July 1978

This manual is directed to the TRAX application designer. This person is responsible for studying the functional specification of a proposed system and developing an appropriate technical design. The application designer may have one of several titles — system analyst, analyst/programmer, chief programmer. If you are responsible for selecting an approach for an application or if you must make technical design decisions while developing a TRAX application, you should read this manual.

TRAX
Application Designer's Guide

Order No. AA-D328A-TC

SUPERSESSION/UPDATE INFORMATION: This is a new manual.

OPERATING SYSTEM AND VERSION: TRAX 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

CONTENTS (CONT.)

CONTENTS (CONT.)

CONTENTS (CONT.)

CONTENTS (CONT.)

CONTENTS (CONT.)

FIGURES

FIGURES (CONT.)

PREFACE

This manual is directed to the TRAX application designer. This person is responsible for studying the functional specification of a proposed system and developing an appropriate technical design. The application designer may have one of several titles – system analyst, analyst/programmer, chief programmer. If you are responsible for selecting an approach for an application or if you must make technical design decisions while developing a TRAX application, you should read this manual.

This manual presents the structure and operation of TRAX transaction processing applications. The manual has two main parts:

Part One. Chapters 2 through 11 present general concepts and facilities of the TRAX system. This Part gives an application designer the background needed to develop a business application – what TRAX is, and what it can do.

Part Two. Chapters 12 through 23 describe the procedure an application designer follows when designing a TRAX business application. This Part presents methods and techniques for the design process and provides worksheets for many phases of the design process.

You can get details on the TRAX support environment and other specific technical topics by consulting these TRAX reference manuals:

TRAX Application Programmer's Guide	AA-D329A-TC
TRAX Application Terminal Language (ATL) Reference Manual	AA-D330A-TC
TRAX Support Environment User's Guide	AA-D331A-TC
TRAX System Manager's Guide	AA-D332A-TC
TRAX System Generation Manual	AA-D335A-TC
TRAX SORT Reference Manual	AA-D346A-TC
TRAX DATATRIEVE User's Guide	AA-D347A-TC
TRAX BASIC-PLUS-2 Language Reference Manual	AA-D336A-TC
TRAX BASIC-PLUS-2 User's Guide	AA-D337A-TC
TRAX COBOL Language Reference Manual	AA-D338A-TC
TRAX COBOL User's Guide	AA-D339A-TC
DEC EDITOR Reference Manual	AA-D347A-TC
TRAX Linker Reference Manual	AA-D342A-TC

PART ONE

Introduction to TRAX Concepts and Facilities

CHAPTER 1
INTRODUCTION TO TRAX

TRAX is a computer system specially designed to meet the requirements of business transaction processing.

Here are some of its features:

- On-line terminals suitable for transaction processing as well as management information systems (MIS)
- Excellent response time and throughput characteristics
- A system of modest size that can be tailored to the business organization
- Facilities for a network of systems, permitting distributed data processing (DDP)
- A sophisticated data management system called RMS
- Reliable, recoverable data processing – complete with journals, audit trails, and (when necessary) transaction restart
- A choice of two standard, high-level languages: ANSI-74 COBOL and DIGITAL's BASIC-PLUS-2
- Unattended batch processing

The rest of this chapter is a brief description of TRAX features and their application to business data processing.

1.1 THE TRANSACTION

A *transaction* is the exchange of information, money, or goods between two or more parties. You execute a transaction when you withdraw money from your bank account, purchase gasoline with a credit card, leave a forwarding address with the post office, or make a plane reservation.

Sometimes, a transaction involves several interactions between parties. For example, in a store you might see something that does not have a marked price. If you are interested, you first ask the price, and then perhaps decide to purchase the item. This transaction has two interactions between yourself and the storekeeper; with TRAX we would call this a *two-exchange transaction.*

Transactions are basic to business. And although electronic data processing has been used for many years to process and record transactions, most processing is done after the fact, using written records of the original transactions. In comparison, TRAX is designed to process transactions as they occur.

1.2 WHAT IS A TRANSACTION PROCESSING SYSTEM?

A *transaction processing system* is an on-line system for processing business transactions. Even though it is an on-line system, a transaction processing system does not necessarily process transactions as they occur. Transactions can still be recorded manually or by other automated means and then entered into the transaction processing system.

More and more, though, business is realizing the benefits of having a real-time *transaction processing system* available at the time the transaction occurs. Paperwork can be eliminated, and the system always contains up-to-date data.

A real-time transaction processing system can check entered data immediately. Errors can be caught and corrected promptly while the customer is available for questioning and the user has the transaction in mind.

This approach, *real-time transaction processing,* demands a reliable system with backup and audit trail capabilities, because paperwork has been eliminated.

1.3 RESPONSE TIME AND THROUGHPUT

Transaction processing also demands good *response time.* This means that the transaction processing system must record the transaction in an amount of time that will not annoy or frustrate the transaction's participants . . . one of whom will often be an impatient customer.

On the other hand, adequate *throughput* refers to the system's capability of keeping up with its workload – that is, being able to process data quickly enough to keep it from backing up.

The portions of TRAX that handle on-line transaction processing are designed for maximum throughput and optimum response times. These system characteristics are important to any application designer. The design methods by which these are accomplished are discussed in later chapters of this manual.

1.4 FORMS AND TRANSACTION PROCESSING

Most businesses make extensive use of forms. They do this because forms are more convenient than blank paper. Forms structure a task by specifying the information needed; and they save writing effort by preprinting routine information. An order blank is a typical form. A good order blank is easy to fill out, because it lets the customer know what information is needed – a catalog page number, a shipping and handling charge, and the like.

Not only is the order blank more convenient for the customer, it speeds the processing of the order. Firms that use preprinted order blanks can usually process orders faster than firms that do not, because the form helps staff members to collect complete, sequenced information.

TRAX makes extensive use of the "forms" concept. In fact, all communication between application terminal users and the system is done with forms. Within the system, the user is the "customer" and the system is the "salesperson." But the effect is the same: the system does not have to "interview" the user. TRAX displays a predefined form and leaves the user to fill it out. Until the user is finished, TRAX is not involved in the user data entry. Only when the form has been completed does TRAX begin to process the user's data.

1.5 TRAX APPLICATION TERMINALS

The efficiency of TRAX forms is greatly enhanced by the microprocessor-based terminals that are used with the system. These terminals receive complete forms from TRAX, display them, and assure that the user enters data correctly to those forms. With these terminals, TRAX is freed from monitoring user input. No central system support is needed from the time the empty form is sent to the terminal until the time the correctly completed form is returned to the system.

TRAX is the first system to support this terminal, the VT62. The VT62 provides a variety of screen display methods and user data entry checks, all under the control of its resident microprocessor. In addition, it offers features not generally available in terminals of its type:

- Synchronous data communication protocol for maximum data transmission reliability and speed
- Multi-drop capability to allow several terminals to be connected to a single data communication line
- Attached (but completely independent) hard-copy printers
- Editing keys to allow the user to edit the contents of data entry fields
- Function keys that can be enabled and sensed by application programs
- Internally generated error messages for common data entry errors
- Full numeric keypad

You can find more about the VT62 terminal and its forms-display capabilities in the *TRAX Application Terminal Language Reference Manual.*

1.6 DISTRIBUTED DATA PROCESSING

Distributed data processing (DDP) is the technique of adapting to business organizations a number of small, widely dispersed data processing stations, rather than a big centralized data processing systems. DDP enables the manager to support his operational area with an appropriate data processing system, while relying on other departments' data processing systems where necessary.

TRAX is well suited for distributed data processing applications for several reasons:

- TRAX handles inter-system transactions, as well as transactions originated from terminals.
- TRAX provides extensive inter-system communication facilities.
- TRAX systems are moderate size – large enough to accommodate impressive processing power yet small enough to be deployed outside traditional data processing centers.

In fact, the microcomputers inside the VT62 application terminals represent the first level of a distributed data processing configuration: some processing is done within the terminal itself.

1.7 DATA MANAGEMENT SYSTEM

A system that lacks a sophisticated *data management system* cannot be effectively used for business applications. If a good data management system does not exist, you must build it before application implementation can proceed.

For most business applications, this means support for at least three major kinds of files:

- Sequential files
- Relative-record (random) files
- Indexed files

Indexed file structures are the most powerful and convenient for programmers to access; they are generally the most often used kind of file in business applications. For maximum flexibility in business applications, indexed files should permit:

- Multiple or alternate indexes (that is, cross-reference indexes)
- Retrieval of a series of records in sequential key order
- Random retrievals of specific records

TRAX provides all of these features and more through DIGITAL's business-oriented data management system, RMS.

One more data management feature is mandatory in any transacation processing system: the ability for many users to share access to common data files. Here, TRAX provides such features as a record-locking capability, automatic retry of retrievals on locked records, the ability to access locked records in a read-only fashion, and more. Taken together, these features allow many different users to work in the same files with minimum interaction.

1.8 RELIABILITY

As transaction processing techniques are adopted, two things begin to take place:

- An increasing proportion of business records are recorded on magnetic media within the transaction processing system
- A decreasing proportion of business records remain on paper or other tangible, hard-copy media.

This means that many transaction processing systems carry data that are unavailable from any other source. Even if data are available from other sources, conversion and replacement may not be economically feasible. The loss of this data might be disastrous to a business, so you must assure that the data on your transaction processing system are safe.

Data safety means:

- Unauthorized access must be prohibited.
- Authorized access to sensitive data must be logged.
- Catastrophic system failures must not risk massive data loss.

NOTE

In many applications, *any* data loss – however slight – cannot be permitted.

Even if a system catastrophe does not cause loss of data, the unavailability of a transaction processing system may cause problems. If the system is unavailable, the business may not be able to function.

Several other system reliability requirements are:

- Proven hardware reliability
- Quick recovery from system catastrophes

- Ability to reconfigure a system or divert work to other systems
- Fault detection and measurement facilities that detect faults before they turn to disaster

TRAX provides you with the facilities you need to satisfy these system requirements.

1.9 LANGUAGES

Most application designers try to program their business applications in an accepted, high-level language, using as many of the language's standard features as possible and avoiding system-dependent features where feasible. This approach:

- Minimizes training of your application programmers
- Makes hiring additional qualified programmers easier
- Allows new programmers to adapt to the resulting programs with minimal confusion
- Develops better programs with programmers of average skill
- Makes testing and debugging easier.

TRAX provides you with your choice of two high-level languages that have been popular for business applications: COBOL and BASIC-PLUS-2. Both languages are straightforward extensions of their industry counterparts.

COBOL-11 conforms closely to the standards of ANSI-74 COBOL, except for a few features required by the architecture of TRAX. These changes have been held to a minimum, and TRAX allows a COBOL programmer to work easily with the COBOL he knows. In particular, the methods that TRAX uses to control application terminals avoid many problems often created when COBOL is used with interactive devices. With TRAX, data coming from and going to application terminals resembles fixed-length records from ordinary data files – something that COBOL handles extremely well.

BASIC-PLUS-2 has always been strong in on-line applications. Its interactive debugging sessions speed the programmer's work. The version of BASIC-PLUS-2 supplied with TRAX builds on the strengths of BASIC, while adding features that are important in business applications: longer mnemonic variable names, data structures for input/output operations, and improved commenting facilities.

Choose a language according to your preference and the background of your programming staff. Either choice gives you a strong, flexible language for a transaction processing environment.

1.10 OVERNIGHT PROCESSING

Business applications require occasional overnight or other batch-oriented processing. Files must be reorganized and backed up, printed reports must be run, and other activities must be accomplished with minimum human intervention.

These activities are best handled with a batch processing facility, where predetermined sequences of operations are performed on request or on a periodic schedule. A good batch processing system keeps a log of batch processing events, runs with minimum human intervention, and reacts appropriately to faults and error conditions during a batch run.

TRAX provides these capabilities and more. Several batch processors can operate at once, working from a common work queue or separate queues. Printing despoolers allow work to be optimally scheduled among printer units. And assignable device names allow complete operator flexibility in setting up each batch run.

1.11 HOW DOES TRAX DO ALL THIS?

A TRAX system has several elements:

- **A kernel** (an underlying operating system) provides rudimentary peripheral device support, memory allocation, and scheduling services.
- **A support environment** is used for nontransaction processing: batch processing and on-line programming, debugging, and system management. Programmers and operators sitting at support environment terminals have access to a general-purpose time-sharing environment which is not available to application terminals. The multi-user nature of this environment allows quicker program development and permits program development to proceed in parallel with application operation.
- **A transaction processor** handles each set of on-line application terminals that runs a particular application. A transaction processor is a set of software modules and tabular specifications that can process a predefined set of transactions, and makes these transactions available to users at application terminals by predefined access rules.

  A TRAX system can have several transaction processors defined. Each can be started or stopped as desired. Depending on the size and capacity of the system, two transaction processors may be active concurrently.

The support environment and the transaction processors are kept separate by the TRAX kernel. Programming and system management terminals can access only the support enviornment; application terminals can access only an assigned transaction processor. This separation provides superior system security and allows each part of the system to be optimally deisgned for its primary purpose.

To design and build an efficient transaction processing system, you must concentrate on the construction of an appropriate transaction processor – and this is what the remainder of this manual will cover.

CHAPTER 2
THE TRAX SAMPLE APPLICATION

Throughout this manual and other TRAX manuals, you find examples of business applications. These are drawn from a prototype business application supplied with each TRAX system. This prototype application is called the *TRAX Sample Application.*

2.1 THE BUSINESS PROBLEM

The TRAX Sample Application is designed to solve a paperwork problem for a moderately large wholesale distribution business specializing in collectable American coins.

The system supports other processing and the associated invoicing and payment cycle. It also assists certain other company operations such as maintaining an up-to-date customer list.

The coin firm is called the TRAX Coin Corporation. It deals primarily with retailers rather than individual collectors. Most sales are made on credit; that is, the ordered coins are shipped with an invoice for payment. Most business is done by mail or over the phone, but customers also buy coins over the counter.

TRAX Coin Corporation has several order handling policies they intend to keep even after the new system is installed:

- Orders are accepted for coins that are not presently in stock, and TRAX Coin then tries to acquire the specified coins from its suppliers. Items in this status are called "back-ordered."
- If only a few items on an order are back-ordered, the remainder of the order is shipped right away. But partial quantities of any item are never shipped.
- When replacement stock is received, back-orders are given priority over current orders. That is, orders are always filled on a first-come, first-served basis for each item.
- Invoices are sent after each shipment. The balance on the invoice is due when received.
- A single order can, of course, have several shipments. (This will occur for any order that has a back-ordered item.) Each of these shipments is accompanied by a separate invoice; each invoice covers only the coins in that shipment.
- The balance for each invoice is carried in TRAX Coin Corporation books until it is paid. If payments are received without specific instructions, the funds are applied against the oldest invoices first.
- The TRAX Coin Corporation keeps a journal of invoices and payments received.

Figure 2-1 shows the order processing cycle from the time the order is received until corresponding invoices are paid. This is the cycle that the new system is meant to support.

2.2 THE SAMPLE APPLICATION ADDRESSES THE BUSINESS PROBLEM

The functions of the TRAX Sample Application correspond with the manual functions shown in Figure 2-1. The system handles the following activities, significantly reducing the manual paperwork:

- Order entry
- Miscellaneous functions, such as checking stock availability and price
- Printing multi-part order paperwork
- Keeping the back-order file
- Activating back-order paperwork when new stock arrives
- Keeping the invoice file
- Keeping the payments journal
- Keeping related master files, such as the customer file and the inventory file

The Sample Application supports a large proportion of its processing in an on-line transaction-processing mode. That is, the company personnel assigned to many of the tasks listed here are given conversational terminals attached to the system, and they enter data through those terminals as they go about their task. The system records the data they enter, and it occasionally takes other actions such as printing order paperwork.

In addition to on-line transaction-oriented processing, the Sample Application also needs some off-line batch processing.[1] This is periodic processing that needs no human intervention – such as file reorganization and backup.

The Sample Application uses the following data files:

- **Customer File.** This file provides expanded customer names, addresses, and related information by a unique customer identification number. It is used so that the customer's full name and address need not be included in other files.
- **Order and Invoice File.** This file contains complete information on each order:
  - General order information, such as the customer identification number, order data, and shipping instructions
  - Each order line item, including quantity and price
  - A record of each invoice issued for the order
- **Inventory File.** This file contains a record of each stock item carried by TRAX Coin Corporation. The file carries data such as quantity on hand, price, stock number, and description.
- **Payments File.** This file records payments received and the invoices to which they are applied.
- **Back-Order File.** This file records items that are in "back-order" status.

(If you study the Sample Application carefully, you will see that the order and invoice file really is two data files for technical design reasons: the main order and invoice file, containing the data listed before, and an order access file that serves as a cross-reference to relate all of the data for a given order.)

[1] Not supplied with the TRAX distribution list.

