO RM-ADDRESS-LINE-3.
00213            MOVE ADDRESS-ZIP-CODE TO RM-ADDRESS-ZIP-CODE.
00214            MOVE TELEPHONE-NUMBER TO RM-TELEPHONE-NUMBER.
00215            MOVE ATTENTION-LINE TO RM-ATTENTION-LINE.
00216            MOVE CREDIT-LIMIT-AMOUNT TO RM-CREDIT-LIMIT-AMOUNT.
00217
00218            MOVE 173 TO BUFFER-SIZE.
00219            CALL "PRCEED" USING PROCEED-MESSAGE-BUFFER,
00220                                BUFFER-SIZE,
00221                                 STATUS-WORDS.
00222
00223            GO TO END-PROGRAM.
00224
```

```
COBOL  3.05   SRC:RDCUST.CBL;11          28-AUG-78  14:00:22 PAGE 006

 00225        /
 00226         LOCKED-RECORD.
 00227
 00228              MOVE "The Record you wanted is
 00229        -          " locked by another user.  You may press CLOSE to exit,
 00230        -          " or you may wait and press ENTER to try again."
 00231
 00232              TO REPLY-MESSAGE-BUFFER.
 00233              MOVE 160 TO BUFFER-SIZE.
 00234              MOVE 2 TO REPLY-NUMBER.
 00235              CALL "REPLY" USING
 00236                                 REPLY-MESSAGE-BUFFER,
 00237                                 BUFFER-SIZE,
 00238                                 REPLY-NUMBER,
 00239                                 STATUS-WORDS.
 00240
 00241         END-PROGRAM.
 00242
 00243              EXIT PROGRAM.
```

APPENDIX C
BASIC TST EXAMPLE — RDCUST

The example in this Appendix shows the BASIC-PLUS-2 TST RDCUST. This TST is part of the CHGCUS transaction shown in Figure 1-2.

```
1       !*****************************************************************&
        !                          RDCUST                                 &
        !                                                                 &
        TST TSTEP(MSG,SPACE$,WRK,SPACE$)        ! Start TST here        &
        !                                                                 &
        !                                                                 &
        !                                                                 &
        !                                                                 &
        !       V A R I A B L E S   A N D    A R R A Y S   U S E D        &
        !                                                                 &
        !                                                                 &
        !                                                                 &
        !       NAME                    DESCRIPTION                       &
        !       ----                    ---------------------------       &
        !       EDIT,STRG$              DESCRIBES FORMAT OF PACKED FIELD&
        !                                                                 &
        !*****************************************************************&
        \  EDIT,STRG$  = "Z,ZZZ,ZZZ,99"                                   &

620     !*****************************************************************&
        !                                                                 &
        !               Exchange Message for Form CHCUS1                  &
        !                                                                 &
        \ MSGMAP        !CHCUS1!                ! Level = 03    !         &
        !                                                                 &
                EM,CUSTOMER,NUMBER$ =   6                                 &

700     !$****************************************************************&
        !                                                                 &
        !       W O R K S P A C E    D E F I N I T I O N                  &
        !                                                                 &

775     !*****************************************************************&
        !                                                                 &
        !                     Transaction Workspace                       &
        !                                                                 &
        !                        for TST RDCUST                           &
        !                                                                 &
        \ WRKMAP        !CUSTOM!                ! Level = 02    !         &
        !                                                                 &
                WS,CUSTOMER,FILE,RECORD$        =  205                    &
        !                                                                 &
        \ WRKMAP        !CUSTOM!                ! Level = 03    !         &
        !                                                                 &
                WS,CUSTOMER,NUMBER$     =   6                             &
        ,       WS,CUSTOMER,NAME$       =  30                             &
        ,       WS,ADDRESS,1$           =  30                             &
        ,       WS,ADDRESS,2$           =  30                             %
        ,       WS,ADDRESS,3$           =  30                             &
        ,       WS,ZIP,CODE$            =   5                             &
        ,       WS,TELEPHONE,NUMBER$    =  10                             &
        ,       WS,ATTENTION,LINE$      =  20                             &
        ,       WS,CREDIT,LIMIT$        =  12                             &
        ,       WS,CURRENT,BALANCE$     =  12                             &
        ,       WS,PURCHASES,YTD$       =  12                             &
        ,       WS,NEXT,ORDER,NUMBER$   =   4                             &
        ,       WS,NEXT,PAYMENT,NUMBER$ =   4                             &
        !*****************************************************************&
```