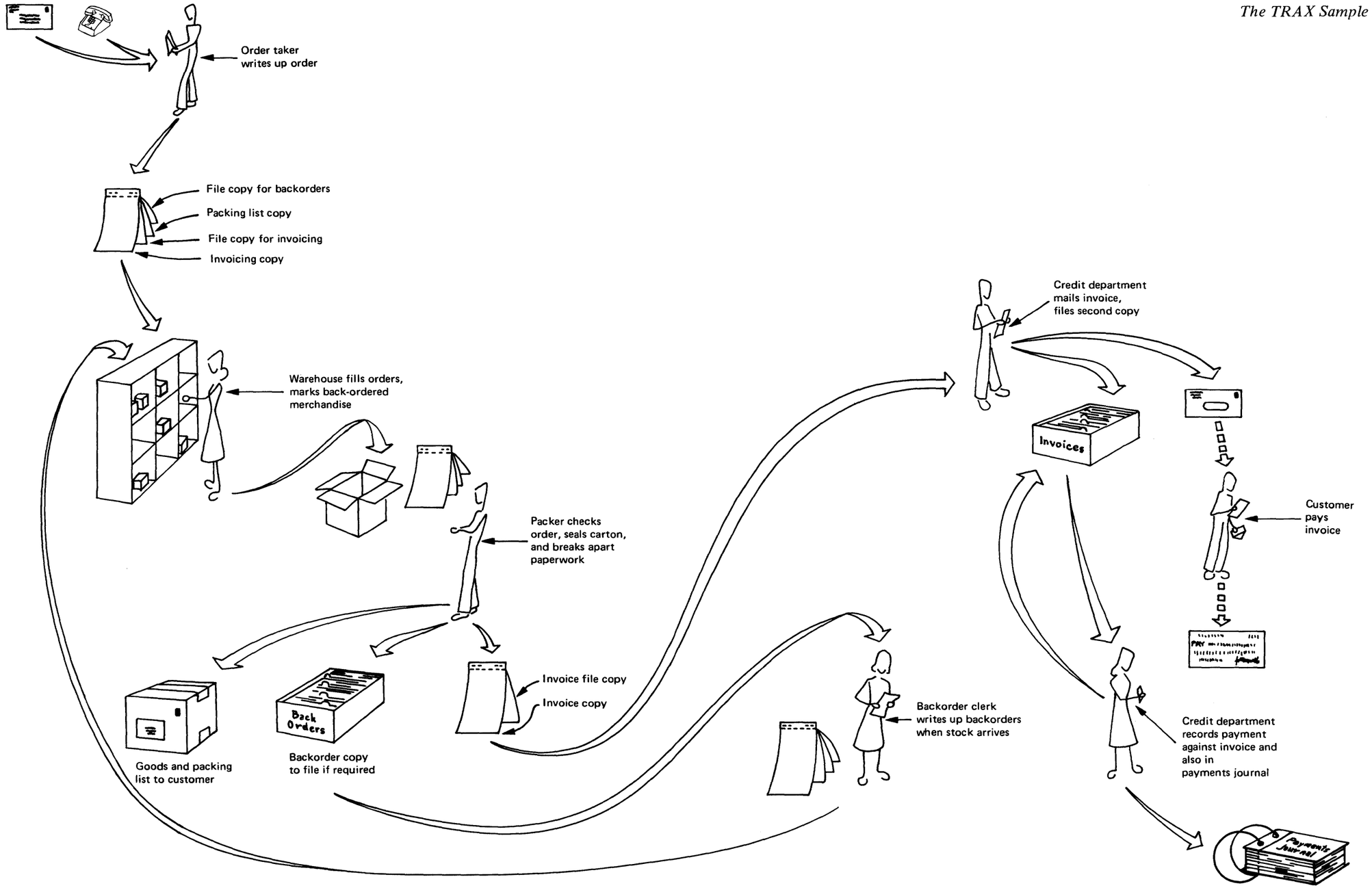


Figure 2-1 TRAX Coin Corp. Order Processing Cycle

2.3 THE EXAMPLES IN THIS MANUAL

The examples in this manual are taken from the group of four on-line transactions used to maintain the customer file. These transactions are:

- **Add Customer.** Inserts a new customer record into the customer file.
- **Change Customer.** Changes data in a customer record.
- **Delete Customer.** Marks a customer record obsolete, so that a subsequent file reorganization will remove it.
- **Display Customer.** Displays the data in a customer record or a sequence of customer records.

Although these functions are only a small subset of the entire TRAX Sample Application, they provide enough examples of situations in which an application designer is required to make important decisions.

CHAPTER 3
AN INTRODUCTION TO TRANSACTION PROCESSORS

By now you know that a TRAX *transaction processor* supports on-line transaction-oriented processing and consists of a collection of tabular specifications and software modules.

Before you begin to design applications with TRAX, you must learn more details about transaction processors and how they operate. First, though, you will probably find it helpful to have an overview of how a transaction processor works and how this method of on-line processing differs from others.

3.1 A SAMPLE TRANSACTION

A sample transaction that changes customer records is used as an example throughout this chapter. This transaction involves a two-step conversation with a terminal user each time it is executed.

- First, the transaction asks for the identification number of the customer whose data is going to be changed (Figure 3-1). The user enters an identification number, the corresponding record is retrieved from the customer file, and the transaction moves to the second step.
- Second, the data from the file is displayed (Figure 3-2). The user is permitted to change whatever data he likes. When he is done, the final version of the data is checked for consistency and replaced in the customer file.

This is the overall structure of the transaction. There are of course several additional path variations in actual use, to permit error messages and corresponding error recovery procedures.

3.2 IMPLEMENTING THE SAMPLE TRANSACTION ON SYSTEMS OTHER THAN TRAX

To implement the sample transaction on an interactive business sytem other than TRAX, you would probably use the technique shown in Figure 3-3. This figure shows several terminals, each associated with its own copy of a transaction program.

If a user at terminal 2, for instance, wants to execute this sample transaction, he needs a copy of the appropriate program from the program library. Within this program would be enough code to support the functions of asking for a customer identification number, reading the record from the customer file, displaying it, helping the user edit the data, checking the final data values, and rewriting the data into the customer file.

If other terminals wish to execute the same transaction, each needs its own copy of the same program. Further, each of the programs is "active" for the duration of its own transaction – even though the program spends most of its time "waiting" for the user to finish typing his input.

3.2 IMPLEMENTING THE SAMPLE TRANSACTION ON TRAX

Implementing this transaction on TRAX is entirely different.

Figure 3-1 First Form of Change Customer Transaction

Figure 3-2 Second Form of Change Customer Transaction

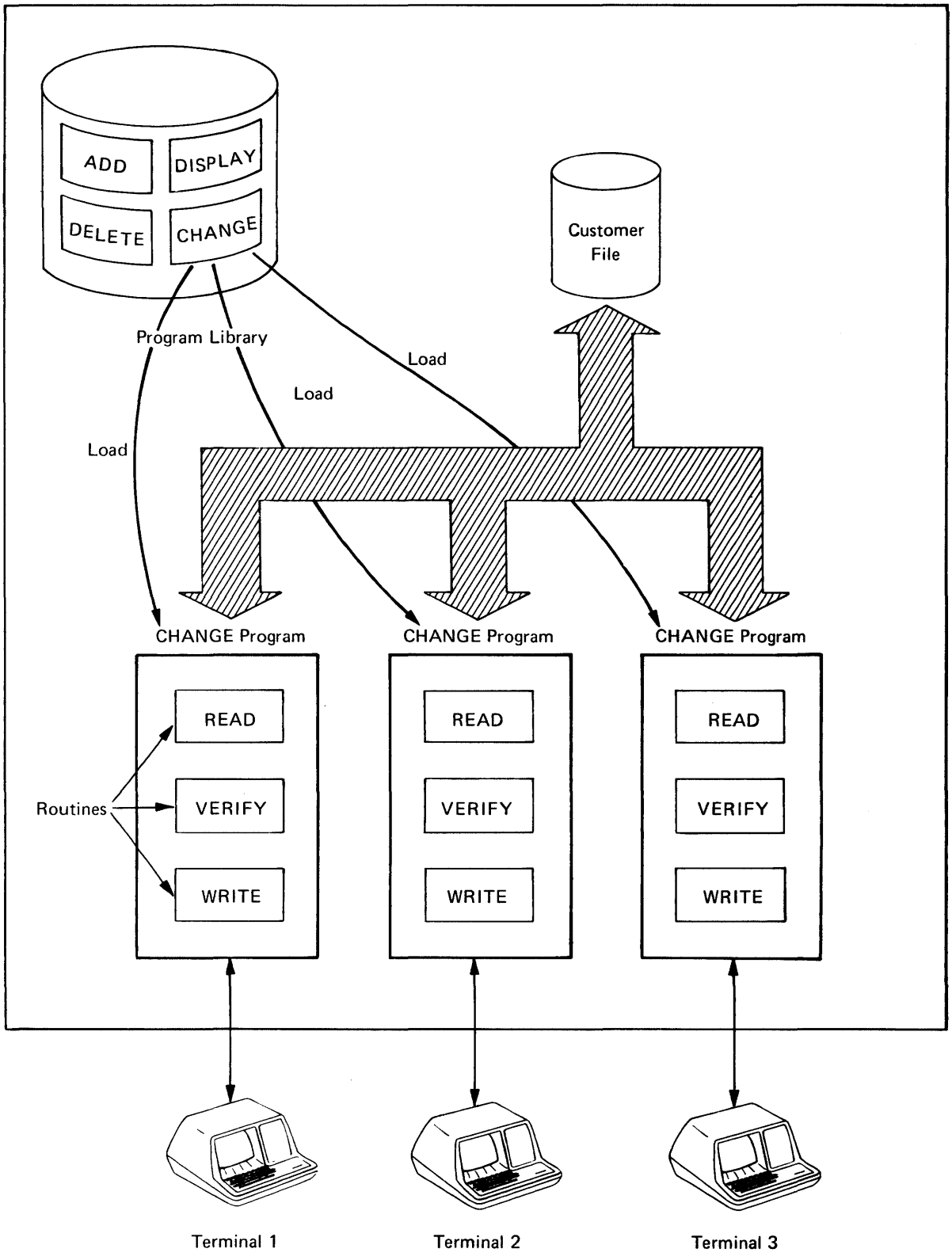


Figure 3-3 Implementation of Sample Transaction System Other than TRAX

To implement the sample transaction under TRAX, you would construct a transaction processor. You would include in the processor the necessary programs for reading the customer record, validating the new data, and writing the modified record back into the file. Although this may sound similar to the usual method of implementing on-line business sytems, there are two important differences:

- The programs are never directly associated with any given terminal; rather, each program is capable of communicating with all terminals that are associated with the transaction processor.
- The programs are never directly associated with any given transaction; rather, any program can process part of any transaction.

This means that application programs no longer "control" either a terminal or a transaction. Instead, they become a generally available resource to be invoked whenever necessary. The control of transaction execution rests with two other components of a transaction processor:

1. **Sets of tabular specifications** composed by you when you set up a transaction processor. They determine the sequence of programs used at any stage in the execution of a transaction.
2. **System software modules** provided with TRAX and customized to the requirements of each transaction processor. They interpret the tabular specifications and manage the activities of the transaction processor accordingly.

Figure 3-4 shows how a transaction processor might handle the sample transaction. When a user at terminal 2 enters a customer identification number, this information is sent to the first program (called RDCUST), so that the corresponding customer record can be read (Step ❶ in Figure 3-4). This program is selected by the transaction processor according to a predefined list of programs that specify the processing for this particular transaction.

The RDCUST program reads the record and sends the data back to the terminal (Step ❷). *The RDCUST program is now free to process other data from other terminals.*

When the data is displayed at the terminal, the user can change the data as he wishes. He then sends it back to the transaction processor. The transaction processor consults the list of programs used by this transaction and forwards the user's data to the VALIDC program (Step ❸). When this program assures that the data is valid, it terminates. The transaction processor sends the same data to the REWRIT program (Step ❹). This program rewrites the updated data into the customer file. A confirming message is then sent to the terminal signaling the end of the transaction.

3.4 FORMS ARE IMPORTANT TO TRANSACTION PROCESSORS

Terminal conversation in a transaction processor is strictly forms oriented. This means that all conversation between a transaction processor and an application terminal is accomplished via predefined specifications. These specifications are called *form definitions.*

Terminals are controlled by system software modules that are part of each transaction processor. These system software modules use form definitions that you have prepared. The form definitions describe the displays that must be presented and the rules for user data entry. These specifications are forwarded to TRAX application terminals at the proper time, and the specifications are interpreted and executed by microprocessors inside each terminal. In this way, terminals can be effectively controlled during data entry without an active application program to control them.

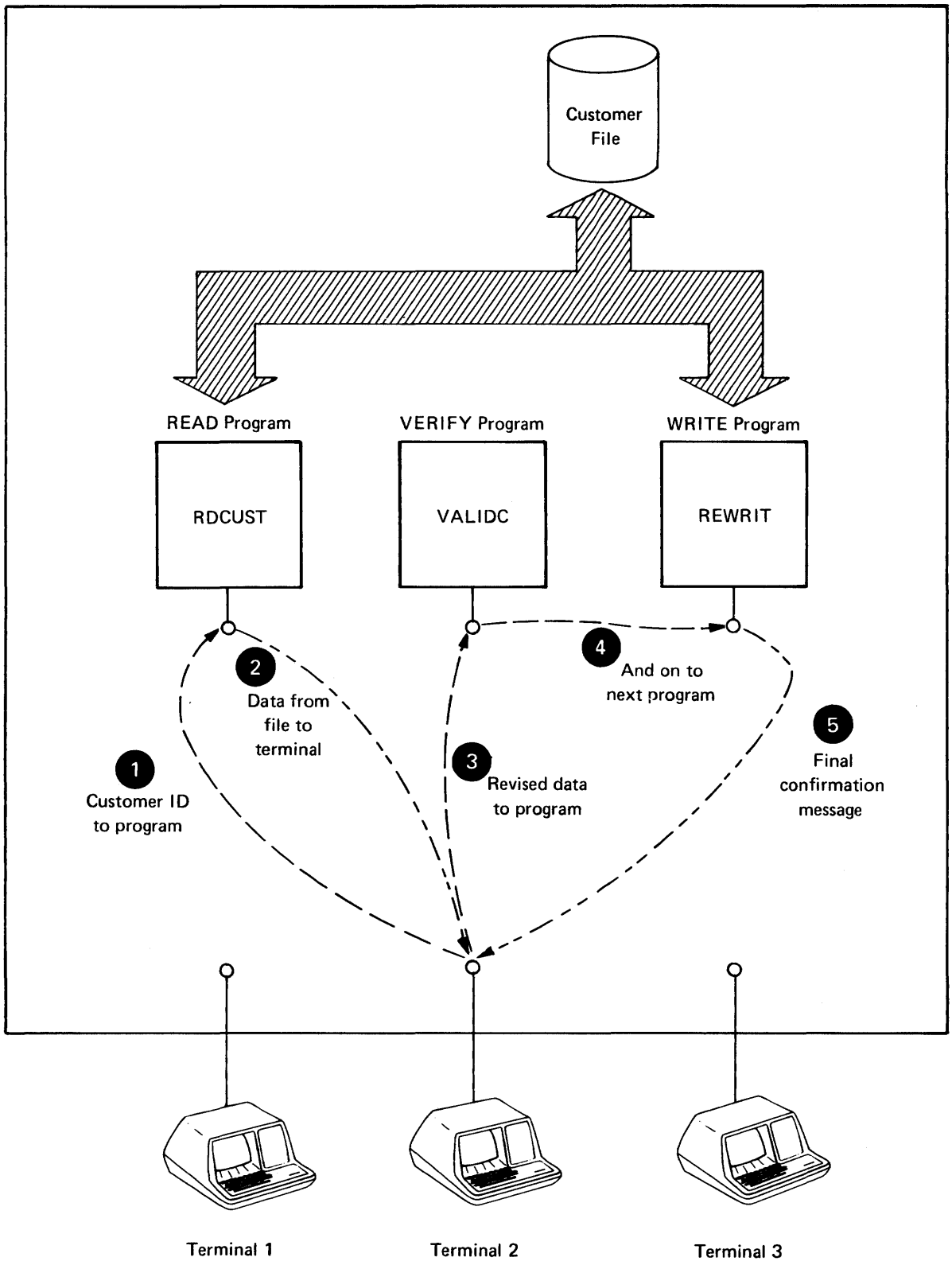


Figure 3-4 Implementation of Sample Transaction TRAX using a TRAX Transaction Processor

The conversation at an application terminal follows this sequence:

1. A form is displayed according to its definition.
2. The user enters data in predefined fields or edits data already in those fields. During this process, he must conform to the data entry rules that are part of the form definition.
3. The user presses a function key, which sends the data to the transaction processor.
4. The transaction processor takes one of two actions after processing the data:

   It may direct the user to another form, and this series of steps is repeated using the new form; or

   It may direct changes to the current form and the series of steps repeat for the modified form. The transaction processor can only select modifications that are defined and included in the form definition.

This approach differs significantly from a typical conversational approach where questions appear one by one on the terminal screen and are answered as they appear.

Each form available to a transaction processor is defined by a form definition. This definition specifies the appearance of the form and the placement of fields and captions. It also specifies:

- How data coming from a preceding program is interpreted and included in the display
- Where and how the user enters data to the screen
- Which function keys the user employs
- What data is assembled from the user entries and sent to subsequent programs
- Where and how error messages (and other form modifications) are displayed

In summary, then, forms are important to transaction processors because they are the means by which the transaction processor communicates with its terminals. Application programs cannot communicate directly with terminals and are not active when communication takes place. To design workable transaction processors, you must think exclusively in terms of a forms-oriented conversation between application programs and the transaction processor.

3.5 IMPORTANT TRANSACTION PROCESSOR TERMS

It is time to introduce some important transaction processor terms. These terms refer to components of a transaction processor or to the transaction processor's unique methods of handling transactions.

- **A Transaction** is a generalized procedure for the exchange of information, money, or goods. It does *not* refer to the details of any particular act of exchange.
- **A Transaction Instance** is a specific execution of a transaction.

For example, the procedure for buying a coin involves a sequence of actions, both for the customer and for the salesman. This sequence of actions is a *Transaction.* But when a specific customer buys a specific coin, this is a *Transaction Instance.* The Transaction is the sequence of actions by which business is transacted; a Transaction Instance is a specific case in which a transaction is executed.

So, if the manager of a coin shop talks about how to sell a certain kind of coin, we would say he was discussing a Transaction. On the other hand, if he talks about a particular customer and the specific coins that the customer recently purchased, we would say he was discussing a Transaction Instance.

Maintaining a distinction between these two terms is important to you. You must be comfortable with their meanings.

- **An Application Terminal** is a terminal that is served by a transaction processor; that is, it is a terminal used for transaction processing.

  The connection between an Application Terminal and a transaction processor is made when the transaction processor is started. In effect, a transaction processor "seizes" a specified set of terminals when it begins execution (such as the beginning of the business day). These terminals cannot be used for another purpose until the transaction processor is stopped. But once stopped, the transaction processor releases its terminals and these terminals may be seized by any other transaction processor that subsequently begins execution.

  Terminals that have access to the support environment of a TRAX system are called Support Environment Terminals, even though a user may be running an application-related program. That program would be a stand-alone support environment program; it would not be related to a transaction processor; and the terminal would not be considered an Application Terminal.

  Application Terminals are handled by TRAX in a way different from support environment terminals. You can see this in the communication methods used with Application Terminals: synchronous communication lines, multi-drop configurations, and formal line protocol. None of these are possible with support environment terminals. The distinction between Application Terminals and support environment terminals is made during the *configuration* and *sysgen* processes, and a single terminal cannot be used for both the support and application programs.

  This distinction between Application Terminals and support environment terminals improves system security (Section 10.1.1).

  Figure 3-5 shows the distinction between Application Terminals and support environment terminals. This figure shows a large TRAX system, with two transaction processors and on-line application development proceeding at the same time. Notice that some of the application terminals are not attached to any active transaction processor; these terminals *cannot* access the support environment. For these terminals to be placed in service, one of the two active transaction processors must be stopped and another started.

  Application Terminals can be either video displays or hard copy devices. The video displays are used only as interactive (that is, transaction processing) devices; hard copy devices are used only as unattended output-only printers.

  In the output-only mode of operation, a terminal receives and prints data for any transaction instance. In the interactive mode of operation, a terminal is associated with a given transaction instance until that transaction instance terminates.

- **An Exchange.** A transaction often requires several steps – that is, several interactions – between the transaction's participants. Each of these interactions is called an Exchange.

  To understand this, let's return to the example of the coin firm. The purchase of a coin is frequently a three-exchange transaction.

  1. You ask the clerk to recite the list of coins available in the category that interests you.
  2. After listening to the list, you may ask to buy a coin.
  3. Finally, after being given the coin, you pay for it. The clerk returns your change with a receipt.

Each Exchange, then, is an interaction between the party requesting service (you) and the server (the coin store clerk). In a transaction processor, each Exchange is an interaction between the terminal user and the transaction processor. The terminal user enters data; the transaction processor processes it and usually returns an answer to the user.

- **A Transaction Step Task (TST)** is an application program within a transaction processor. It is really a subroutine that helps process various transactions. A TST is not a complete, stand-alone program for it does not have control of the transactions it participates in.

  In the coin store example, the clerk is like a TST. The clerk receives something (a question, an order, or some money) from the customer, takes some action, and then returns something else (information, a coin, or change and a receipt) to the customer.

  Notice something important about this example: *the clerk can handle several customers at the same time.* While you are selecting a coin, another customer can be asking about coins, ordering, or paying for his purchase. In fact, if you had the clerk to yourself, you would be wasting his time while you select a coin.

  We can extend this example to illustrate transactions involving two or more TSTs in one exchange. Although the clerk serves customers by himself, he may have a cashier to take customer money and wrap purchases. When you buy a coin, both the clerk and the cashier become involved. The clerk handles the first exchange (where you ask questions) and the second exchange (where you buy a coin). But the third exchange (where you pay for a coin and receive change and a receipt) requires the participation of the cashier.

  Notice that both the clerk and the cashier can handle other orders between the steps (or *Exchanges*) of your order, even if other customers are looking at or buying different merchandise. Both the clerk and the cashier approach their jobs one task at a time. This same characteristic applies to TSTs within a transaction processor.

- **A Message** is the data structure that elements of a transaction processor use to communicate with each other. For example, data must be sent from an application terminal to one or more application programs (TSTs) for processing. The data are sent to the TST in Messages.

  There are many different kinds of messages that can be sent within a transaction processor. They are classified by their purpose and destination. The two most frequently used kinds of messages are exchange messages and response messages.

  - **Exchange Messages** are messages that travel from an application terminal to a series of TSTs; they carry data entered by the user.
  - **Response Messages** are messages generated by TSTs and directed to the user's terminal.

  In the coin example, your questions, requests, and money represent exchange messages; the clerk's answers are response messages.

- **A Station** receives messages for a component of a transaction processor. The Station serves the same purpose as the "in-box" on your desk, constantly ready to receive messages. In the same way that an "in-box" keeps you from being interrupted by arriving mail, a Station keeps its transaction processor component from being interrupted by arriving messages. The Station buffers the messages and releases them one at a time as needed.

  There are many kinds of stations. The two most frequently used kinds of stations are terminal stations and TST stations.

  - **Terminal Stations** receive messages that are directed to their terminals. Each Application Terminal has its own Terminal Station.
  - **TST Stations** receive messages that have been directed to their TSTs. Each TST has its own TST Station.

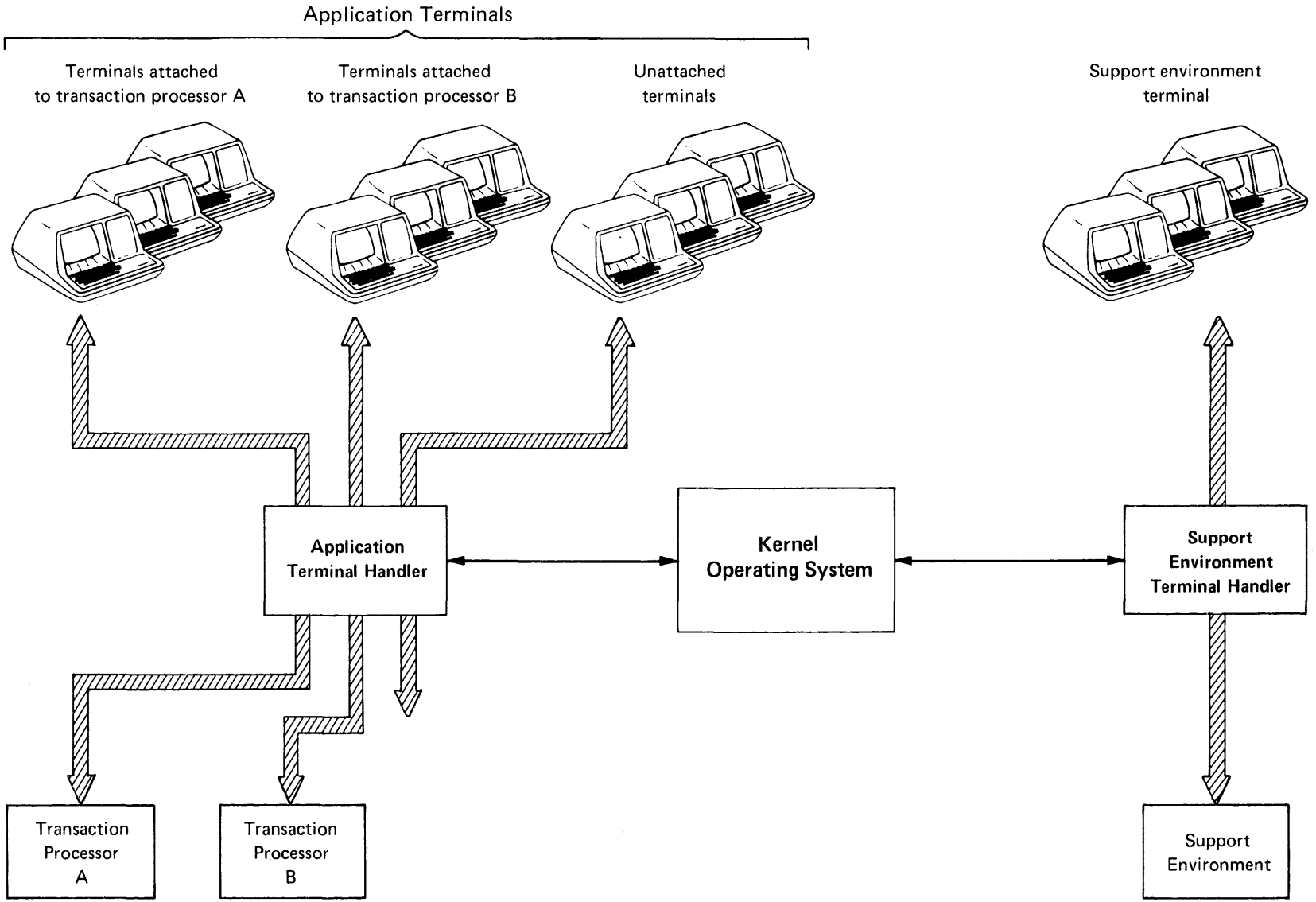


Figure 3-5 Application Terminals & Support Terminals

Stations are supported by system software within the transaction processor, and each type of station requires special support. For example, messages arriving at Terminal Stations must be interpreted according to a specific forms definition before they can be passed to the terminal. Messages arriving at a TST station require the TST to be loaded into memory and executed. Other types of stations require different processing support.

- **A Routing List** is the list of stations to which an exchange message is addressed. An exchange message can have several addresses, and it visits each station on its Routing List in turn. Usually, these stations are TST stations.

  An exchange message is the only message that can have a Routing List. All other messages have a single address.

  Routing lists for the exchange message of a transaction are specified in the definition of the transaction. By consulting this definition, the transaction processor knows where to send the exchange message.

  In addition, the Routing List for any exchange message can be modified by any TST that processes that message. This applies only to a single exchange message and does not affect other similar exchange messages belonging to other transaction instances. This facility allows customized exchange message routing under program control.

- **A Form** is a data structure presented through an application terminal that is an interface between the user at that terminal and the transaction processor.

  In other words, a Form is a fill-in-the-blanks display. If a user wishes to send data to the system, he fills in fields on a form and sends the form to the system. If the system wishes to send data to the user, it fills in fields on a form and sends the form to the user's Application Terminal. One Form can be used to send data in both directions with careful design – for instance, a user can ask a question using certain fields on a form and the system can answer using other fields on the same form.

- **A Form Definition** tells a transaction processor how to present and handle each different form. Besides describing the position and size of each field on the form, a Form Definition also:
  - Describes the attributes of each field, including its appearance, behavior, and the characters that can be entered
  - Describes the way that the data entered by the user is formatted into an exchange message for routing to TST stations
  - Describes the terminal function keys the user may use after filling out the form
  - Describes the ways that the form can be modified, such as in response to a user data entry error

  Every exchange normally has its own Forms Definition. These definitions are identified by unique names and are kept in a special file within the transaction processor. The definitions are consulted each time data arrives from an Application Terminal and each time a message arrives at a terminal station.

- **A Transaction Definition** sets the structure of each of the transactions a transaction processor can handle. The Transaction Definition specifies parameters such as:
  - How many exchanges comprise the transaction
  - What form definition is used with each exchange
  - Where each exchange message is routed
  - In what sequence exchanges are executed
  - Whether specialized reliability or recovery procedures, such as exchange recovery, are needed

In the chapters ahead, you will see how a terminal user or a TST programmer can alter the course of a transaction – that is, change the sequence of events from that specified in the transaction definition. As you study those techniques, remember that the structure of the transaction always *starts* with the Transaction Definition; the techniques you read about can only *alter* a pre-existing transaction structure within certain prescribed limits.

NOTE

In most cases, these structure modification techniques are not necessary and should not be used.

- **A Transaction Workspace** is a scratch area associated with each transaction instance. Two terminals executing the same transaction have different Transaction Workspaces, as would two executions of the same transaction from the same terminal. The Transaction Workspace can be accessed by every TST that becomes involved in the processing of the corresponding transaction instance.

  The Transaction Workspace is different from an exchange message: The exchange message is regenerated for each new exchange, using fresh input from the user; the Transaction Workspace is available throughout the life of a Transaction Instance. Its data content remains unchanged unless explicitly modified by a TST. This is a valuable method of transferring information between the TSTs that process the Transaction Instance, both between TSTs in the same exchange and between TSTs in different exchanges.

3.6 A TRANSACTION PROCESSOR AT WORK

Now let's return to the sample transaction introduced in Section 3.3: the Change Customer transaction that allows a terminal user to update a record in the Customer File.

3.6.1 The Components Involved

These transaction processor components play a part in the execution of the Change Customer transaction:

- **Terminals and Terminal Stations.** Figure 3-6 shows three terminals, each with its own terminal station. The number of terminals is not important. As you follow this example, though, remember that *all* the terminals are likely to be active in a system, not just one terminal. Imagine what each step of the transaction would be like with transactions from other terminals underway.
- **Transaction Step Tasks and Their Stations.** Each of the programs shown in Figure 3-6 is, of course, a TST. Every TST has its own station. The functions of the three TSTs are:

  RDCUST – To read a customer record and pass it to the user's terminal
  VALIDC – To validate the new data submitted by the user
  REWRIT – To rewrite the updated customer record into the customer file
- **Forms and Form Definitions.** The Change Customer transaction has two exchanges and requires two forms. The functions of the two forms are:

  CHCUS1 – To let the user specify the customer whose record is to be read for possible update
  CHCUS2 – To display the old customer data, allow the user to enter new data, and return the new data to the transaction processor.

The transaction definition is stored in the transaction processor's Transaction Definition File, with the definitions of other transactions handled by the transaction processor.

There is, in addition, one component of the transaction processor that is not shown in Figure 3-6: the set of system software modules that manage many of the transaction processor's internal operations. Master copies of these system software modules are supplied with each TRAX system, and the generation of a transaction processor includes making customized copies of each of them. It is these modules that interpret forms definitions, transaction definitions, and the other specifications that guide the operation of a transaction processor.

Communication between the terminals and TSTs require exchange messages and response messages. These messages, shown in Figure 3-6 as arrows flowing between stations, are:

- **Exchange Messages** (shown as solid arrows) contain data derived from the user's input and are generated according to specifications contained in the corresponding form definition. The messages are routed to one or more TST stations for processing.
- **Response Messages** (shown as dashed arrows) originate with TSTs and are directed to the terminal that sent the exchange message.

3.6.2 The Sequence of Events

The Change Customer transaction causes the following sequence of events:

1. **The Transaction Begins.** The user selects this transaction from a menu or list of transactions. This process is described in Chapter 9.
2. **Displaying the First Exchange's Form.** The first exchange begins with the display of a form. The form is specified in the transaction definition, and the form definition itself can be found in the form definition file.
3. **First User Input.** The user responds to this form by entering a customer identification number and pressing the ENTER key.
4. **Exchange Message Constructed.** System software within the transaction processor takes this input and formats it into an exchange message according to the specifications in the form definition.
5. **Exchange Message Routing.** The transaction processor consults the transaction definition to find which TST (or TSTs) the exchange message should be routed to. In this case, the message is routed only to the RDCUST TST.
6. **Processing by RDCUST TST.** The exchange message waits at the TST station until a copy of the TST is available to process it. The TST then reads the specified customer record from the file.
7. **Generation of Response Message.** Having read the record, the TST generates and sends a response message containing the data from the customer record. The TST selects the proper response message so that the transaction processor moves to the next exchange of the transaction.
8. **The Second Exchange.** The second exchange is entered because the response message from the previous exchange directed that this occur. The second exchange displays the old customer data and asks the user to edit it.
9. **Displaying the Second Exchange's Form.** The process by which this form is displayed makes use of the data returned in the first exchange response message. (That is how the old customer data is transferred from the file to the terminal screen.) The form definition specifies the manner in which the data from the previous response message is used.

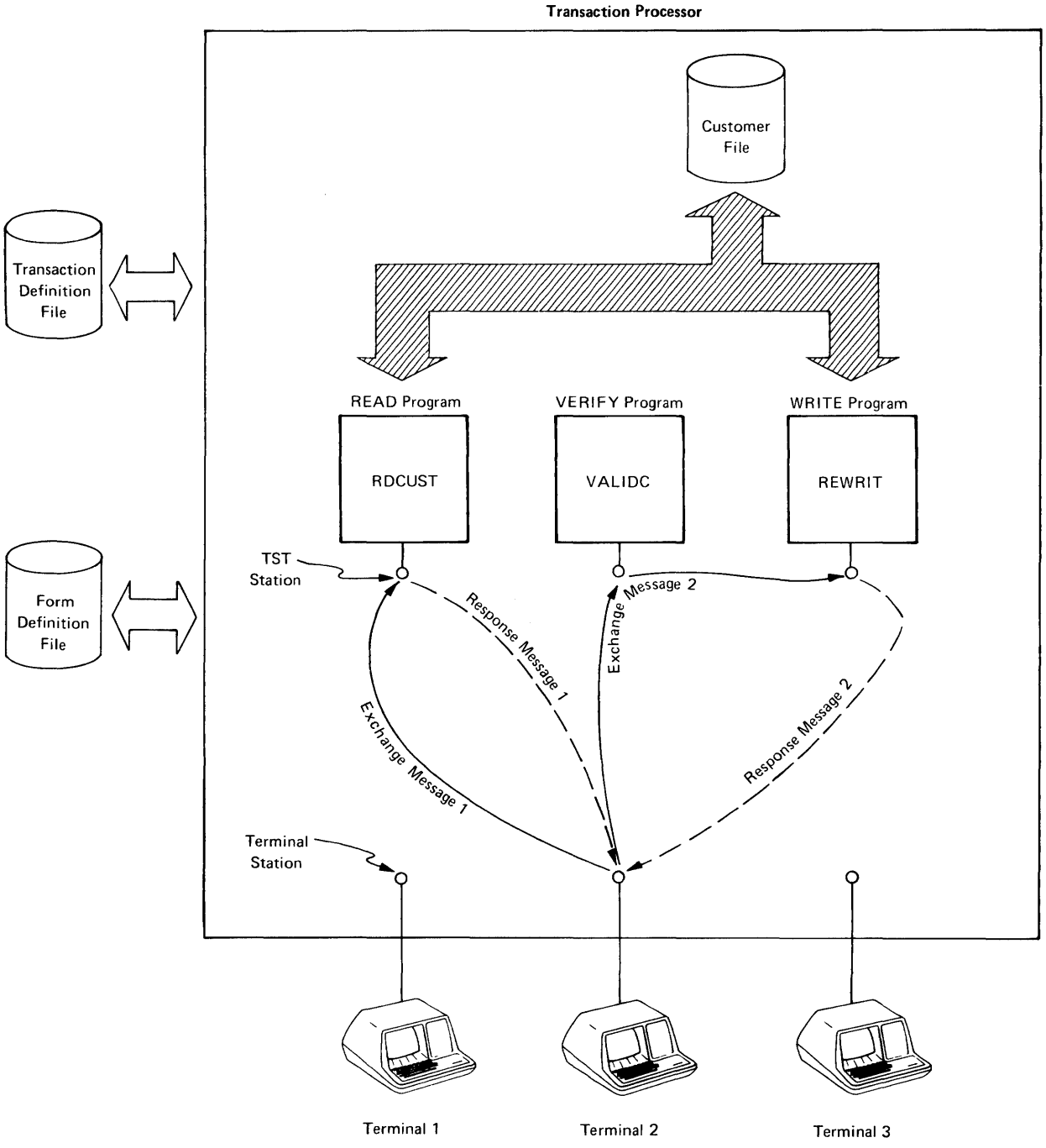


Figure 3-6 A Transaction Processor at Work

10. **Second User Input.** The user's input in the second exchange is not necessarily a data entry operation; the existing data might be edited slightly – or not at all.
11. **Exchange Message Constructed.** When the user presses the ENTER key, the data is formatted into an exchange message according to specifications in the form definition. *All* customer data is included in this exchange message, even if it has not been changed by the user.
12. **Exchange Message Routing.** The transaction processor again checks the transaction definition to see where the exchange message should be routed. In the second exchange, the message is routed sequentially to two TSTs. First, the VALIDC TST checks the data, and then the REWRIT TST writes the updated record in the file.
13. **Processing by VALIDC TST.** As in the first exchange, the exchange message may have to wait at the TST station. Eventually, it is processed. When this TST terminates, the transaction processor forwards the exchange message to the second station on the routing list.
14. **Processing by REWRIT TST.** Again, the exchange message may have to wait at the corresponding TST station. After writing the customer data into the customer file, this TST generates and sends a response message.
15. **Generation of Response Message.** This response message contains no data but informs the user that the update has been accomplished.
16. **Response Message Arrives at Terminal Station.** When the response message arrives at the terminal station, it causes a confirmation message (previously encoded as part of the form definition) to be displayed. It also unlocks the AFFIRM key on the terminal.
17. **User Presses AFFIRM Key.** As soon as he sees the confirmation message, the user presses the AFFIRM key, ending the transaction instance. The terminal screen returns to the first form of the transaction, awaiting another execution of the same transaction.
18. **User Returns to Transaction Selection Screen.** If he does not wish to execute another change transaction, the user presses the CLOSE key to return to the transaction menu.