```
950     !*****************************************************************&
        !                                                                 &
        !               Customer File Record                              &
        !                                                                 &
        \ MAP   (CUSTOM)                          ! Level = 01    !       &
        !                                                                 &
              CUSTOMER.FILE.RECORD$   = 205                               &
        !                                                                 &
        \ MAP   (CUSTOM)                          ! Level = 03    !       &
        !                                                                 &
              CUSTOMER.NUMBER$        =   6                               &
        ,     CUSTOMER.NAME$          =  30                               &
        ,     ADDRESS.1$              =  30                               &
        ,     ADDRESS.2$              =  30                               &
        ,     ADDRESS.3$              =  30                               &
        ,     ZIP.CODE$               =   5                               &
        ,     TELEPHONE.NUMBER$       =  10                               &
        ,     ATTENTION.LINE$         =  20                               &
        ,     CREDIT.LIMIT$           =  12                               &
        ,     CURRENT.BALANCE$        =  12                               &
        ,     YTD.PURCHASES$          =  12                               &
        ,     NEXT.ORDER.NUMBER$      =   4                               &
        ,     NEXT.PAYMENT.NUMBER$    =   4                               &
        !                                                                 &

1000    !$****************************************************************&
        !                                                                 &
        !               M A I N    T S T    L O G I C                     &
        !                                                                 &
        !*****************************************************************&
        !                                                                 &
        \ ON ERROR GO TO 19000                                            &
        !                                                                 &
        !       Set up standard default error trap                        &

2000    !$****************************************************************&
        !                                                                 &
        !               O P E N   D A T A   F I L E S                     &
        !                                                                 &
        !*****************************************************************&

2100    !$****************************************************************&
        !                                                                 &
        !       C H A N N E L     A S S I G N M E N T S                   &
        !                                                                 &
        !       CHANNEL #       ASSIGNMENT                                &
        !       ---------       ------------------------------------      &
        !           3           Customer File          CUSTOM.DAT         &
        !                                                                 &
        \ CUSTOM.CHNL% = 3%                                               &

        !*****************************************************************&
```

```
2200    !*****************************************************************&
        !                                                                 &
        !               O P E N   F I L E   CUSTOM.DAT                    &
        !                                                                 &
        !       Assign the channel number.                                &

2220    \ OPEN "CUSTOM" AS FILE CUSTOM.CHNL%,                             &
              ORGANIZATION              INDEXED                           &
                                        VARIABLE,                         &
              ACCESS                    MODIFY,                           &
              ALLOW                     READ,                             &
              MAP                       CUSTOM,                           &
              PRIMARY KEY               CUSTOMER.NUMBER$,                 &
              ALTERNATE KEY             CUSTOMER.NAME$                    &
                                        DUPLICATES                        &
                                        CHANGES                           &
        !                                                                 &
        !       Open the Customer File.                                   &

3000    !$****************************************************************&
        !                                                                 &
        !     Test to see if exchange message is a legal value            &
        !                                                                 &

3010    \ GO TO 4100    IF EM.CUSTOMER.NUMBER$ > "000000"                 &
        \ GO TO 4200                                                      &
        !                                                                 &
        !       Two cases possible:                                       &
        !               Customer number entered,                          &
        !               Illegal number entered (operator error).          &

4100    !$****************************************************************&
        !                                                                 &
        !       C U S T O M E R   N U M B E R   E N T E R E D             &
        !                                                                 &

4110      GET #CUSTOM.CHNL%, KEY #0 EQ EM.CUSTOMER.NUMBER$,LOCK           &
        \ WS.CUSTOMER.FILE.RECORD$ = CUSTOMER.FILE.RECORD$                &
        \ GO TO 4800                                                      &
        !                                                                 &
        !       Read and lock the customer file.  ("Record not found"     &
        !         is trapped at statement 19020.)                         &
        !       Move the record into workspace.                           &

4200    !$****************************************************************&
        !                                                                 &
        !       N O   D A T A   F R O M   O P E R A T O R                 &
        !                                                                 &

4210      REPLY.BUF$ = "Operator Error - Invalid Key Supplied  "          &
        \ CALL REPLY BY REF (REPLY.BUF$,LEN(REPLY.BUF$),2%,STATUS%())     &
        \ GO TO 19950   IF STATUS%(0%) <> 1%                              &
        \ GO TO 32000                                                     &
        !                                                                 &
        !       Restart Transaction with Reply Message                    &
        !       Abort transaction if bad status.                          &
        !       Exit TST.                                                 &
```

```
4000      ED.RESULT$  =  FN.FORMAT$(EDIT,STRG$,WS.CREDIT.LIMIT$)          &
        \ WS.CREDIT.LIMIT$ = ED.RESULT$                                    &
        \ ED.RESULT$  = FN.FORMAT$(EDIT,STRG$,WS.CURRENT.BALANCE$)        &
        \ WS.CURRENT.BALANCE$ = ED.RESULT$                                 &
        \ ED.RESULT$  =  FN.FORMAT$(EDIT,STRG$,WS.PURCHASES.YTD$)          &
        \ WS.PURCHASES.YTD$ = ED.RESULT$                                   &

5000      CALL PROCEED BY REF (WS.CUSTOMER.FILE.RECORD$,205%.STATUS%())    &
        \ GO TO 19950 IF STATUS%(0%) <> 1%                                 &
        \ GO TO 32000                                                      &

5200    !$*****************************************************************&
        !                                                                  &
        !       C U S T O M E R   I D   N O T   O N   F I L E              &
        !                                                                  &

5210      REPLY.BUF$ = "No Record under that ID"                           &
        \ CALL REPLY BY REF (REPLY.BUF$,LEN(REPLY.BUF$),2%,STATUS%())      &
        \ GO TO 19950   IF STATUS%(?%) <> 1%                               &
        \ GO TO 32000                                                      &
        !                                                                  &
        !       Send Reply if Record Not Found                             &
        !       Abort Transaction if Bad Status Return                     &
        !       Exit TST.                                                  &
        !                                                                  &
        !*****************************************************************&

15000   !$*****************************************************************&
        !                                                                  &
        !       F U N C T I O N S     L O C A L     T O                    &
        !                                                                  &
        !                         T H I S    T S T                         &
        !                                                                  &
        !*****************************************************************&

15001   DEF FN.FORMAT$( FIELD$, INPUT$ )
15010   NUMBER.FORMAT$ = TRM$( INPUT$ )                                    &
        \ NUMBER.FORMAT$ = STRING$(LEN(FIELD$)-LEN(NUMBER.FORMAT$),48%)    &
                         + NUMBER.FORMAT$                                  &
        \ FOR I.FN% = LEN(FIELD$) TO 1% STEP -1%                           &
              \ CHAR.FORMAT$ = MID( FIELD$, I.FN%, 1% )                    &
              \ GO TO 15020   IF (CHAR.FORMAT$ = '9')                      &
                              OR (CHAR.FORMAT$ = 'Z')                      &
              \ CHAR.FORMAT$ = SPACE$(1%)     IF CHAR.FORMAT$ = 'B'        &
              \ NUMBER.FORMAT$ = MID( NUMBER.FORMAT$, 2%, I.FN%-1% )       &
                      + CHAR.FORMAT$                                       &
                      + RIGHT( NUMBER.FORMAT$, I.FN%+1% )
15020          NEXT I.FN%                                                  &
        \ FOR I.FN% = 1% TO LEN(FIELD$)                                    &
              \ CHAR.FORMAT$ = MID( FIELD$, I.FN%, 1% )                    &
              \ CHAR1.FORMAT$ = MID( NUMBER.FORMAT$, I.FN%, 1% )           &
              \ GO TO 15030   IF CHAR.FORMAT$ = '$'                        &
                              OR CHAR.FORMAT$ = '.'                        &
                              OR CHAR.FORMAT$ = '8'                        &
              \ GO TO 15040   IF CHAR.FORMAT$ = '9'                        &
              \ IF CHAR1.FORMAT$ <>  0                                     &
                AND CHAR1.FORMAT$ <>                                       &
                THEN  GO TO 15040                                          &
                ELSE  NUMBER.FORMAT$ =                                     &
                              SPACE$(1%)                                   &
                              + LEFT( NUMBER.FORMAT$, I.FN%-1% )           &
                              + RIGHT( NUMBER.FORMAT$, I.FN%+1% )
15030          NEXT I.FN%
15040    FN.FORMAT$ = NUMBER.FORMAT$
15050   FNEND                                                              &
```

```
19000   !$*****************************************************************&
        !                                                                 &
        !       S T A N D A R D     E R R O R     H A N D L I N G         &
        !                                                                 &
        !*****************************************************************&

19020   \ IF ERR = 155%                                                   &
          THEN                                                            &
                  RESUME 5200                                             &
        !                                                                 &
        !       Trap for customer id not on file.                         &

19040   !                                                                 &
        !       Trap for Record Lock Failure                              &
        !                                                                 &
        \ IF (ERR = 172% OR ERR = 154%)                                   &
          THEN                                                            &
          REPLY.BUF$ = "Access Denied, Record Locked by Another Task"     &
        \ CALL REPLY BY REF (REPLY.BUF$,LEN(REPLY.BUF$),2%,STATUS%())     &
        \ GO TO 32000                                                     &

19900   \ REPLY.BUF$ = " I-O Error Number "                               &
                       +NUM$(ERR)                                         &
                       +"at Line # "                                      &
                       +NUM$(ERL)                                         &
        !                                                                 &
        !       For unexpected errors, go to                              &
        !         system default error dump out.                          &

19950   !$*****************************************************************&
        !                                                                 &
        !       A B O R T   T H E   T R A N S A C T I O N                 &
        !                                                                 &
        !*****************************************************************&

19960     CALL ABORT BY REF (REPLY.BUF$,LEN(REPLY.BUF$),2%,STATUS%())     &
        \ GO TO 32000                                                     &
        !                                                                 &
        !  Standard ABORT handling:                                       &
        !       Send Reply with Abort to Terminal Station                 &
        !       Call TSTLIB routine to abort transaction.                 &
        !       No return is expected but nevertheless provide            &
        !         an orderly exit from TST.                               &

32000   !$*****************************************************************&
        !                                                                 &
        !       E N D     O F     P R O C E S S I N G                     &
        !                                                                 &
        !*****************************************************************&