CHAPTER 4
TRANSACTION PROCESSING PATHS AND THEIR CONTROL

The principal difference between TRAX and other systems is that TRAX does not have continuously active application programs to guide each user's on-line processing. In a TRAX transaction processor, each user's interactive processing is influenced by many system components. These system components interact to determine the way any transaction instance proceeds.

This chapter examines these influences and the way they interact.

4.1 FUNDAMENTALS: STATIONS AND MESSAGES

At the lowest level, the sequence of processing applied to any transaction instance depends on the content of a series of messages, the stations where these messages are directed, and what happens at each station when the message is processed.

4.1.1 Stations

A station in a transaction processor holds messages until they can be processed.

Stations are strictly first-in, first-out facilities. The message that arrives first will be processed first.

There are several different kinds of stations; they are distinguished by the kind of message processing facility they serve. The different kinds of stations are:

- **Terminal Stations.** These stations receive messages for their particular application terminals.
- **TST Stations.** These stations receive messages addressed to a particular TST.
- **Batch Submit and Batch Slave Stations.** These stations handle messages destined for, or coming from, the support environment.
- **Link Master and Link Slave Stations.** These stations handle messages destined for, or coming from, other transaction processors – perhaps on other computer systems.
- **Mailbox Stations.** These stations store messages for TSTs until they are called.

Each station serves a message processing facility. Only TST stations serve application modules. Others serve system software modules that manage the transaction processor. For instance, terminal stations serve the system software modules that interpret forms definitions and communicate with application terminals. TST stations, of course, serve TSTs.

As you work with transaction processors, you may find it convenient to mentally combine stations with their respective message processing facilities and use the term "station" to apply to the combination. This simplifies the discussion. For example, you could say:

> "Then, the exchange message is sent to the TST station XYZ, which looks up the part number in the part file."

Of course, this statement is not strictly true; the station's purpose is to receive, store, and forward the message; it is the TST itself that eventually looks up the part number in the part file. This is a subtle distinction, however, and usually unimportant. Accordingly, the distinction will only be maintained in this manual where it is significant. Otherwise, you will see the term "station" applied loosely to both the station proper as well as the message processing facility that it serves.

An important step in creating a transaction processor is specifying its set of stations. Once they are specified, a TRAX utility program called STADEF is used to install the list of stations in the transaction processor. Determining the proper set of stations is discussed in Part Two of this manual; the procedures used with the STADEF utility program are described in the *TRAX Application Programmer's Guide.*

The following sections explain some of the special characteristics of different stations.

4.1.1.1 Terminal Stations – Terminal stations receive messages directed to their terminals by TSTs. For interactive terminals, response messages alter the screen display. For output-only terminals, the messages contain data to be printed.

Messages arriving at a terminal are always interpreted according to a form definition. The form definition guides the terminal-handling software as it processes the message.

The parameters needed to define a terminal station are discussed in Section 16.2.

4.1.1.2 TST Stations – TST stations receive exchange messages and wait for their TSTs to process them. Most exchange messages come from terminal stations, but other transaction processor components can originate transaction instances and, consequently, generate exchange messages.

Earlier, you read that stations operate on a first-in, first-out basis. It is important to realize that this FIFO rule applies only to the TST station, and not to the processing of the messages by the TST proper. Each TST station can serve multiple copies of a TST, and different copies may process messages with varying speed.

The number of TST copies that can be active at once is specified when the station is defined. When a TST finishes processing an exchange message, the TST is given another exchange message from its station. If no messages are waiting at the station, the TST copy is deactivated.

While they are processing exchange messages, TSTs can also generate messages. These messages are attributed to the corresponding TST station. For example, when a TST spawns a new transaction instance that transaction instance and its exchange message are attributed to the TST *station.* The TST station name is returned when some other TST issues a system call asking for the source of the spawned exchange message.

The parameters required to define a TST station are discussed in Section 16.2. You can find more about TSTs and how to program them in:

- Chapter 19 of this manual
- The *TRAX Application Programmer's Guide*

4.1.1.3 Batch Submit and Batch Slave Stations – You will only use these stations if some of your transactions need to communicate with the support environment. For information about these stations, consult Chapter 11.

4.1.1.4 Link Master and Link Slave Stations – You will only use these stations if some of your transactions need to communicate with other transaction processors – either those running on the same computer or on remote computers. For information about these stations, consult Chapter 11.

4.1.1.5 Mailbox Stations – Mailbox stations hold messages that are deposited and retrieved by TSTs. There is no restriction on which TSTs can deposit or retrieve messages. Messages can be sent to mailbox stations across exchanges of a transaction instance, between two transaction instances of the same type, or between two unrelated transactions.

Like other stations, mailbox stations operate on a first-in, first-out basis. This means that a transaction instance that deposits a message in a mailbox station and then retrieves a message from that station may not get its own message. TSTs cannot wait for a message to arrive at a mailbox station. If a message is not present, TSTs receive an error code and they continue execution. TSTs can determine the number of messages in a mailbox station before asking for a message.

The contents of a mailbox station is stored on disk and is not affected by system shutdown. The contents of a mailbox station is therefore available when the system begins operation again.

The parameters required to define a mailbox station are discussed in Section 16.2.

4.1.2 Messages

Messages are the data structures through which components of a transaction processor communicate with each other. So far, you have been introduced to two kinds of messages: exchange messages and response messages. There are four different kinds of messages used by transaction processors:

- Exchange Messages
- Response Messages

There are six varieties of response messages:

  PRCEED
  STPRPT
  TRNSFR
  CLSTRN
  REPLY
  ABORT

- Report Messages
- Mailbox Messages

4.1.2.1 Exchange Messages – An exchange message is usually originated by an interactive application terminal. It contains data entered by the terminal user, and it is directed to a series of TST stations (or possibly other stations) that must process the data it contains.

An exchange message is the only kind of message that can have a routing list – that is, a list of several destination stations. Other messages have only one destination. Exchange messages are routed sequentially to the designated destination stations; when a station has processed the message, it is sent to the next station on the routing list.

The routing list of an exchange message has an influence on the sequence of processing steps applied to a user's input. The user input is only seen by those stations listed in the exchange message routing list, and the processing done by the various stations must be applied in the correct order. This way, the routing list acts as the top level of control for transaction processing. Essentially, it prescribes a series of subroutines that are called to process the contents of the message.

The initial routing list for each exchange message comes from the appropriate transaction definition. The definition gives a different routing list for each exchange message in each transaction.

But once the exchange message is constructed and dispatched, any TST that processes the message can change the routing list. These changes, made with TRAX system calls, do not affect the original routing list in the transaction definition. They affect only the routing list of the exchange message being processed at that moment. The changes can be additions, deletions, or complete erasures of the routing list.

In addition, a TST can change the content as well as the routing of an exchange message. The new content will be seen by all "downstream" stations on the routing list – that is, those stations to which the message has not yet been routed. This technique is necessary for any of the batch or link stations, because they require a special exchange message format that is probably best generated by a TST "upstream" from the batch or link station. (For details on batch and link stations, see Chapter 11.) In most other cases, avoid designing TSTs that change the content of exchange messages, transaction processors using such designs are difficult to debug. The transaction workspace is meant to be used as a scratch area, and you should use it for that purpose in preference to the exchange message.

An exchange message survives until it has been processed by the last station on its routing list. When the final station has finished with it, the exchange message is purged from the transaction processor.

Remember that only one exchange message can exist for a transaction instance at one time. This restriction becomes important in several design instances.

4.1.2.2 Response Messages – A response message is generated by a TST and directed back to the application terminal station that issued the exchange message that the TST is processing.

Remember that a TST is only activated to process an exchange message; this means that any active TST must be working on an exchange message. This exchange message is associated with a transaction instance and the terminal station that originated the transaction instance. This is the station to which the TST's response message is always directed. (In fact, a TST does not even direct response messages to specific terminals – the transaction processor automatically directs each response message to the appropriate terminal station.)

The response message tells the terminal user the outcome of the exchange message processing and directs the transaction processor to a new exchange. The precise exchange chosen depends on the type of response message sent by the TST as well as certain details of the transaction definition. The six response messages, the relevant transaction definition parameters, and their combined effects are discussed in Section 4.4

Response messages, unlike exchange messages, have a single destination station – the terminal station that issued the associated exchange message.

The transaction processor expects (with one exception) one response message for each exchange message. So, if an exchange message is processed by a series of TSTs, only one of these TSTs should issue a response message. Additional response messages will cause the transaction instance to be aborted. (The exception occurs with the NOWAIT option described later in Section 4.2.4. When this option is used, *no* response messages are allowed.)

What happens if the TST that issues the response message is not the last station on the exchange message routing list? This situation is shown in Figure 4-1. Here, an exchange message is routed to three TST stations. The second of the three sends the response message, leaving one station to process the exchange message. The following things happen:

- Sending the response message does not terminate TST2. This TST can continue to execute.
- The response message is generated and dispatched as soon as TST2 issues the appropriate system call. For a time, both the exchange message and the response message exist.
- Sending the response message does not erase TST3 from the exchange message routing list. The exchange message continues to TST3 as soon as TST2 terminates.
- In parallel with the progress of the exchange message, the response message arrives at the proper terminal station. The arriving message triggers a display action at the terminal, allowing the user to see a new screen display. The message is then discarded.
- The user proceeds to enter data to the new form as if the processing for the first exchange had been completed.
- Meanwhile, the exchange message is finally processed by TST3 and discarded.

Interleaving user conversation and exchange message processing in this way is a form of parallel processing and can result in complex transaction designs. Most transaction processing situations can be handled in a straightforward way by requiring the last TST for an exchange message to send the response message before it terminates. Alternatively, if you want an earlier TST to send a response message (for instance, one containing an error message), be sure that your TST erases all "downstream" TSTs from the exchange message routing list. The issuing TST therefore becomes the last in the routing list, and you will again have a simple nonoverlapped processing arrangement. Do not attempt overlapped processing designs until you are familiar with transaction processors and their operation.

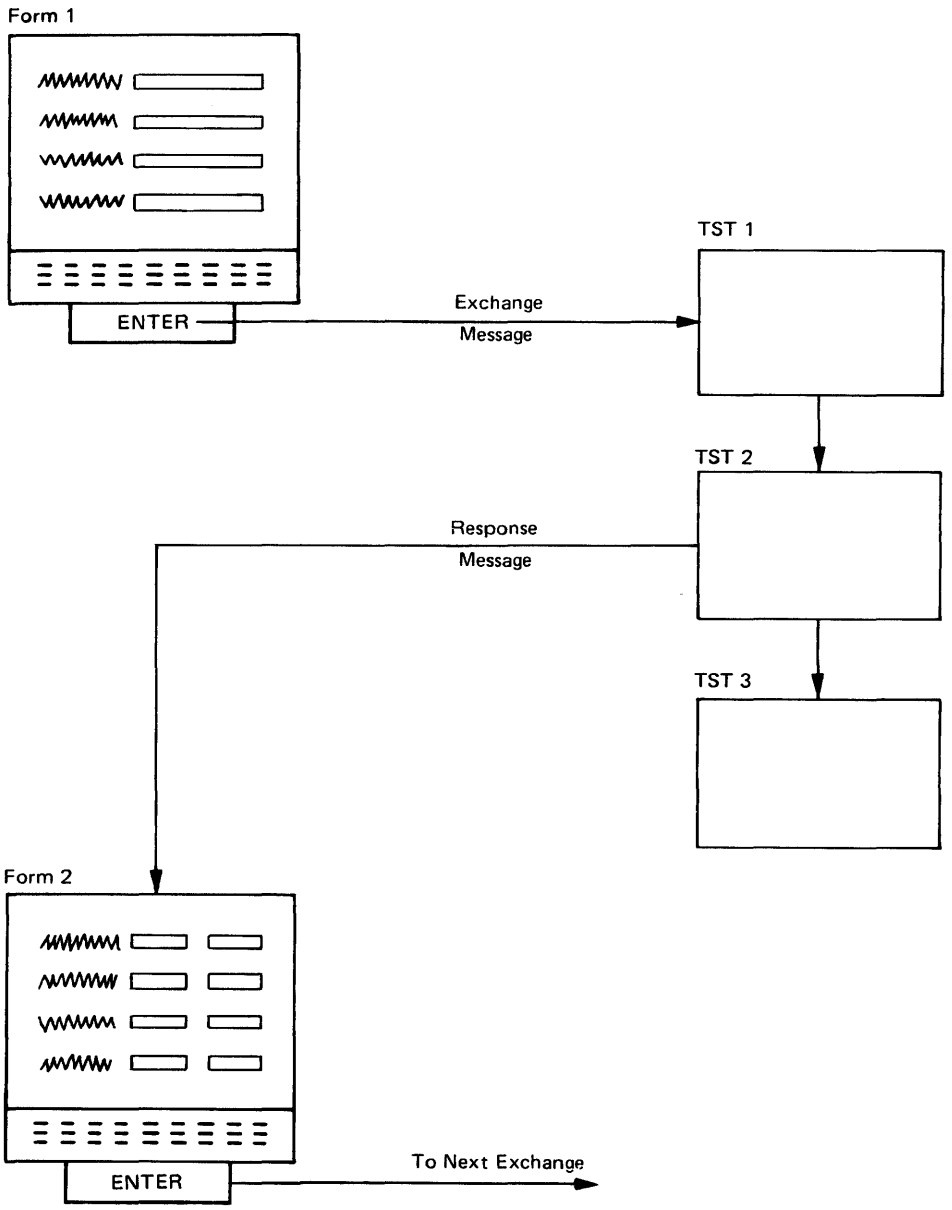


Figure 4-1 Relationship Between Response Messages and Exchange Messages

4.1.2.3 Report Messages – Report messages send data to an output-only application terminal where the data is printed. Report messages are always sent by TSTs and are always directed to a terminal station associated with an output-only terminal. (There is no way to print data at an interactive application terminal, if that terminal did not initiate the transaction instance in question. There is no way to involve more than one interactive terminal in any given transaction instance.)

Remember that everything printed, displayed, or entered at an application terminal is controlled by a form definition. This is true for output-only terminals, and each page printed at an output-only terminal is printed in the format prescribed in a form definition.

The report message consists of two parts:

1. It specifies the form that controls the format of the printed output.
2. It contains the data to be inserted on the form during the printing process.

Constant data can be included as part of the form definition; only the variable data need be included in the report message itself.

Each report message must contain data to fill a page of output. Several report messages cannot be combined to fill a single form.

A single output-only terminal station can receive report messages from a variety of TSTs. Pages from the output-only terminal will be interleaved without regard for transaction instance, and your transaction processor design must allow for this.

4.1.2.4 Mailbox Messages – Mailbox messages allow communication between two transaction instances or a method of data batching for later processing.

Mailbox messages are generated by TSTs and can only be sent to mailbox stations. Mailbox messages may have any data content. They wait at mailbox stations on a strict first-in, first-out basis until another TST asks for them.

4.2 THE TRANSACTION DEFINITION

Transaction definitions associate exchanges, forms, routing lists, and other transaction parameters to give an overall framework to a transaction. Each type of transaction that a transaction processor handles needs a definition.

Parameters specified in a transaction definition are not binding. Many parameters depend not only on the transaction definition, but also on the message content or particular terminal function keys. It is the interaction of all these factors that determines how any transaction instance proceeds. The transaction definition provides a framework in which the other elements can be applied.

A transaction definition divides a transaction into exchanges. For each exchange, it specifies:

- A name for the exchange
- The form to be used
- A routing list for the exchange message
- Optional selection of the NOWAIT option (explained next)
- Optional selection of the REPEAT option (explained next)
- What will happen when the exchange is completed
- A time limit for the exchange

Transaction definitions for a transaction processor are kept in one file, called the *transaction definition file.* Definitions are entered into this file with a utility program called TRADEF, which executes in the support environment of the TRAX system. The TRADEF utility program serves as an editing and listing facility for transaction definitions as well as a means of entering them.

4.2.1 Exchange Label

The exchange label is a short name (up to six characters) that identifies the exchange within the transaction definition. A TST can use this label, for instance, when it wants the transaction processor to transfer to a specific exchange without regard to the other transaction definition parameters.

4.2.2 Form Name

This transaction definition parameter specifies the form to be displayed during the exchange. Most transactions are initiated from interactive terminals. For these transactions, the content of the exchange messages are derived from user input, and this input can only occur under the control of a form.

Only one form can be displayed during any exchange. The only modifications that are permitted to the basic form display are those defined within the form definition itself.

Transactions initiated by other than interactive terminals do not need forms. These alternative transaction initiation techniques are explained in Chapter 9.

4.2.3 Routing List

The transaction definition contains a routing list for each exchange. This routing list is assigned to the exchange message when it is generated.

NOTE

The *content* of the exchange message is specified by the *form definition* used for the exchange; the *routing* of the exchange message is specified by the *transaction definition.*

Once an exchange message is generated, addressed, and sent, its routing list may be changed by any TST that processes it. Such a change affects only that particular exchange message and does not affect the master routing list in the transaction definition.

Most transaction processing requirements can be met easily without having TSTs change the routing lists of exchange messages. A good design allows TSTs only to *remove* entries from the routing lists and then possibly only *all entries at once.* That is, you should be able to design most transaction processors so that the only changes made to routing lists are deletions of remaining destinations – a cancellation of further routing. More complex or varied changes are likely to result in a transaction processor whose structure and operation is more difficult to understand.

4.2.4 The NOWAIT Option

A transaction processor normally waits for a response message in answer to each exchange message. Until the response message is received, the user's terminal cannot proceed to the next conversational step – either to a new exchange or to another cycle through the current exchange.

The NOWAIT option tells the transaction processor that a response message is not coming for an exchange message. The transaction processor therefore proceeds to the next conversational step as soon as the exchange message is constructed and dispatched, and the TSTs that process the exchange message cannot send a response message.

This option is not often used. It is included in the transaction definition for exchanges where no response message is necessary. It might be profitably used, for example, for "blind data entry" – that is, data entry to a file without any edit checks or error messages. The absence of edit checks and error messages makes response messages unnecessary; and, in such instances, the NOWAIT option can be used to improve transaction response times.