32767   !$*****************************************************************&
        !                                                                 &
        !                 E N D     O F     T S T                         &
        !                                                                 &
        !*****************************************************************&
        !                                                                 &
        \ TSTEND
```

INDEX

ABORT,
  BASIC parameters, 4-10
  COBOL parameters, 4-9
  description, 4-8
  examples of usage, 3-15, 4-9, 4-10, 4-11
  library routine, 4-1, 4-8, 6-6
  response message, 3-7, 4-8
  status return codes, 4-11
Aborting a transaction,
  across a link, 6-3, 6-6
  with ABORT, 4-8
  with TABORT, 4-23
Application programmer,
  debugging TSTs, 7-1 to 7-13
  implement application design, 1-7
  integrating and testing application, 14-1 to 14-16
  primary function, 1-1
AROUTE library routine, 4-1
  BASIC parameters, 4-25, 4-26
  COBOL parameters, 4-25
  description, 4-25
  examples of use, 4-25, 4-26
  status return codes, 4-26
ATL utility, 1-2
  used in application security, 13-19
AUTDEF utility,
  adding a user authorization, 13-8
  commands, 13-6
  deleting a user authorization, 13-13
  editing a user authorization, 13-8
  function, 8-1, 13-6
  invoking, 13-6
  listing a user authorization, 13-11
  listing the index of user authorizations, 13-11
  role in application security, 13-17, 13-19

BASIC-PLUS-2, 1-1, 3-1
  APPEND command, 2-1
  coding MSGMAP statement, 2-6
  coding WRKMAP statement, 2-8, 2-9
  compiling TSTs in, 7-1
  data structure for TRAX/TL, 6-5, 6-8
BASIC-PLUS-2 (Cont.),
  data structure for TRAX/3271-TL, 6-12, 6-13
  debugger, 14-1, 14-4
  DELETE statement, 3-17
  FREE statement, 3-18
  GET statement, 3-16
  OPEN statement, 3-16
  PUT statement, 3-16
  /TST compiler switch, 7-1
  TST statement, 2-4
  TSTEND statement, 2-10
  UNLOCK statement, 3-18
  unsupported I/O statements, 3-18
  UPDATE statement, 3-17
Batch job,
  initiating a transaction, 5-1, 5-2
  STTRAN library routine, 5-2, 5-5
  submitted by a TST, 1-2, 5-1, 5-2

CHGCUS transaction,
  annotated design example, 1-4, 1-5, 1-6
CLOSE statement, 3-4
  disconnects permanent files, 3-5
  from BASIC TST, 3-17
  from COBOL TST, 3-14
CLSTRN,
  BASIC parameters, 4-16
  COBOL parameters, 4-15, 4-16
  description, 4-15
  examples of usage, 4-16, 6-6
  library routine, 4-1, 4-15, 6-6
  response message, 3-7, 4-15
  status return codes, 4-16
COBOL, 1-1, 3-1
  ACCEPT statements, 3-15, 14-4
  channel assignment, 3-9
  coding exchange message, 2-6
  coding transaction workspace, 2-8, 2-9
  compiling TSTs in, 7-1
  copy verb, 2-1
  DATA DIVISION, 3-10
  data structures for TRAX/TL, 6-4, 6-7
  data structures for TRAX/3271-TL, 6-11, 6-13
  DECLARATIVES section, 3-14
  DELETE verb, 3-14

COBOL (Cont.),
ENVIRONMENT DIVISION, 3-10
EXIT PROGRAM statement, 2-10
FD group item, 3-10
LINKAGE SECTION, 2-3, 2-6
OPEN STATEMENT, 3-11
PROCEDURE DIVISION header, 2-3, 2-4
READ verb, 3-12
REWRITE verb, 3-14
SELECT verb, 3-10
/TST compiler switch, 7-1
TST entry point, 2-3
UNLOCK ALL verb, 3-15
UNLOCK verb, 3-15
unsupported syntax, 3-15
USE procedure, 3-14
WRITE verb, 3-13
Compiling TSTs,
in BASIC, 7-1, 7-2
in COBOL, 7-1
TST switch, 2-1, 7-1
Crash recovery, 9-2
enabling, 9-8

DALLRT library routine, 4-1
BASIC parameters, 4-28
COBOL parameters, 4-27, 4-28
descriptions, 4-27
examples of usage, 4-28
status return codes, 4-28
Data,
available to a TST, 1-2
flow across a link, 6-1, 6-3, 6-9
flow through a transaction, 1-2
Data files,
adding records to, 3-7
closing, 3-5
deleting records from, 3-7
indexed, 3-2
maximum number, 9-7
opening, 3-4
permanent, 1-2, 3-2
reading records on, 3-7
relative, 3-1
sequential, 3-1
staged, 3-7
TST operations on, 3-1, 3-4
updating records on, 3-7
work, 1-2, 3-2
Date,
using GETIME library routine, 4-36
DEBUG utility, 2-2, 7-5, 7-7
dialog described, 7-7, 7-8 7-9
DEBUG Utility (Con't)
examples, 7-11 to 7-13
input data, 7-9
message files, 7-8
using, 7-7, 7-8, 7-9, 7-10, 14-1
workspace files, 7-9
Debugging,
TSTs in support environment, 7-7
TSTs in TP environment, 7-6, 14-1 to 14-16
DEC editor, 2-1, 2-2, 7-1
Definition Data Files, 8-1, 9-1
descriptions, 9-2
station definitions, 10-1
Deleting records, 3-7
in BASIC TSTs, 3-17
in COBOL TSTs, 3-14
methods, 12-5
on staged files, 3-9
Dialog conventions,
TRAX utilities, 8-2, 8-3
DROUTE library routine, 4-1
BASIC parameters, 4-27
COBOL parameters, 4-26
description, 4-26
examples of use, 4-27
status return codes, 4-27

Entry point,
TST, 2-3
Error recovery,
automatic exchange recovery, 11-3
BASIC TST ON ERROR routine, 3-18
COBOL USE procedure, 3-13, 3-14, 3-15
from locked record, 3-6
on I/O error, 3-71
Exchange, 1-1
definition dialog, 11-4
examples, 1-5, 1-7, 2-6, 2-8
label, 11-6
recovery, 11-3
restarting, 1-2, 2-1, 4-1, 4-19
subsequent action, 11-2, 11-7, 11-8
Exchange message, 1-1, 2-4, 2-5
and TST processing, 1-2, 2-1
as TST argument, 2-2, 2-4
data name, 2-3
defined in LINKAGE SECTION, 2-3
example, 1-5, 1-7

EXCHANGE message (Con't)
for SUBMIT command, 5-1
format for TRAX/TL, 6-2
format for TRAX/3271-TL, 6-9
from STTRAN library routine, 5-2, 5-3
logging, 11-3
maximum size, 2-5, 11-3
shown as TPTRAC output, simulation in debug utility, 7-8
specification, 2-5
Exchange routing list, 1-2, 1-5, 1-7, 2-2, 4-1, 4-24 to 4-28, 6-1, 11-7
Exchange time limit, defining, 11-6

FILDEF utility, 3-1, 3-3
adding a file definition, 12-3
and application security, 13-16, 13-18
and staging, 3-7
commands, 12-2
deleting a file definition, 12-11
described, 12-1, 12-2
editing a file definition, 12-3
function, 8-1, 12-1
invoking, 12-2
listing a file definition, 12-10
listing the index, 12-8
File Access Methods, sequential, 3-3
File name, logical, 3-1, 3-3
File sharing, 3-2, 12-5
maximum concurrent access, 12-5
read only files, 12-5
File specification format, 12-3, 12-4
Form definition file, 1-5
Form name, define for an exchange, 11-6

GETFIL library routine, 4-1
BASIC parameters, 4-44
COBOL parameters, 4-43
description, 4-42
examples of usage, 4-43, 4-44
status return codes, 4-44

GETIME library routine, 4-1
BASIC parameters, 4-37, 4-38
COBOL parameters, 4-36, 4-37
description, 4-36
examples of usage, 4-37, 4-38
status return codes, 4-38
GETMBX library routine, 4-1
BASIC parameters, 4-33
COBOL parameters, 4-31, 4-32
description, 4-31
examples of usage, 4-32, 4-33
status return codes, 4-33, 4-34
GETRAN library routine, 4-1
BASIC parameters, 4-42
COBOL parameters, 4-41, 4-42
description, 4-41
examples of usage, 4-42
status return codes, 4-42
GETSRC library routine, 4-1
BASIC parameters, 4-40, 4-41
COBOL parameters, 4-40
description, 4-40
examples of usage, 4-40, 4-41
status return codes, 4-41
GETSTN library routine, 4-1
BASIC parameters, 4-39
COBOL parameters, 4-38, 4-39
description, 4-38
examples of use, 4-39
status return codes, 4-39

Hard record lock, 3-5, 3-6
error codes,
examples, 3-6

Indexed files, 3-2
and file definition, 12-4
keys defined for, 12-4
maximum key length, 12-5
I/O channel,
assigning, 3-4
connecting, 3-4
disconnecting, 3-4

Journaling,
enabling with TPCTRL, 14-3
in file definition, 12-6

Linking TSTs,
the TSTBLD utility, 7-2, 7-6, 14-1

Logging,
  enabling with TPCTRL, 14-3
  user data from TSTs, 1-2, 4-1, 4-45 to 4-47
Logical file name, 12-3
LOGTRN library routine, 4-1
  BASIC parameters, 4-46, 4-47
  COBOL parameters, 4-45, 4-46
  description, 4-45
  examples of usage, 4-46, 4-47
  status return codes, 4-47

MACRO, 1-1
  usage notes, A-1
Mailbox stations,
  TST operations on, 1-2, 4-1, 4-28 to 4-35