4.2.5 The REPEAT Option

The REPEAT option, when used, indicates that an exchange should be repeated. This loop is repeated until an appropriate terminal function key or response message instructs otherwise. (Terminal function keys and specific response messages are described in Sections 4.3 and 4.4, respectively.)

The REPEAT option causes each repeated execution of the exchange to begin with a fresh copy of the exchange form. All modifications that occurred during the previous execution of the exchange are removed, along with any user-entered data.

This option might be used in a transaction that enters orders to an order file. Such a transaction might consist of three exchanges:

- A preliminary exchange that would collect general order information: customer name, address, ship-to address, and so forth.
- An exchange that would collect individual order lines – that is, each execution of the exchange would allow the entry of one order line: quantity, stock number, price, description, and so forth.
- A final exchange that would verify the order, calculate the total price, and accept payment.

The middle exchange in this transaction would probably use the REPEAT option. It should be repeated until the terminal operator tells the system that all order lines have been entered. Ways in which a terminal operator or a TST could signal an end to the repeated exchange are discussed in Section 4.3.2.

4.2.6 Subsequent Action

The transaction processor needs to know what to do when an exchange (including repeat cycles, if any) has been completed. The parameter that provides this information for each exchange is called the *subsequent action* parameter. Three choices are possible:

- **NEXT.** This indicates that the next exchange in the transaction definition should begin.
- **FIRST.** This indicates that the current transaction instance is to be terminated; and the form from the first exchange of the transaction then is to be displayed so that the terminal user can begin another transaction of the same type.
- **INITIAL.** This indicates that the current transaction instance is to be terminated. After this, the transaction selection menu (or other initial operating mode defined for the terminal) is to be displayed. (Transaction selection forms and initial operating modes refer to what an application terminal displays when it is idle; these terms are discussed in detail in Section 9.1.)

In the order entry example in Section 4.2.5, the subsequent action parameter would probably be set as follows for each exchange:

- In the first exchange, the parameter is set to NEXT.
- Similarly, in the second exchange, the parameter is set to NEXT. This parameter would only come into play, however, when the REPEAT-option loop had been terminated.
- In the third exchange, the parameter could be set either to FIRST or INITIAL. The former would be appropriate if the user were likely to begin another order; the latter if the user were likely to choose another transaction.

4.2.7 Exchange Time Limit

A time limit (in minutes) is provided for each exchange so that the transaction cannot "hang" indefinitely. There are several things that can cause a transaction to "hang" or suspend itself in an exchange – for instance, the lack of a response message where one is needed. There can also be run-time problems, such as an extreme delay during communication with another transaction processor.

The time specified in the time limit starts when the user presses a user function key. If the time limit expires before a response message has been received, the transaction is aborted.

4.2.8 General Transaction Parameters

Besides the parameters listed before, which apply to each exchange, the transaction definition also includes some parameters that apply to the transaction as a whole:

- **Transaction Name.** Each transaction is identified with a short name (up to six characters).
- **Exchange Recovery.** Exchange recovery is a technique for reprocessing exchanges after certain error conditions. The transaction definition specifies whether the transaction supports exchange recovery. Exchange recovery is fully described in Section 10.2.1.
- **Message Logging.** Exchange messages and other messages sent during the transaction can be logged to the system journal device if desired.
- **Data Structure Sizes.** The size of the largest exchange message, the size of the transaction workspace, and the size of the system workspace must all be specified in the transaction definition. (The system workspace is explained in Chapter 7.)

4.3 THE EFFECTS OF TERMINAL FUNCTION KEYS

Video display application terminals have ten function keys:

System Function Keys
- AFFIRM
- STOP REPEAT
- CLOSE
- ABORT

User Function Keys
- ENTER
- DOT (.)
- 0
- 1
- 2
- 3

These keys cannot be used either to enter or edit data on a form. Instead, they are used to initiate some action once all data entry and editing have been completed.

There are two groups of function keys:

- **System Function Keys** *do not* cause an exchange message to be constructed. A terminal user avoids exchange processing by pressing one of these keys. Depending on the transaction definition in use and the exact key pressed, these keys can either send the user to another exchange or terminate the transaction instance.
- **User Function Keys** *do* cause an exchange message to be constructed from the user input. The exchange message is then routed to processing stations (usually TSTs) by the routing list specified in the transaction definition. All the keys in this group have an identical function; they only differ by the legend on the key cap. This legend (or a substitute legend, if desired) can be included in the exchange message, so the TSTs can determine which key was struck.

The terminal function keys in both groups (except the ABORT function key) can be enabled and disabled in the course of a transaction. This enabling and disabling is under the control of form definitions. The form definition specifies the keys to be enabled when the form is first displayed and specifies how keys are to be enabled and disabled for each reply definition. (Reply definitions are explained in Section 4.4.5.)

The ABORT key is always enabled.

Function keys can be pressed only when the transaction processor is expecting input from the user's terminal. For example, the user *could* press a function key while he is completing a form; he *could not* press a function key after he has pressed ENTER and his exchange message is being processed. This is also true for the ABORT key – it cannot abort exchange message processing that is already in progress.

Figure 4-2 summarizes the effect of each terminal function key. Note that all the user function keys are treated as a single type of key, because they each have the same effect on the flow of a transaction.

4.3.1 The AFFIRM Key

The AFFIRM key interacts with two parameters in the transaction definition: REPEAT/ NOREPEAT and subsequent action.

- When the AFFIRM key is pressed, the transaction processor first tests the REPEAT/ NOREPEAT parameter for the current exchange. If this parameter is set to REPEAT, the terminal screen is erased and the exchange is repeated. If set to NOREPEAT, the transaction processor tests the second parameter, subsequent action.
- The subsequent action parameter can have one of three values:

  NEXT. If this value is found, the transaction processor moves the user to the next exchange in the transaction definition. If there is no "next exchange," the transaction processor acts as if the INITIAL value had been chosen.

  FIRST. If this value is found, the transaction processor ends the current transaction instance. Then the user is brought back to the first exchange of the same transaction for another execution.

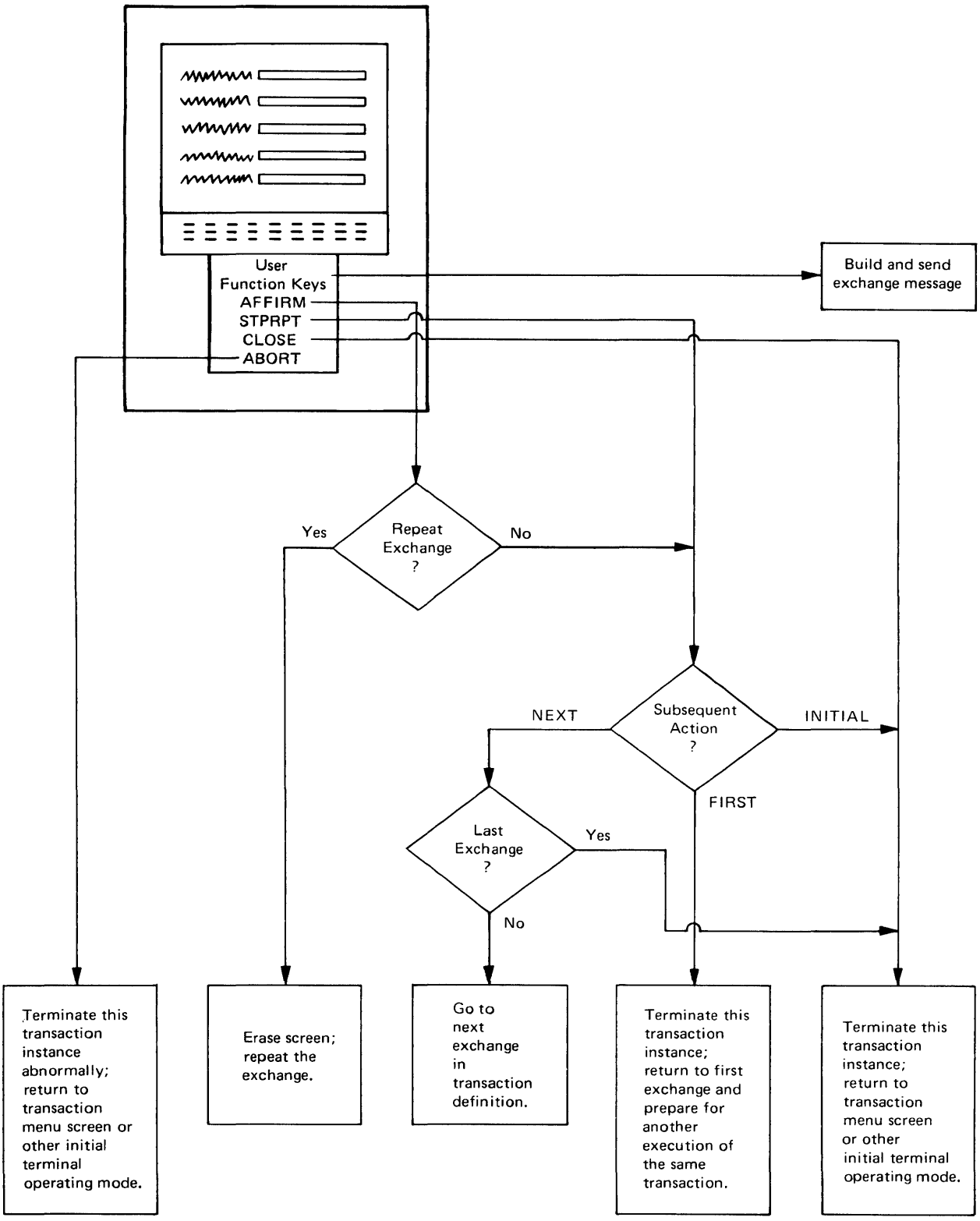


Figure 4-2 Effects of Terminal Function Keys

- **INITIAL.** Like FIRST, this value causes the transaction to be terminated. But the user is taken to the terminal's transaction selection menu or other initial operating mode. The user can then select a new transaction.

4.3.2 The STOP REPEAT Key

The STOP REPEAT key is similar to the AFFIRM key. The difference is that the STOP REPEAT key never causes the REPEAT/NOREPEAT parameter in the transaction definition to be tested. The exchange is always treated as if that parameter were set to NOREPEAT.

The STOP REPEAT key uses the subsequent action parameter as the AFFIRM key does.

4.3.3 The CLOSE Key

The CLOSE key terminates the transaction instance. The user is returned to the terminal's transaction selection menu or other initial operating mode. The user can then select another transaction.

4.3.4 The ABORT Key

The ABORT key terminates the transaction instance with an "abnormal" status. This means that certain statistics are recorded, and staged file updates that are pending for the transaction instance are not done. The user is returned to the terminal's transaction selection menu or other initial operating mode. The user can then select a new transaction.

The ABORT key is unique in that it is the only function key that cannot be disabled. The user may press it at any time that his keyboard is unlocked.

4.3.5 The User Function Keys

The user function keys (ENTER, DOT, 0, 1, 2, and 3) have an identical effect on the flow of the transaction: they cause an exchange message to be constructed from the form, and the exchange message is routed to the appropriate list of stations.

User function keys are enabled and disabled in the same way as system function keys. Usually, the ENTER key is left enabled and the others are disabled. Additional user function keys can be enabled if a TST must be able to distinguish between two or more user data entry actions.

User function keys indicate a user's choice between several alternatives. Several user function keys can be enabled, and the user chooses one by pressing it. If the form designer has included the identifier of the function key in the exchange message, the TSTs that process the exchange message can tell which key was pressed.

The user must press the Shift key when he presses a user function key. (The user function keys are part of the numeric keypad when the Shift key is not pressed.) The ENTER key is an exception. The ENTER key does not need a Shift key.

4.4 THE RESPONSE MESSAGES

Response messages are TST answers to exchange messages. During normal transaction processing, each exchange message is answered by one response message. The response message is sent by one of the TSTs that processes the exchange message, and it is sent to the terminal station that initiated the exchange message.

Response messages serve two purposes:

- They can contain data that the TSTs have generated during the processing of the exchange message. For example, if an exchange message contained a customer identification number, the corresponding response message might contain that customer's record in a Customer file.
- They can give instructions to the transaction processor to move to a specific exchange.

It is the second function – controlling movement between exchanges – that we cover next.

In most cases, response messages alone do not determine which exchange is executed next. Like function keys, they interact with two parameters in the transaction definition. The final choice of an exchange depends on both the specific response message and the two transaction definition parameters.

There are six different kinds of response messages.

PRCEED
STPRPT
TRNSFR
CLSTRN
REPLY
ABORT

These response messages can be distinguished by their instruction to the transaction processor concerning the next exchange to be executed.

Each message can include data that the TST wishes to send to the terminal. The data included in the message does not alter the *effect* of the message; that is, the data cannot affect which exchange is executed next. The data can only be used to construct the destination exchange's form. The *effect* of the message is determined by the *kind* of message the TST chooses to send.

4.4.1 The PRCEED Message

The PRCEED message is the kind of response message most commonly used by TSTs. When a TST sends this message, the exchange executed next depends on two transaction definition parameters: REPEAT/NOREPEAT and subsequent action.

- The transaction processor first checks the REPEAT/NOREPEAT parameter for the current exchange. If the REPEAT parameter is set, the terminal screen is erased and the current exchange is executed again. If the NOREPEAT parameter is set, the second transaction definition parameter is checked.
- The second parameter can have one of three values:

  If the parameter is set to NEXT, the next exchange in the transaction definition is executed. If there is no "next exchange", the transaction processor acts as if the INITIAL value was present instead.

  If the parameter is set to FIRST, the current transaction instance is terminated. Then, the first exchange of the same transaction definition is begun as a new transaction instance.

- If the parameter is set to INITIAL, the current transaction instance is terminated. But, instead of preparing to execute the same transaction again, the transaction processor returns the terminal to its initial operating mode. This will probably be a transaction selection menu, and the user can choose a new transaction.

If a PRCEED response message contains application data, that data is used with the newly selected exchange form definition to build the form.

4.4.2 The STPRPT Message

The STPRPT response message is like the STOP REPEAT terminal function key: it ignores the REPEAT/NOREPEAT option in the transaction definition. In other respects, the STPRPT message is similar to the PRCEED message.

4.4.3 The TRNSFR Message

The TRNSFR response message allows a TST to override the transaction definition parameters and select the exchange to be executed next. This type of response message is useful when a TST must choose among two or more possible successor exchanges to the current exchange. The parameters in the transaction definition allow only two alternatives upon completion of each exchange – termination of the current transaction instance, or the execution of the *next* exchange in the transaction definition. If you need a transaction that can take several alternative paths out of an exchange, you must use the TRNSFR message in your transaction design to overcome this limitation.

When a TST issues a system call to send a TRNSFR response message, it must specify an exchange name. This name, which corresponds to the short name (or *label*) given to an exchange in the transaction definition, identifies the exchange to be executed. The transaction instance transfers execution to this exchange, no matter where the exchange may be in the transaction definition.

If a TST chooses an exchange name that happens to be the first exchange in the transaction definition, a new transaction instance is *not* begun. The same transaction instance continues, just as if the chosen exchange were elsewhere in the transaction definition.

NOTE

This is a different effect from the interaction between a PRCEED message and a FIRST parameter in the transaction definition, where a new transaction instance *would* begin.

If a TRNSFR response message contains application data, that data is used with the form definition from the specified exchange to build a form at the user's terminal.

4.4.4 The CLSTRN Message

The CLSTRN response message causes the current transaction instance to be terminated. The terminal reverts to its initial operating mode, which will probably consist of a transaction selection menu. The user can then choose a new transaction.

If a CLSTRN response message contains application data, that data is used with the form definition for the terminal's initial operating mode. That is, if the terminal's initial operating mode consists of a transaction selection menu, the application data is available to the definition of that form and may be included when the form is displayed at the terminal. Similarly, the application data is available to the form definition of the first form of the designated transaction, if that is the terminal's initial operating mode. (Initial operating modes are described in Chapter 9.)

4.4.5 The REPLY Message

The REPLY response message indicates that the user should remain in the current exchange. But before the user is allowed to enter data to the exchange form, the form will be modified as defined in the form definition.

REPLY response messages are typically used to return error messages and similar data to the terminal. The REPLY response message tells the user to "try again" with different input data. If he enters different data, it will be processed like his first entries.

Each form definition can contain one or more *reply definitions.* Each reply definition specifies several modifications that might be applied to that form. Some aspects of a form can be modified by a reply, such as:

- The text displayed in any field
- The set of function keys that are enabled
- The position of the terminal cursor

Yet, some aspects of a form are "frozen" when a form is first displayed and cannot be changed by a reply:

- The size of a field
- The data entry rules for a field
- The mode in which a field is displayed (such as, black-on-white)
- The rules for assembling an exchange message

Reply definitions are given identifying numbers in the form definition. A TST specifies the appropriate identifying number when it issues the system call that sends the REPLY response message.

When the REPLY response message is received at the terminal station, the form modifications specified in the specified reply definition are made. The user is then allowed to enter data and/or press any of the enabled function keys. If the user presses a user function key, an exchange message is constructed exactly as during the first cycle through the exchange. That exchange message is given the same routing list, and the exchange processing begins again with the new exchange message.

If a REPLY response message contains application data, that data is used with the specified reply definition from the current form. Thus, the data is available to the reply definition and may be displayed on the form as part of the reply modifications.

For example, a reply definition for error messages might specify that 80 characters of text are to be taken from the REPLY response message and placed in a field on the form. A TST could then invoke this reply definition by sending a REPLY response message. The message would have to specify the proper reply definition number, as well as 80 characters of error message text.

4.4.6 The ABORT Message

The ABORT response message is similar to the REPLY response message. The difference is that after the reply definition is applied to the form, *any function key* struck by the user acts like an ABORT function key.

This message enables a TST to inform the user of the reasons for the ABORT condition and lets the user read the informational message before the terminal reverts to its initial operating mode.

Do not confuse the ABORT response message with the TABORT system call. The TABORT call aborts the transaction instance immediately, and the user's terminal reverts to its initial operating mode without any possibility of an informational message. Consult the *TRAX Application Programmer's Manual* for further information about the TABORT system call.

4.5 AN EXAMPLE OF TRANSACTION PROCESSING PATHS

Let us return to the change customer transaction we have been using for an example in previous chapters.

Figure 4-3 shows how the transaction is divided into exchanges and shows the form and TSTs for each exchange. The arrows are possible processing paths.