Master link station,
  defined in TPDEF, 9-6
  used in TRAX/TL, 6-1 to 6-8
  used in TRAX/3271-TL, 6-9 to 6-13
MBXNUM library routine, 4-1
  BASIC parameters, 4-35
  COBOL parameters, 4-34
  description, 4-34
  examples of usage, 4-34, 4-35
  status return codes, 4-35
Messages,
  see
    Exchange message
    Mailbox message
    Receive link message
    Report message
    Response message

OPEN statement, 3-4
  in BASIC TST, 3-16
  in COBOL TST, 3-11

Passwords,
  for user authorization, 13-8
Permanent data files, 3-1
  access shown by TPTRAC, 14-8 to 14-16
  defined using FILDEF, 12-1 to 12-11
  disconnected by CLOSE statements, 3-5
  opened by transaction processor, 3-2
  used by a transaction processor, 12-1
  TST record operations on, 1-2
PRCEED library routine, 4-1
  BASIC parameters, 4-12
  COBOL parameters, 4-11, 4-12
PRCEED library routine (Con't)
  description, 4-11
  examples of use, 4-12, 4-13
  in slave transaction, 6-6
  status return codes, 4-13

Random access to files, 3-3
RDCUST TST, 1-5
  compiled, 7-1
  linked, 7-5, 7-6
Reading records, 3-7
  in BASIC TSTs, 3-16
  in COBOL TSTs, 3-13
  trace output from, 14-11
Receive link message, 9-6
  size in slave transaction, 6-1
Record layout, 3-10
  specification sheet, 3-12
Record locking, 3-5
  actions that cause, 3-5
  and work files, 3-2
  hard lock, 3-5, 3-6, 3-13, 12-6
  in COBOL TSTs, 3-12, 3-13, 3-14
  in permanent data files, 3-2
  lock wait interval, 12-5
  procedure, 3-5
  read access to locked records, 12-6
  soft lock, 3-5, 3-6, 3-13, 12-6
Relative files, 3-1
Repeat,
  exchange parameter, 11-8
REPLY,
  BASIC parameters, 4-7, 4-8
  COBOL parameters, 4-6, 4-7
  examples of usage, 3-15, 3-18, 4-7, 4-8
  library routine, 4-1, 4-5, 4-6
  response message, 3-7, 4-5
  shown with TPTRAC, 14-16
  status return codes, 4-8
  used in TRAX/TL, 6-6, 6-7
REPORT library routine, 4-1
  BASIC parameters, 4-4
  COBOL parameters, 4-2, 4-3
  description, 4-2
  example of use, 4-3, 4-4
  status returns, 4-5
Report messages,
  in TRACE output, 14-15
  sent by a TST, 1-2, 4-1, 4-2, 14-15
Response message, 1-1
  ABORT, 3-7, 4-1, 4-5, 4-8, 4-9, 4-10, 4-11

RESPONSE message (Con't)
CLSTRN, 3-7, 4-1, 4-5, 4-15, 4-16
example, 1-5, 1-7
in terminal initiated transaction, 4-5
PRCEED, 4-5, 4-11, 4-12, 4-13, 14-14
REPLY, 3-7, 4-1, 4-5, 4-6, 4-7, 4-8
sent by a TST, 1-2, 4-1, 4-5
sent to slave batch station, 5-2
sent to slave link station, 6-6
STPRPT, 4-1, 4-5, 4-13, 4-14, 4-15
TRNSFR, 4-1, 4-5, 4-17, 4-18, 4-19
RESTRT library routine, 4-1
BASIC parameters, 4-20
COBOL parameters, 4-19
description, 4-19
examples of usage, 4-19, 4-20
status return codes, 4-20
unsuccessful, 4-20
REWRIT TST, 1-7

Sequential access to files, 3-3
Sequential files, 3-1
deletions not allowed, 3-7
only one TST has write access, 3-5
rewinding, 3-5
SERCTL utility, 14-2
SIGNOF,
form, 13-19, 13-22
incorporating into application, 13-13, 13-15
transaction, 13-12, 13-13, 13-14, 13-15, 13-19
TST, 13-17, 13-22
work classes, 13-13
SIGNON,
form, 13-19, 13-20, 13-21
incorporating into application, 13-13, 13-15
transaction, 13-12, 13-13, 13-14, 13-15, 13-20
TST, 13-17, 13-21
work class, 13-17, 13-22
Skeleton source files, 2-1
Slave batch station, 5-2
Slave link station, 6-6 to 6-8
defined in STADEF, 10-13
defined in TPDEF, 9-6

SNDMBX library routine, 4-1
BASIC parameters, 4-30
COBOL parameters, 4-29
description, 4-29
examples of usage, 4-30, 4-31
status return codes, 4-31
Soft record lock, 3-5, 3-6
error codes,
example, 3-6
Software Error Log, 2-2, 14-1
SERANL, 14-2
SERDAY, 14-2
SERLOG, 14-1
Source language libraries, 2-1
Spawning a transaction instance,
aborting a spawned transaction, 4-23
with TSPAWN, 4-20
STADEF utility,
and application security, 13-17
commands, 10-2
defining mailbox stations, 10-13, 10-14
defining master link stations, 10-11 to 10-13
defining slave batch stations, 10-13
defining slave link stations, 10-13
defining submit batch stations, 10-13
defining terminal stations, 10-3 to 10-7
defining TST stations, 10-8 to 10-10
deleting station definitions, 10-8
function, 8-1
invoking, 10-1
listing station definitions, 10-15, 10-17
listing station index, 10-15, 10-16
station types, 10-3
Staging files, 3-7, 12-6
Stations, 1-1, 9-2, 10-1 to 10-18
and routing lists, 4-25 to 4-28
defining using STADEF, 10-1 to 10-18
mailbox, 1-2, 4-1, 9-7, 10-1, 10-13, 10-14
master link, 6-1 to 6-13, 9-6, 10-1, 10-11, 10-12
slave batch, 5-2, 9-7, 10-1, 10-13

STATIONS (Con't)
slave link, 6-6 to 6-9, 9-6, 10-1, 10-13
submit batch, 5-1, 9-6, 10-1, 10-13
terminal, 4-1, 9-5, 10-1, 10-3 to 10-5, 13-17
TST, 1-1, 1-5, 1-7, 2-1, 9-6, 10-1, 10-5 to 10-8, 14-10
STPRPT library routine, 4-1
BASIC parameters, 4-14, 4-15
COBOL parameters, 4-13, 4-14
description, 4-13
examples of usage, 4-14, 4-15
status return codes, 4-15
used with TRAX/TL, 6-6, 6-7
STTRAN library routine,
BASIC parameters, 5-4, 5-5
COBOL parameters, 5-3
description, 5-2
examples of use, 5-4, 5-5
status return codes, 5-5
Submit batch station, 5-1
SUBMIT command,
sent from TST to batch processor, 5-1
Subsequent action,
definition, 11-7, 11-8
System library routines, 1-2
calling from TSTs, 4-1
listed, 4-1
tracing calls to, 14-8
System workspace,
calculating, 11-5
size, 11-4

TABORT library routine, 4-1
BASIC parameters, 4-24
COBOL parameters, 4-23
description, 4-23
examples of use, 4-24
status return codes, 4-24
Terminal stations, 4-1, 9-5
associated work class, 10-4, 13-17
defining, 10-3
Time,
using GETIME library routine, 4-36
TPCTRL utility,
commands, 14-3
installing a TP, 14-3
removing a TP, 14-6
starting a TP, 14-3, 14-4
stopping a TP, 14-6
using, 14-2

TPDEF utility, 9-1 to 9-14
and application security, 13-13
commands, 9-4
copying a TP definition, 9-13
creating a TP definition, 9-4, 9-8, 9-10
deleting a TP definition, 9-14
dialog description, 9-3
editing a TP definition, 9-4, 9-8
function, 8-1
printing a TP definition, 9-11, 9-12
renaming a TP definition, 9-13
TP INDEX, 9-8, 9-9, 9-11
TPTRAC utility,
annotated output, 14-9 to 14-16
example of dialog, 14-8
using, 14-4, 14-6
TRADEF utility,
adding a transaction definition, 11-2 to 11-10
and application security, 13-14, 13-15
commands, 11-2
deleting a transaction definition, 11-12
editing a transaction definition, 11-2 to 11-10
function, 8-1
invoking, 11-1, 11-2
listing the index of transaction definitions, 11-11
listing a transaction definition, 11-11, 11-12
Transaction, 1-1
CHGCUS, 1-2 to 1-4
control, 4-1
debugging in TP environment, 14-1
definition, 1-2, 1-5, 1-7, 2-1, 2-5, 2-6, 11-1 to 11-13
design of, 1-1
design using TRAX/TL, 6-1 to 6-8
design using TRAX/3271-TL, 6-9 to 6-13
example of, 1-2, 1-3, 1-4
names in work class definitions, 13-2, 13-5
SIGNOF, 13-15 to 13-24
SIGNON, 13-15 to 13-24
types, 9-2

Transaction instance, 1-1, 2-1
aborting from a TST, 1-2, 4-1, 4-23, 4-24
initiated by batch job, 5-2
instances, 9-2
maximum concurrent, 9-5
record context, 3-4
spawned from a TST, 1-2, 4-1, 4-20, 4-21
traced output, 14-9 to 14-16
Transaction processor, 1-1
calls TST, 2-1
controlled by TPCTRL, 14-2 to 14-6
data structures, 9-1, 9-2
debugging, 2-2
definition file, 9-1
definition process, 8-1, 9-1 to 9-14
installing and testing, 14-1 to 14-16
opens data files, 3-1, 3-4
passes parameters to TST, 1-1, 2-3
specification sheet, 9-9
tracing operations of, 14-7 to 14-17
Transaction processor definition utilities, 1-1
AUTDEF, 8-1, 9-2
dialog conventions, 8-2, 8-3
FILDEF, 3-1, 3-3, 8-1, 9-2
STADEF, 1-1, 8-1, 9-2, 10-1 to 10-18
TPDEF, 8-1, 9-1 to 9-14
TRADEF, 8-1, 9-2, 11-1 to 11-13
WORDEF, 8-1, 9-2
Transaction processor trace facility, 2-2, 14-1, 14-4, 14-9 to 14-16
Transaction slot, 1-1