Two things about the processing flow in Figure 4-3 deserve special attention:

1. Figure 4-3 shows two response messages during the processing of the second exchange. One message displays a data input error message, and the other displays a confirmation message. *Only one of these messages is sent during each execution of the transaction.* If the VALIDC TST decided that a data field was invalid, it would send an error message and remove the REWRIT TST from the routing list of the exchange message. However, if the VALIDC TST found no problems with the data, it would *not* send the error message, and the REWRIT TST would send the confirmation message to end the transaction.
2. At the end of the second exchange (after the confirmation message has been displayed) two function keys are enabled: The AFFIRM and the CLOSE keys. (The ENTER key has been disabled.) These two function keys have these effects:
   - The AFFIRM key takes the user back to the beginning of the transaction so that another customer record can be changed.
   - The CLOSE key takes the user to the transaction selection menu, so that another transaction can be chosen.

   Both keys end the current transaction instance.

You may have noticed that there is no provision in the first exchange for a nonexistent customer identification number or an invalid input format. As an exercise, you may want to add a transaction processing path to the diagram for this kind of error message. What kind of response message will the TST send? Will the TST include application data in the message? If so, what will this data be? What will have to be added to the first exchange's form definition to process this message properly?

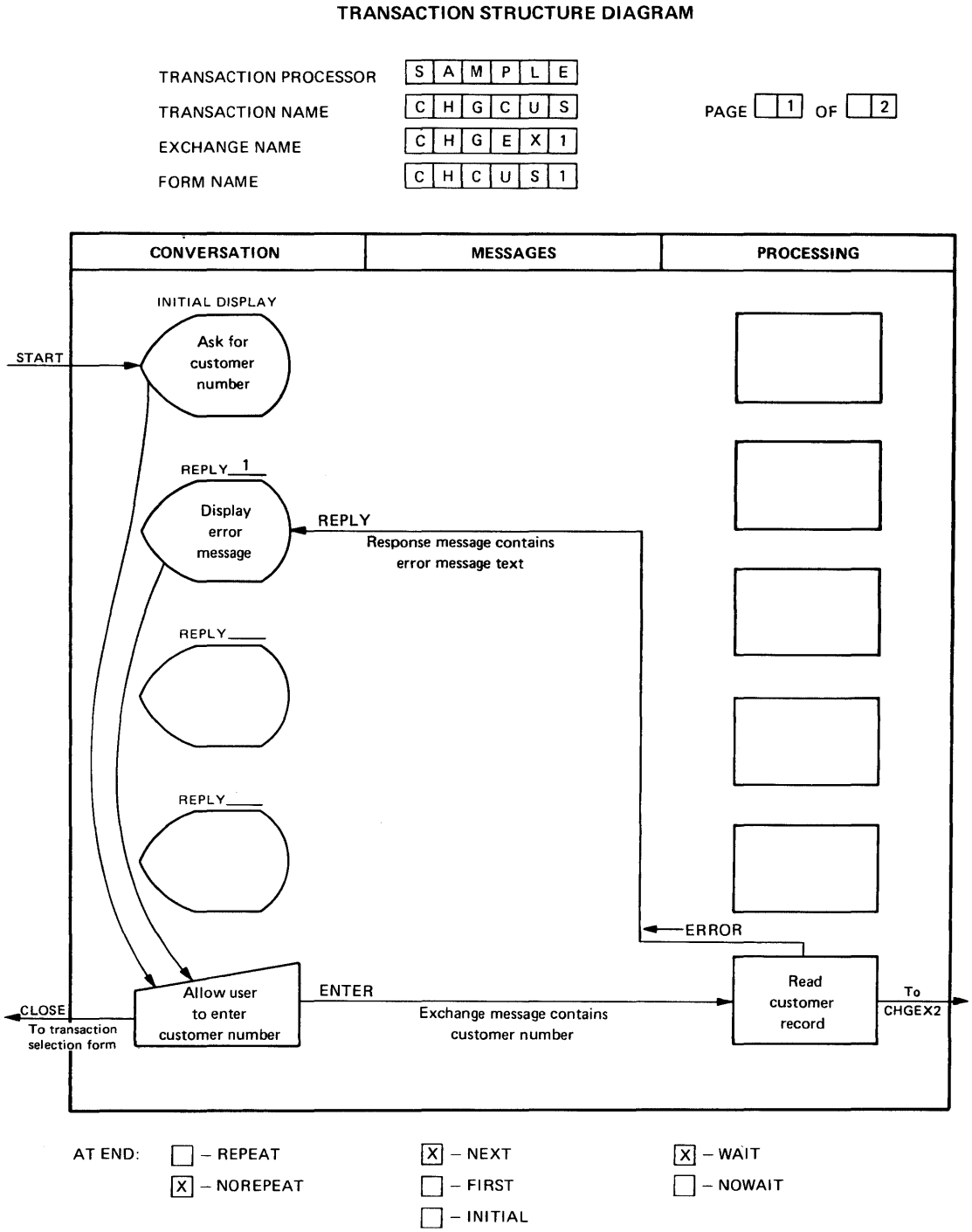


Figure 4-3 An Example of Transaction Processing Paths

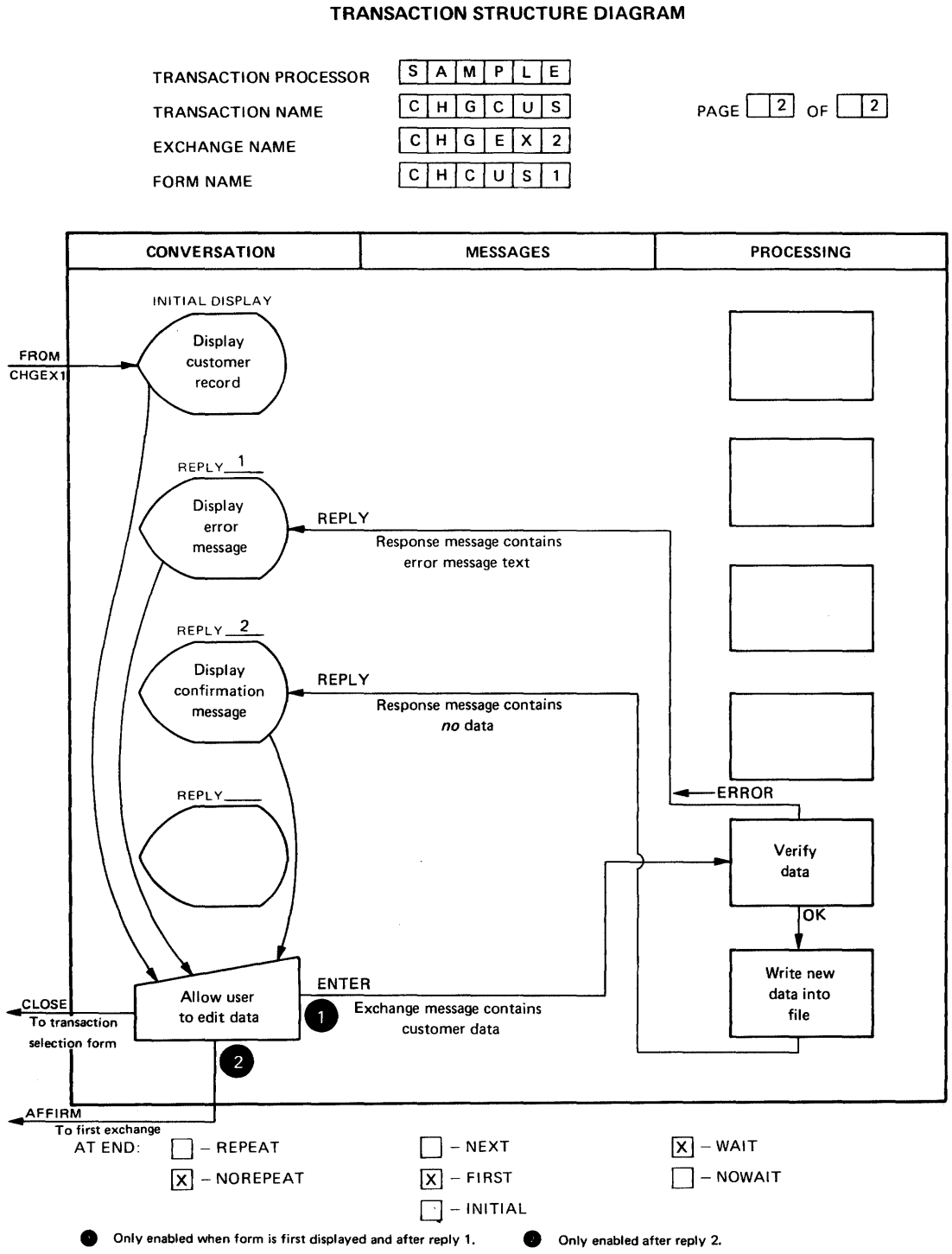


Figure 4-3 An Example of Transaction Processing Paths (Cont.)

CHAPTER 5
FORMS AND THE APPLICATION TERMINAL LANGUAGE

This chapter explains important forms characteristics and introduces the language you will use to write form definitions. At the end of the chapter, there is a form definition for an example transaction.

5.1 THE PURPOSE AND SCOPE OF THE APPLICATION TERMINAL LANGUAGE

The Application Terminal Language (ATL) is the language you will use to write form definitions. You can express every aspect of form design and usage with ATL. This includes:

- The layout of the form as it appears on the user's terminal
- The rules for how the user fills out the data fields on the form, including special restrictions for characters entered in each field and rules for field justification
- The function keys that the user uses at various points in the conversation
- The format of the response messages, if any, that supply data while the form is being generated at the user's terminal
- The format of the exchange message constructed from the data the user enters
- The modification to the form on instruction from a TST
- Special form options, such as whether the form is to be designated as a transaction selection menu

Because ATL is a specification language, you will probably find it convenient to write form definitions yourself rather than ask an application programmer to write them. It will not take you long to learn ATL. In fact, you may find that writing your own form definitions saves you significant paperwork, because once you prepare a form definition, the ATL compiler (see Section 5.2) generates most of the documentation your application programmers need – even mockups of each form, drawn to scale!

5.2 PREPARING A FORM DEFINITION WITH ATL

ATL is a compiled language. This means that there is an ATL compiler, and the compiler processes your source file into an encoded form definition that the transaction processor can use.

Creating an ATL source file is like creating a source file for any TRAX language. You use a support environment terminal and the EDIT program. Follow your preference about creating your source file: you can write the definition on paper and enter it at a terminal, or you can sit at the terminal and enter the definition as you develop it. The relative simplicity of ATL, as well as the power of the EDIT program when run from a video display terminal, makes either method feasible.

Once you prepare a source file, you must compile it. The ATL utility program does this for you. This utility program also manages the form definition files. You can use the ATL utility program to see what forms have been installed or to delete forms from the system, as well as to compile and install new forms.

When you use it to compile a form definition, the ATL utility program will do this:

- Scan your source file, checking your ATL statements for validity and consistency
- Generate an encoded form definition that can be used by a transaction processor
- Generate printed output that documents the compilation process and selected form parameters

The ATL utility program provides options that allow you to print all, part, or none of the compilation output and to install the finished form definition in a transaction processor form definition file.

5.3 KINDS OF FORMS

You can use the Application Terminal Language to define three kinds of forms:

- **Entry Forms.** These forms are used at the beginning of each exchange to collect data from the user. Your transaction processor will probably have more of this form than any other. The names of these forms appear in transaction definitions.
- **Transaction Selection Forms.** These are the "menu" forms that let a user choose the transaction he wishes to execute. As such, they are never part of the definition of any transaction. Instead, a terminal can be assigned one of these transaction selection forms, and the form will be displayed between transactions. Usually, these forms collect one item of data: the name of the chosen transaction.
- **Report Forms.** These forms are used to print data at output-only application terminals. Like transaction selection forms, they are never part of a transaction definition. Instead, each report message containing data to be printed also specifies a report form that determines the format to be used. Unlike the other two forms, report forms are never used interactively; that is, they never collect data from a terminal user. So, they never generate exchange messages and use only a small proportion of the ATL language facilities.

Even though only one type of form (the entry form) ever becomes part of a transaction definition, all three forms are kept in the transaction processor's form definition file. Within the file, they are identified by a short name (up to six characters in length) that is assigned when they are placed in the file. This six-character name is used to access forms within the transaction processor.

5.4 FORMS AND FIELDS

A form is composed of fields. Each field is a defined, contiguous area of the screen that has predefined characteristics.

Fields may not overlap one another; in other words, each character position on the screen or page may belong to only one field.

Any area of the screen or page that is not part of a field cannot contain application data.

There are five kinds of fields in ATL:

- Two fields, DISPLAY and PROMPT, are used to display information for the user.
- Two other fields, MENU and INPUT, are used to collect information from the user.
- The last field, PRINT, is used to print information on a hard-copy device.

Fields can continue from the end of one line to the beginning of the next line. This is considered a "contiguous" field, and the field uses the two lines as one long line.

5.5 ATL LANGUAGE ELEMENTS

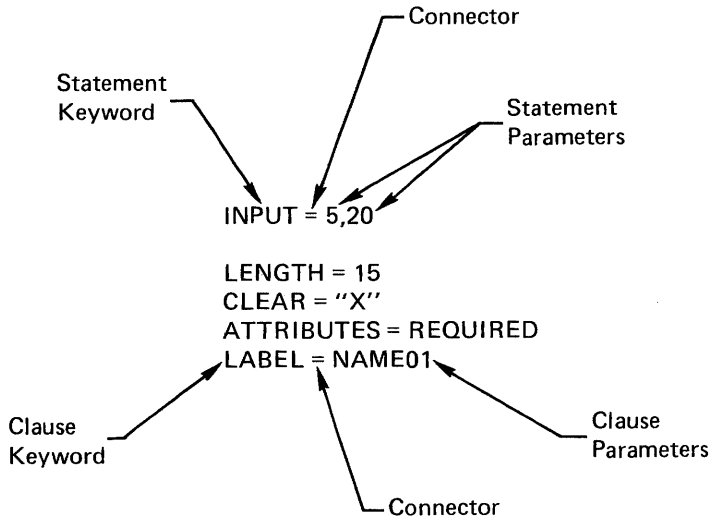
The *statement* is the fundamental grammatical unit in the Application Terminal Language. There are twelve statements in ATL. Each statement begins on a new source line, and has a unique statement keyword.

A statement keyword is often followed by one or more *statement parameters.* These parameters provide additional information about the statement's desired effect. Parameters are separated from the statement keyword by the equal symbol (=) and from each other by commas.

Following the statement keyword and the optional statement parameters, there may also be one or more *clauses.* Each clause modifies the effect of its associated statement.

Each clause has a *clause keyword* (by which it is recognized) and perhaps one or more *clause parameters* (which specify the effect desired). Like statement keywords and statement parameters, clause keywords and clause parameters are separated from each other by an equal symbol (=), and the clause parameters are separated from each other by commas.

A typical ATL statement is shown here. Note the statement components:



The format of this statement is arbitrary; the format shown was selected for easy reading. The only format restriction is that the statement keyword (INPUT) must begin on a new line.

Figure 5-1 might help you see the general syntax rules for ATL statements. This figure shows which ATL language elements follow one another. As long as you follow the arrows in a *forward direction,* any path through this diagram will describe a legal ATL statement. (Of course, specific ATL statements make further restrictions upon syntax.)

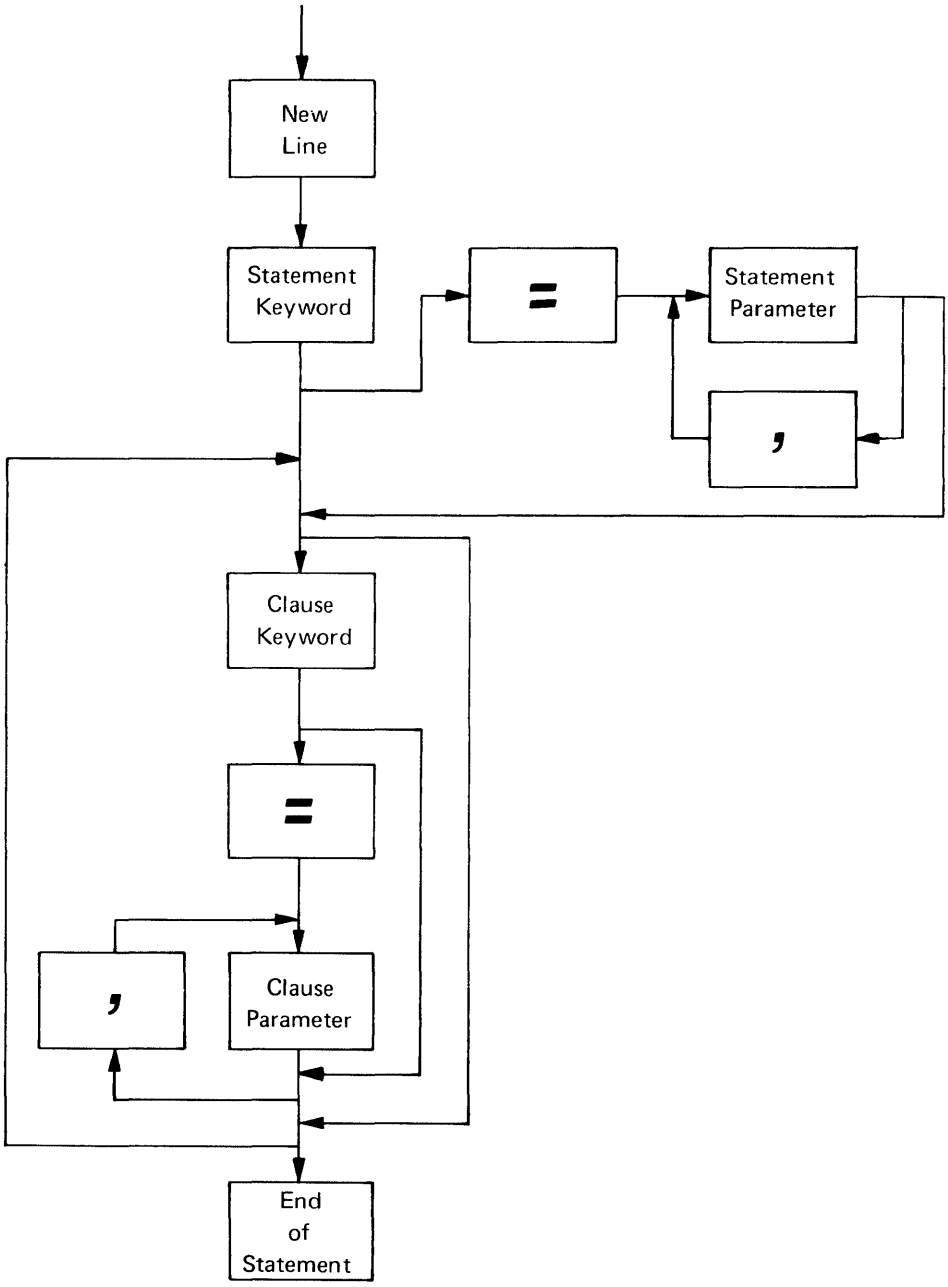


Figure 5-1 ATL Statement Syntax Diagram

You might find it helpful to take the typical ATL statement shown before and verify that it is legal according to the syntax diagram in Figure 5-1.

5.6 STATEMENT GROUPS

Remember, a form definition serves many purposes. Each aspect of a form must be specified in the form definition; and each specification must be handled by a separate group of one or more kinds of ATL statements.

These groups are listed in Table 5-1, with the ATL statements for each group. You can learn more about these statements and how they are used in the *ATL Language Reference Manual.*

Table 5-1 ATL Statements (Grouped by Purpose)

Group	Purpose	Keyword	Usage		
			Entry Forms	Transaction Selection Forms	Report Forms
A	Define General Form Parameters	FORM	x	x	x
B	Define Fields on Screen	INPUT	x	x	
		PROMPT	x	x	
		DISPLAY	x	x	
		MENU	x	x	
C	Define Fields on Report	PRINT			x
D	Define Exchange Message	MESSAGE	x	[1]	
E	Define Reply Actions	REPLY	x	[2]	
F	Compiler Directives	DEFAULT	x	x	x
		REPEAT	x	x	x
		REND	x	x	x
		END	x	x	x

[1] Possible in special circumstances

[2] These replies are not activated by a TST; they are used for error messages during transaction selection

5.7 STATEMENT ORDER

Because ATL is a *specification* language and not an *algorithmic* language, the order in which statements appear in the definition is generally unimportant.

There are, however, five instances where the order of statements becomes important:

- Each form definition should begin with a FORM statement and must end with an END statement.
- The INPUT statement (Group B, Table 5-1) defines a field so the user can enter data. The user's cursor advances from INPUT field to INPUT field in the order these fields are defined in the form definition.
- The placement of DEFAULT statements (Group F) is important, because these statements affect only those statements that follow them in the form definition.
- The placement of REPEAT and REND statements (Group F) is important, because they affect only those statements between them.
- The placement of statements that use the dot symbol (.) in statement or clause parameters is important, because the value of this symbol depends on preceding statements. The dot symbol is explained in Section 5.9.

Although statement order is generally not important to the ATL compiler, you should organize your form definitions in sections conforming to statement Groups A through E in Table 5-1. Statements in group F, of course, must be scattered throughout the form definition.

This way, you will have distinct sections within each form definition corresponding to the different purposes of a form. For example, one section might define the fields and where they appear, another section might define the format of the exchange message, and so forth. This makes your form definition easy to understand and maintain.

You can preserve these distinct sections of the form definition even if you use the REPEAT and REND statements. (These statements allow you to define a series of related fields, or other aspects of a form, without having to enter separate specifications for each field. To preserve the distinct sections of the form definition, use several REPEAT/REND pairs: one pair in the section that defines fields; another in the section that defines the exchange message; and perhaps another in the section that defines the effect of replies. You *could* define the fields, their position in the exchange message, and their modifications during replies in a single REPEAT/REND sequence – but you would have to abandon separate form definition sections.

Use multiple REPEAT/REND sequences without worrying about their effect on system performance. REPEAT/REND sequences are only a compiler shorthand, and they do not result in statement loops that are executed at run time!

See the *ATL Language Reference Manual* for details on REPEAT/REND.

5.8 COMMENTS IN FORM DEFINITIONS

Comments are a valuable part of form definitions. Consequently, ATL allows you to insert comments in a form definition at any point in the source file. These comments help document the application and simplify maintenance of the form definition.

Begin each comment with an exclamation point (!). The ATL compiler ignores source text beginning with an exclamation point. (This does not include exclamation points within text literals.) After an exclamation point, the ATL compiler resumes scanning text at the next source file line.

5.9 SHORTHAND NOTATION

ATL allows two shorthand notations to reduce bulk in your source files and to speed the process of writing form definitions.

1. **REPEAT and REND Statements** (Section 5.7) allow you to avoid repetitious sections of a form definition. Enter a portion of a repetitious section, and the compiler expands it for you when the form definition is compiled.
2. **The Dot Symbol (.)** allows you to have a numeric parameter behave in a sequential manner. For example, the position of an INPUT statement might be better expressed as "two spaces to the right of the last field" rather than as an exact row and column position. You can use the dot symbol in two places:
   a. **Group B and C Statements.** You may use the dot symbol in both the row and column parameters of Group B and C statements. When used in a line-number parameter, the dot symbol stands for the line on which the previous field was placed. When used in a column-number parameter, it stands for the first column to the right of the previous field.
   b. **The REQUEST Function.** The REQUEST function is the ATL construct that you use to retrieve data from a response message. You may use the dot symbol as the first parameter of this function; it specifies the byte position in the response message where the data is to be found. In this case, the dot symbol stands for the first byte of the response message immediately after the last data accessed by the REQUEST function. Successive REQUEST functions can retrieve sequential fields from a response message by using the dot symbol together with explicit field lengths as the second REQUEST parameter.

Whenever you are allowed to use the dot symbol, you can combine it with a numeric offset either plus or minus. For instance, to indicate the line below the previous field, the parameter .+1 might be used.

Several examples of the dot symbol are listed here.

INPUT = .,20	An INPUT field on the same line as the previous field, but beginning at column 20
DISPLAY = .+2,1	A DISPLAY field two lines down from the previous field, beginning at the left margin
PROMPT = .,.	A PROMPT field immediately to the right of the previous field
VALUE = REQUEST (.,6)	Retrieves a 6-character data field from a response message, beginning at the end of the previously retrieved data field

WRITE = FLD01,REQUEST(.+10,25)	Retrieves a 25-character data field from a response message, beginning 10 characters after the end of the previously retrieved data field

5.10 ATL AND FORM DESIGN

ATL is a simple language, but that does not limit the forms it can define. Good form design is a challenge, and it is an exercise in human factors and aesthetics as well as a technical activity. When your application is installed, the forms are its most visible component. And, mediocre form design takes its toll in user productivity and morale. With this in mind, be sure to spend the necessary effort and time to design suitable forms for your application.

5.11 A TYPICAL FORM DEFINITION

A typical form definition begins on page 5-9. It is the form from the second exchange of the change customer transaction.

As you recall, this form displays the old values from a customer record and allows the user to edit those values. The new values are then sent back to the system in an exchange message.

This form has two defined replies:

Reply 1: This reply displays error messages received from TSTs. The enabled function keys are not changed, so the user can edit the data he has entered and send it again.

Reply 2: This reply is used only after the new customer data has been written in the customer file. It displays a confirmation message for the user and adjusts the function keys so that the user can press only the AFFIRM key.

```
!****************************************************************
!
!       Definition of form CHCUS2
!
!       The second form for the Change Customer transaction
!
!       This form displays the selected customer master file record
!       and allows the user to change the data contained in it.
!       The changed data is then sent back to the system in an
!       exchange message.
!
!****************************************************************

!****************************************************************
!
!       Group A Statements - Define General Parameters
!
!****************************************************************

DEFAULT
        ENABLE = AFFIRM
        ENABLE = CLOSE
        CLEAR = " "

FORM
        SPLIT=8                         !8 lines of display area

!****************************************************************
!
!       Group B Statements - Define Fields on Screen
!
!****************************************************************
!================================================================
!
!       These are the application fields
!
!       The two error text fields are used to display error
!       messages contained in response messages
!
!================================================================

DISPLAY = 3,12
        VALUE = "Customer Master File Subsystem - Change Customer Transaction"
        LENGTH = 60
        ATTRIBUTE = REVERSE,NOBLANK

DISPLAY = 5,1
        LABEL = REPLY.TEXT.A
        LENGTH = 80

DISPLAY = 6,1
        LABEL = REPLY.TEXT.B
        LENGTH = 80
```

```
PROMPT = 1,1
         VALUE = "Customer Number"

INPUT = .,20
         LABEL = CUST.NO
         VALUE = REQUEST(.,6)
         ATTRIBUTE = REVERSE,NOMODIFY

PROMPT = .+1,1
         VALUE = "Customer Name"

INPUT = .,20
         LABEL = CUST.NAME
         LENGTH = 30
         VALUE = REQUEST(.,30)
         ATTRIBUTE = REQUIRED,REVERSE

PROMPT = .+2,1
         VALUE = "Address"

INPUT = .,20
         LABEL = ADDRESS.A
         LENGTH = 30
         VALUE = REQUEST(.,30)
         ATTRIBUTE = REVERSE

INPUT = .+1,20
         LABEL = ADDRESS.B
         LENGTH = 30
         VALUE = REQUEST(.,30)
         ATTRIBUTE = REVERSE

INPUT = .+1,20
         LABEL = ADDRESS.C
         LENGTH = 30
         VALUE = REQUEST(.,30)
         ATTRIBUTE = REVERSE

PROMPT = .+1,1
         VALUE = "ZIP Code"

INPUT = .,45
         LABEL = ZIP.CODE
         LENGTH = 5
         VALUE = REQUEST(.,5)
         ATTRIBUTE = REVERSE,NUMERIC,FULL

PROMPT = .+1,1
         VALUE = "Telephone"

PROMPT = .,20
         VALUE = "("

INPUT = .,.
         LABEL = TEL.AREA.CODE
         LENGTH = 3
         VALUE = REQUEST(.,3)
         ATTRIBUTE = TAB,REVERSE,NUMERIC,FULL
```

```
PROMPT = .,.
        VALUE = ") "

INPUT = .,.
        LABEL = TEL.EXCHANGE
        LENGTH = 3
        VALUE = REQUEST(.,3)
        ATTRIBUTE = TAB,REVERSE,NUMERIC,FULL

PROMPT = .,.
        VALUE = "-"

INPUT = .,.
        LABEL = TEL.EXTN.NO
        LENGTH = 4
        VALUE = REQUEST(.,4)
        ATTRIBUTE = REVERSE,NUMERIC,FULL

PROMPT = .+2,1
        VALUE = "Attention"

INPUT = .,20
        LABEL = ATTENTION
        LENGTH = 20
        VALUE = REQUEST(.,20)
        ATTRIBUTE = REVERSE

PROMPT = .+2,1
        VALUE = "Credit Limit ($)"

INPUT = .,20
        LABEL = CREDIT.LIMIT
        LENGTH = 12
        VALUE = REQUEST(.,12)
        CLEAR = "0"
        ATTRIBUTE = RIGHT,REVERSE,SIGNED

!=============================================================
!
!       This is a prompt field that tells the user what
!       function keys may be used.  The content of this field
!       may be changed by reply definitions (see below) if those
!       reply definitions change the enabled function keys.
!
!=============================================================

PROMPT = 15,1
        LABEL = KEY.PROMPT
        LENGTH = 80
        VALUE = "Function Keys:  ",
                "ENTER to refile customer record, ",
                "CLOSE to quit without filing"
        ATTRIBUTE = REVERSE
```

```
!************************************************************
!
!       Group D Statements - Define Exchange Message
!
!************************************************************

MESSAGE = 1
        VALUE =
                CUST.NO,
                CUST.NAME,
                ADDRESS.A,
                ADDRESS.B,
                ADDRESS.C,
                ZIP.CODE,
                TEL.AREA.CODE,TEL.EXCHANGE,TEL.EXTN.NO,
                ATTENTION,
                CREDIT.LIMIT

!************************************************************
!
!       Group E Statements - Define Replies
!
!************************************************************

!============================================================
!
!       Reply 1 is affirmative reply:
!
!               Enable AFFIRM Key
!               Disable all other function keys
!               Write "TRANSACTION COMPLETE" on screen
!               Erase old function key message
!               Write new function key message
!               Write assigned Customer Number on screen
!               Write Reply Number into screen header
!
!============================================================

REPLY = 1
        ENABLE = AFFIRM
        DISABLE = ENTER
        WRITE = REPLY.TEXT.A," *** TRANSACTION COMPLETE ***"
        WRITE = KEY.PROMPT,FILL(" ",80)
        WRITE = KEY.PROMPT,"Function Keys:  AFFIRM to proceed"


!============================================================
!
!       Reply 2 is the reply for validation error messages
!
!               Write 2 80-character error messages onto screen
!               Write reply number into screen header
!
!============================================================

REPLY = 2
        WRITE = REPLY.TEXT.A, REQUEST(1,80)
        WRITE = REPLY.TEXT.B, REQUEST(81,80)

END
```

CHAPTER 6
TRANSACTION STEP TASKS

This chapter explains how transaction step tasks (TSTs) are written, compiled, debugged, and finally installed as part of a transaction processor.

6.1 THE PURPOSE OF TSTS

TST stations are usually responsible for the application-related processing in a transaction processor. Other stations play primarily "overhead" or "housekeeping" roles – managing terminals, storing messages, transmitting data to other transaction processors. It is typically the TST stations in a transaction processor that do the application processing – checking user input, calculating results, reading records from files, updating files.

As you know from earlier chapters, a TST accomplishes this by accepting an exchange message that contains data to be processed and returning a response message that contains results of that processing. Each TST in a transaction processor has a predefined purpose and applies a specified sequence of processing steps to an arriving exchange message. The definition of each transaction determines which TSTs are activated to process that transaction's exchange messages.

While processing an exchange message, TSTs typically take one or more of these actions:

- If the transaction instance has originated at an application terminal, the exchange message contains user input. The TST can validate this input and perhaps edit it to packed rather than display format.
- The TST retrieves the data file records needed to process the transaction instance. Frequently, the retrieved records are then placed in the transaction workspace so that subsequent TSTs access them easily.
- The TST applies logical and arithmetic calculations to the data in the exchange message and also to the data retrieved from files. The results of these calculations are placed in the transaction workspace, if necessary, for the benefit of subsequent TSTs.
- The TST updates data files. It updates records read by previous TSTs, as well as records that it has read itself.
- The TST controls the progress of the transaction instance by altering the contents of the exchange message, the routing list, or the exchange that will be executed next. In circumstances that warrant it, the TST terminates the transaction instance.
- The TST creates, writes in, or reads from work files. Work files are not part of the application's permanent data file set but are temporary files used by transaction instances.
- The TST deposits messages at a mailbox station and retrieves messages left at mailbox stations.
- The TST spawns other transaction instances. That is, the TST itself initiates new transaction instances with exchange messages of its own construction. These spawned transaction instances have a life of their own, separate from the transaction instance the TST is processing at the time.
- The TST sends data to an output-only terminal to be printed.
- If it is processing a transaction instance initiated by a user at an application terminal, the TST sends data to the terminal for display.

6.2 GENERAL STRUCTURE OF A TST

The central element in a TST is an application program. This program can be written in your choice of three languages:

- COBOL
- BASIC-PLUS-2
- MACRO-11

NOTE

MACRO-11 is not recommended for applications.

You can find more about these languages in the corresponding TRAX language reference manual.

The application program is written as a subroutine. That is, the program has the same structure that it would have if you were going to call it from a mainline program written in the same language.

As a subroutine, the application program must accept two parameters: an exchange message data structure and a transaction workspace data structure. Each time the TST is activated, the transaction processor will provide a pair of parameters – an exchange message and its corresponding transaction workspace.

The entry point in this application program (that is, the instruction at which execution must begin) must always be called TSTEP. This stands for "*TST E*ntry *P*oint" and marks the place in the program where the transaction processor will start execution. For COBOL and BASIC-PLUS-2, this means that the program must be called TSTEP as well, because these languages use the same name for the entry point in the compiled code as for the name of the entire module. You will have to use comments in each program to identify the program, and the "official" program name must be TSTEP.

NOTE

The term "program name" in the preceding paragraph applies to the name declared in the program itself. For example, it refers to the name used in the PROGRAM-ID clause of a COBOL program. This is the name that must always be declared as TSTEP. The source file where the program's source code resides and the task image file where the executable program image resides can be given any name you wish.

The central application program in each TST can have separately compiled subroutines if you wish. The languages that you can use to write subroutines will depend on the capabilities of the language you have chosen for the central program. For example, COBOL programs can have subroutines written in MACRO-11; but MACRO-11 programs cannot have subroutines written in COBOL. For further information about a language's restrictions on separately compiled subroutines, consult the appropriate language reference manual.

A TST in its ready-to-execute state is a task image. As such, it will contain additional code besides the compiled application program and its subroutines. These additional support routines serve several purposes:

- Initialization when the TST is first activated
- Support for special transaction processor functions, such as the system calls available to TSTs
- Linkage to RMS, the TRAX file access subsystem
- Linkage to run-time support for the programming language

For system efficiency and memory conservation, each TST task image does not contain copies of all supporting routines. Instead, a single set of support routines (called the *TST library*) is kept in the transaction processor. Each TST then contains only the minimum linkage routines, and the TST library is connected to the TST task image by memory mapping techniques when the TST is activated. This results in smaller TST task images, quicker TST task loading from disk, and lower memory requirements for a running transaction processor.

6.3 PROGRAMMING A TST

You will find that programming a TST is similar to any programming project in the language you have selected. You must remember, of course, to provide the two input parameters to your TST; and you must familiarize yourself with the system calls that TSTs use to interact with the transaction processor. Other than these two considerations, coding TSTs is straightforward.

6.3.1 Input Parameters

The application program you write for a TST is a subroutine, and this subroutine has two input parameters: the exchange message and the transaction workspace for a transaction instance.

These parameters are passed *by name* and not *by value*. This means that the master copy of these data structures is passed to the application program; if the application program changes data in either of the two data structures, the changes will be seen by the TSTs that subsequently receive them.