maximum size, 9-7, 9-8
traced output, 14-10 to 14-16
Transaction Step Tasks (TSTs), 1-1
actions performed by, 1-2
calling parameters, 1-1
coding, 2-1, 2-2
compiling, 2-2, 7-1
debugging, 1-1, 7-6 to 7-13, 14-1
developing, 2-2
exiting from, 2-9
integrating into a transaction processor, 1-1
linking, 2-2
preparing for I/O, 5-4
station, 1-1, 1-5, 1-7, 2-1
Transaction Step Tasks (TSTs) (Cont.),
structural requirements, 2-2
submitting batch jobs, 5-1
using, 2-1
using system library routines, 4-1
writing for TRAX/TL, 6-1 to 6-8
writing for TRAX/3271-TL, 6-9 to 6-13
Transaction workspace, 1-1, 1-2, 2-6
as TST argument, 2-2, 2-4, 2-6
coding in COBOL, 2-8
data name, 2-3
defined in LINKAGE SECTION, 2-3
logging, 11-3
maximum size, 2-6
simulated in debug utility, 7-9
size, 11-3
specification, 2-7, 2-8
trace output showing, 14-10 to 14-16
transmits data across exchanges, 2-6
WKMAP statement, 2-8, 2-9
TRAX/TL, 6-1 to 6-8
TRAX/3271-TL, 6-9 to 6-13
TRNSFR library routine, 4-1
BASIC parameters, 4-18
COBOL parameters, 4-17
description, 4-17
examples of usage, 4-18
status return codes, 4-19
used in TRAX/TL, 6-6, 6-7
TSPAWN library routine, 4-1
BASIC parameters, 4-22
COBOL parameters, 4-20, 4-21
description, 4-20
examples of usage, 4-21, 4-22
status return codes, 4-22
TSTBLD utility, 4-1
and TP debug, 14-1, 14-4
dialog described, 7-2 to 7-4
examples of use, 7-5, 7-6, 14-1
specification defaults, 7-3
using, 2-2, 7-2, 14-1
TSTEP,
required as TST program ID, 2-3
required in TST statement, 2-4

Updating records, 3-7
in BASIC TSTs, 3-17

UPDATING records (Con't)
in COBOL TSTs, 3-13, 3-14
on staged files, 3-9
User authorizations,
and work classes, 13-8, 13-21
defined, 13-7
names, 13-8, 13-20
passwords, 13-8, 13-20
role in application security, 13-7, 13-17, 13-19, 13-21

VALIDC TST, 1-7

WAIT,
exchange parameter, 11-7, 11-8
WORDEF utility,
adding a work class definition, 13-2
commands, 13-2
deleting a work class definition, 13-5
editing a work class definition, 13-2
function, 8-1, 13-1
WORDEF utility (Con't)
invoking, 13-1
listing the index of work classes, 13-4
listing a work class definition, 13-5
role in application security, 13-17, 13-19
Work classes, 13-1 to 13-6
defined, 13-1
for terminal station, 10-4
in user authorization, 13-8, 13-21
names, 13-2, 13-3, 13-8
role in application security, 13-1, 13-17, 13-19
Work files, 3-2, 12-4
defined using FILDEF, 12-1 to 12-11
opened and closed, 3-2
TST operations on, 1-2
used by a transaction processor, 12-1
Writing records, 3-7
in BASIC TSTs, 3-17
in COBOL TSTs, 3-13
to staged files, 3-9

TRAX Application
Programmer's Guide
AA-D329A-TC

READER'S COMMENTS

NOTE: This form is for document comments only. DIGITAL will use comments submitted on this form at the company's discretion. If you require a written reply and are eligible to receive one under Software Performance Report (SPR) service, submit your comments on an SPR form.

Did you find this manual understandable, usable, and well-organized? Please make suggestions for improvement.

Did you find errors in this manual? If so, specify the error and the page number.

Please indicate the type of user/reader that you most nearly represent.

- ☐ Assembly language programmer
- ☐ Higher-level language programmer
- ☐ Occasional programmer (experienced)
- ☐ User with little programming experience
- ☐ Student programmer
- ☐ Other (please specify)

Name _____ Date _____

Organization _____

Street _____

City _____ State _____ Zip Code or Country _____

---------- Fold Here ----------

---------- Do Not Tear - Fold Here and Staple ----------

FIRST CLASS
PERMIT NO. 33
MAYNARD, MASS.

BUSINESS REPLY MAIL
NO POSTAGE STAMP NECESSARY IF MAILED IN THE UNITED STATES

Postage will be paid by:

digital

Software Documentation
146 Main Street ML 5-5/E39
Maynard, Massachusetts 01754