In addition, the data structures are passed without moving them in memory. That is, an exchange message does not move from TST to TST; it remains in a fixed memory location and the TSTs are "connected" to it by memory mapping techniques. This means that the TST must use specific language constructs to access these parameters:

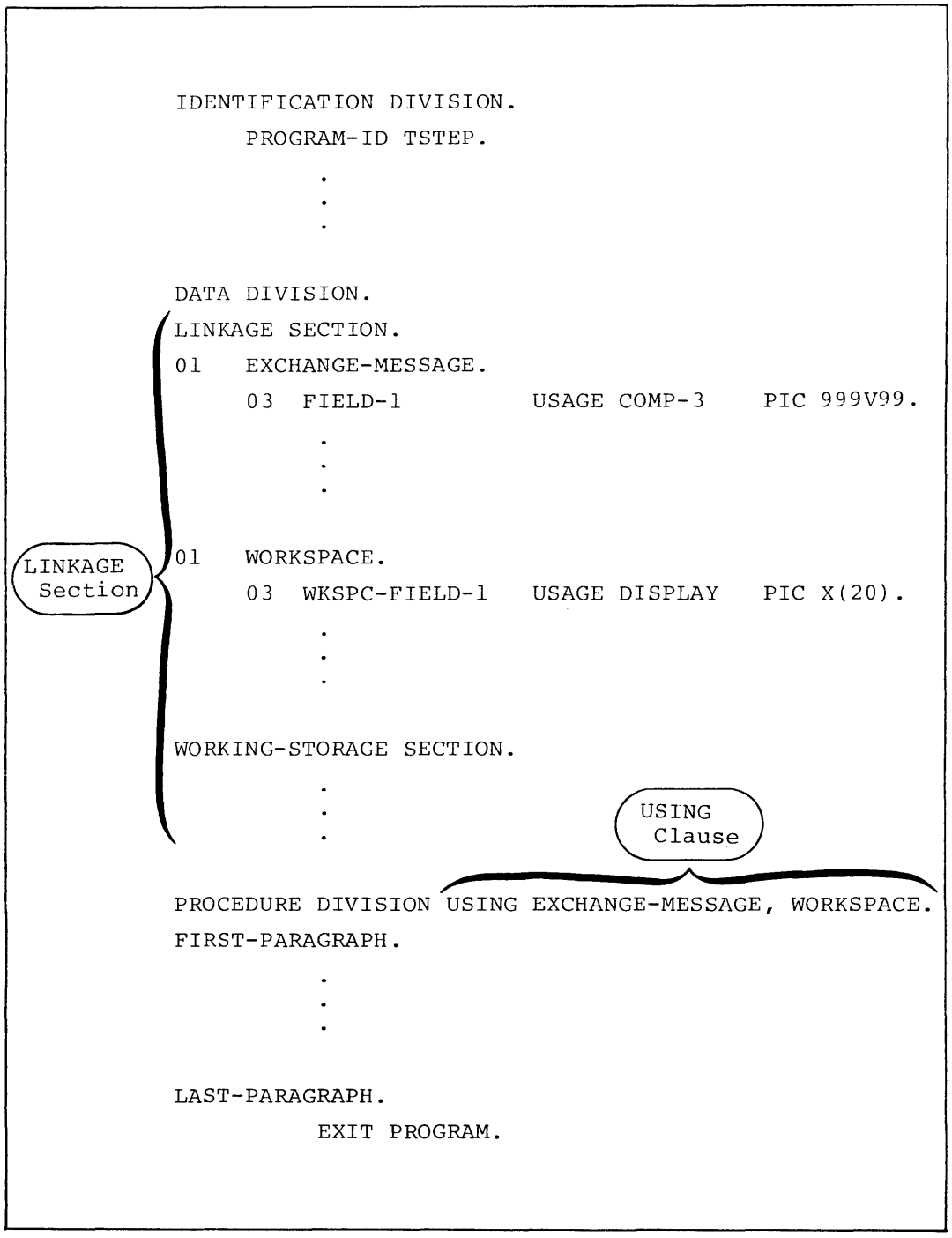
- In COBOL, the linkage is accomplished through the LINKAGE section of the DATA division and the USING clause of the PROCEDURE division.

  The data structures for both input parameters are defined in the LINKAGE section, and they are given data names so they can be referenced in the rest of the program.

  Then the USING clause names the two data structures. The first data name designates the data structure that will be used to access the exchange message; the second, the transaction workspace.

  Once this linkage is set up, the COBOL programmer can access the exchange message and transaction workspace without concern for special techniques.

  The following example shows fragments of a COBOL TST, illustrating the LINKAGE section and the USING clause.


IDENTIFICATION DIVISION.
PROGRAM-ID TSTEP.
DATA DIVISION.
LINKAGE SECTION.
01 EXCHANGE-MESSAGE.
03 FIELD-1 USAGE COMP-3 PIC 999V99.
01 WORKSPACE.
03 WKSPC-FIELD-1 USAGE DISPLAY PIC X(20).
WORKING-STORAGE SECTION.
PROCEDURE DIVISION USING EXCHANGE-MESSAGE, WORKSPACE.
FIRST-PARAGRAPH.
LAST-PARAGRAPH.
EXIT PROGRAM.
LINKAGE Section
USING Clause

- In BASIC-PLUS-2, the linkage is accomplished with enhancements to the language features normally used when writing subroutines.

  Instead of starting his program with the SUBROUTINE statement, a BASIC-PLUS-2 programmer starts a TST with a special TST statement. This statement declares the program a TST, assigns the mandatory name TSTEP to the program, and designates the two mandatory parameters: the exchange message and the transaction workspace.

  If the programmer wishes to break the exchange message into component fields for easier programming, he uses one or more MSGMAP statements. The MSGMAP statement is similar to the MAP statements used to break a record into component fields, except the MSGMAP statement operates on the exchange message rather than a record in a file.

  Or, if the programmer wishes to break the transaction workspace into component fields for easier programming, he uses one or more WRKMAP statements. The WRKMAP statement is similar to the usual MAP statement, except the WRKMAP statement operates on the transaction workspace rather than a record in a file.

  Finally, the programmer ends his program with the TSTEND statement rather than the END statement.

  Once these conventions are satisfied, the TST programmer accesses fields in the exchange message and the transaction workspace by using the variable names declared in the MSGMAP and WRKMAP statements.

  Here are portions of a BASIC-PLUS-2 TST, illustrating these conventions.

```
100       TST TSTEP (MSG.SPACE$,WRK.SPACE$)
                        .
                        .
                        .


200       MSGMAP
            EM.CUSTOMER.REC$              =36
210       MSGMAP
            EM.CUSTOMER.NO$               =6
            EM.CUSTOMER.NAME$             =30
300       WRKMAP
            TEMP1$                        =10
            TEMP2$                        =25
              .
              .
              .
            (BODY OF PROGRAM)
              .
              .
              .
32767     TSTEND
```

- In MACRO-11, the parameters passed are not the data structures themselves, but addresses where those data structures may be found. The data structures are mapped into the TST address space before the TST is activated, and the address parameters are always 16-bit addresses within the TST address space.

  You can find more information about MACRO-11 TST programming techniques in the *TRAX Application Programmer's Guide.*

6.3.2 System Calls

A TST must use a TRAX system call whenever it undertakes some action which is not supported directly by the language in which the TST is programmed. These actions are primarily control functions for the transaction processor: adding or deleting stations on routing lists; sending response, mailbox, or report messages; asking for information about the state of the system or the current transactior instance. Each of the available system calls is discussed in the *TRAX Application Programmer's Guide.*

The language you use to write a TST will affect the way in which the system calls are programmed:

- In COBOL, the CALL verb is used with its USING clause. The parameter following the CALL verb is a data name or character string containing the name of the routine being called. Following this comes the USING clause, which lists each of the parameters being passed to or from the routine.
- In BASIC-PLUS-2, the CALL verb is used with its BY REF clause. The parameter following the CALL verb is the name of the routine being called. This is followed by the BY REF clause, which lists each of the parameters being passed to or from the routine.

NOTE

A routine name is coded directly into the CALL statement; the routine name is *not* held in a string variable or string constant as with COBOL.

6.4 DEBUGGING A TST

Unlike programs written for other systems or for use in the support environment of a TRAX system, TSTs are not invoked directly from a terminal. TSTs execute in a "background" mode without direct connection to application terminals. This means that programmers have to use a different technique to test and debug TSTs they write.

TRAX provides three debugging techniques for TSTs. These techniques are presented here as they would probably be used during the testing of a newly written TST. The methods are:

- Stand-alone debugging
- Debugging in a transaction processor
- Traced operation in a transaction processor

6.4.1 Stand-alone Debugging

For rudimentary debugging, the programmer must build a stand-alone task image containing the TST and necessary support software: RMS access methods, TST library modules, support routines (the Object Time System or OTS) for the programming language, and so forth.

These modules are included in the task image because the task image will be debugged in the support environment and not under the supervision of a transaction processor. The task image must therefore be self-supporting.

During this phase of debugging, the TST cannot receive exchange messages and system workspaces from the transaction processor nor can it send messages to other transaction processor components. Instead, the programmer prepares a file containing test exchange messages and workspaces, and any messages sent by the TST are *saved* for the programmer's inspection. Similarly, any calls the TST makes to the TST library will be *noted and logged* for the programmer's inspection; but the calls will have no other effect in most cases.

The programmer can use the features of his programming language to assist the debugging process. For instance, he will be able to use BASIC-PLUS-2's extensive breakpoint and variable inspection facilities.

6.4.2 Debugging in a Transaction Processor

Once the TST begins to function on a rudimentary level, it can be installed in a transaction processor for in-place testing. This involves the building of a new task image, because many of the modules that were included in the stand-alone task image are no longer required. The modules will be provided by the transaction processor via memory-mapping techniques.

In this phase of debugging, the TST becomes an operating part of the transaction processor; but it still retains a connection to a support environment terminal so the application programmer can inspect the TST operation.

The TST is activated in the usual way when exchange messages arrive at its TST station. When it is activated, a breakpoint previously set within the TST activates the support environment terminal. The programmer can step through the TST as it processes the newly arrived exchange message. Now, all file accesses and system calls can have their proper effect, because the TST has been installed as part of the transaction processor.

For this phase of debugging, then, there will be two terminals involved with the TST: the application terminal that provides the exchange message the TST is processing and the support environment terminal the programmer will use to debug the TST.

6.4.3 Traced Operation in a Transaction Processor

A third debugging technique is sometimes useful to debug TSTs as they operate in a transaction processor. This technique involves a trace log, which records significant events as they occur in the transaction processor.

In this technique, no support environment terminal is involved. The transaction processor runs on its own, without outside intervention. But a special option is invoked that causes the transaction processor to record each event: messages being sent, system calls being isssued, and so forth.

After some period of traced operation, the transaction processor can be stopped and the trace log inspected. Typically, the trace log is most useful in finding system-integration problems – that is, problems with interaction between TSTs rather than problems isolated in one TST.

6.5 INSTALLING A TST

Installing a TST involves two separate procedures:

- Making sure the TST station has the proper parameters
- Making sure the TST task image is built correctly

6.5.1 TST Station Parameters

The parameters you assign to a TST station will have a direct and significant effect on the way the TST behaves. For example, you can specify the maximum number of executing copies of the TST that can be active at one time. Your choice of this parameter will affect the performance (and perhaps the correct operation) of the TST.

The TST station parameters and their effects on TST operation are discussed further in Section 16.2.

6.5.2 The TST Task Image

TST task images are built with the TSTBLD utility program. This program can build the different task images required for normal TST operation as well as the three modes of TST debugging. When a TST is installed, you must be sure that the task image has been constructed so that it is compatible with the way you will be installing and using it.

The *TRAX Application Programmer's Guide* tells how to use the TSTBLD utility program.

6.6 EXECUTING A TST

Once installed in a transaction processor, a TST remains idle until an exchange message arrives at its TST station. The TST's location on the system disk is remembered, though, so that the TST task image can be located quickly when it must be activated.

When an exchange message arrives, the transaction processor attempts to start the TST. However, there may be several obstacles:

- There may be too many active copies of this TST already. If so, the new exchange message will have to wait.
- The TST task image may not be in memory. If so, an area of memory must be allocated and the image read from disk.
- Even if an area of memory is available, there may be a TST of higher priority waiting to be activated. If this is the case and if that TST will fit into the available memory, it will be given priority and the other TST will have to wait.

In most situations, though, none of these obstacles will be present. The maximum-copies limit will not be exceeded, and a TST task image will be in memory from a previous execution of that TST. The TST can therefore be activated promptly and can begin to process the exchange message.

Once a TST has been started successfully, it runs as a separate task under the TRAX kernel operating system. It runs independently of and parallel to the management services of the transaction processor. The TST is run as a *non-checkpointable* task, which means that it will run to completion without being interrupted or swapped out of memory. The only pauses in its execution will be those necessary to handle its system calls and input/output operations.

6.7 APPLICATION FILE ACCESS FROM TSTS

Although application programmers use normal programming language verbs to read and write application data files, the requirements of shared file access and transaction processor architecture make the flow of data different from that in a stand-alone program. This topic is discussed in Chapter 8.

6.8 STUDYING A TYPICAL TST

Perhaps the best way to gain a familiarity with TSTs and their use is to study an existing TST. You can find examples of TSTs programmed in various languages in the *TRAX Application Programmer's Guide.*

CHAPTER 7
PRESERVING TRANSACTION INSTANCE CONTEXT

A stand-alone program provides its own context: values stored in its own program variables let the stand-alone program know what has been done and what must be done. In a transaction processor, a TST's working storage cannot be used for this purpose; each TST executes only a short time, and many different TSTs execute as part of a single transaction instance.

This means that the transaction processor must maintain enough information for each transaction instance so that TSTs and system support modules that work on that transaction instance can tell what has happened previously. This information is called the *transaction instance context.*

A discussion of transaction instance context must center on two subjects:

- The transaction slot
- Context requirements for file access by TSTs

7.1 THE TRANSACTION SLOT

The transaction instance slot, or *transaction slot* for short, is the primary repository of context information. Each transaction instance has a transaction slot, and it contains three principal kinds of data:

1. It contains the current *exchange message* for the transaction instance.
2. It contains the *transaction workspace* for the transaction instance.
3. It contains the *system workspace* for the transaction instance.

The exchange message and transaction workspace have been discussed in earlier chapters. The third section of the transaction slot, the system workspace, is introduced here for the first time. The transaction processor uses this section of the transaction slot to keep context information it needs to support the transaction instance.

If a copy of the transaction slot is made immediately after a new exchange message has been constructed and before the first TST begins to process that exchange message, this copy can be used as a checkpoint to restart the transaction instance. This is possible because the transaction slot at that moment completely defines the state of the transaction instance.

When you specify *exchange recovery* for a transaction, you are asking the transaction processor to do exactly this. With exchange recovery, each time an exchange message is built the entire transaction slot is written to a disk file. Should anything go wrong during the processing of the exchange message, a TST can request that this copy be read back from the disk so that processing can start again. Except for updates to unstaged files, this restart will be transparent to the user at the terminal. (Exchange recovery is discussed in detail in Section 10.2.1.)

Figure 7-1 shows a transaction slot.

The transaction workspace and system workspace are of constant size; they are allocated according to parameters in the transaction definition.

The exchange message, of course, can be of varying size. The exchange message area in the transaction slot, however, is of constant size; the transaction slot is allocated to hold the largest exchange message used by the transaction. Smaller exchange messages will fill only a part of the space allocated.

As you know, the exchange message and transaction workspace are accessible to the TST programmer in a read/write manner. The system workspace is never accessible to a TST programmer; it is accessible only to system support software within the transaction processor.

The transaction slot also plays a role in transaction journaling. When a transaction uses journaled files, the transaction slot is written to the journal device as each transaction instance finishes. This records the state of the transaction instance at the end of transaction execution and provides a record of the transaction instance. If a TST updates a file that is supposed to be journaled, a copy of each updated record is placed in the system workspace. When the transaction slot is written to the journal device, these copies of the updated records will be written too. A record of file updates is therefore maintained via the transaction slot. (Journaling is described in detail in Section 8.7.)

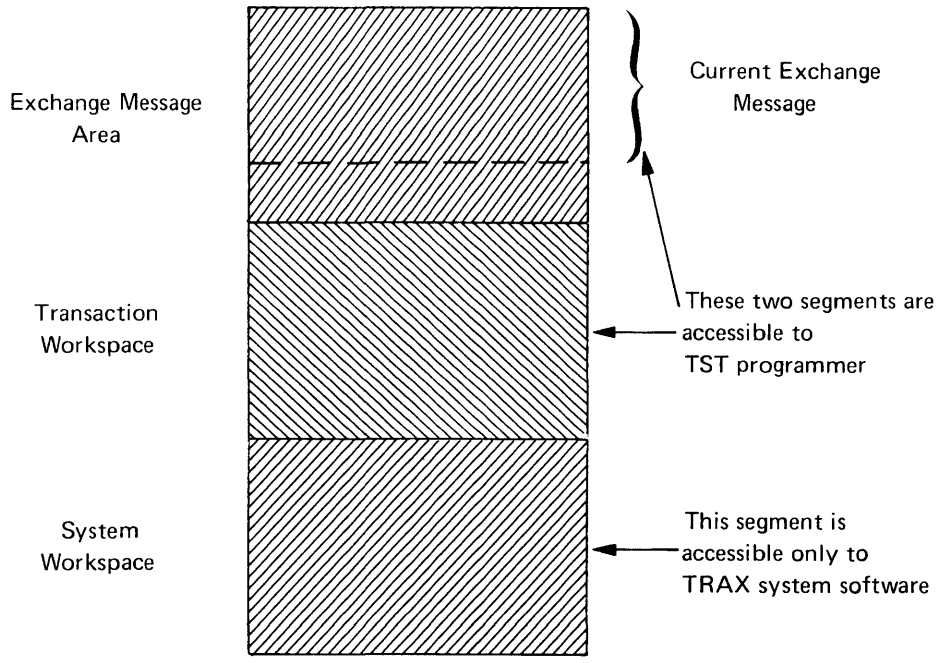


Figure 7-1 A Transaction Slot

7.2 CONTEXT REQUIREMENTS OF FILE ACCESS

Although it is TSTs that read, lock, and update records in data files, a transaction processor cannot associate these activities with the TSTs that actually do them. Instead, the activities must be associated with the transaction instances whose exchange messages are being processed.

An example may clarify this point. In the Change Customer transaction we have been using for an example, a TST in the first exchange must read and lock a record in the customer file. A TST in the second exchange will subsequently update that record and release the lock. In an ordinary system using stand-alone programs, only the program that issues a read and a lock can subsequently issue the update. This is insufficient for a transaction processor, since two programs executing at different times must cooperate to achieve the update: one to read and lock the record, the other to update it.

The need for extensive context in file operations is more evident when we look closely at the TST that reads and locks records. This TST is not serving one application terminal alone; it will process exchange messages coming from many terminals for many different transaction instances. The fact that the TST has locked a record for one user does not mean that it will access that same record for another user; in fact, the opposite is true: it must *not* access a record for a second user if it has read and locked that same record for a previous user.

For these reasons, TRAX transaction processors keep track of file access operations by *transaction instances* rather than by TSTs. This is possible because the file access operations are handled by a central file access facility within the transaction processor, and it is this facility that operates the file access and locking mechanisms.

CHAPTER 8
APPLICATION DATA FILES

A carefully designed, efficient set of data files is a necessity in any transaction processing application. Poor file design will have many effects on such an application, from poor performance to total loss of data. This chapter discusses the TRAX facilities for file access from TSTs and presents design considerations that you must analyze when you design data files.

8.1 RMS

RMS is a data management system that is available for several operating systems on the PDP-11 family of processors. It provides extensive facilities for the reading, writing, and insertion of records in many file organizations.

Application data file access under TRAX is done through RMS. This provides file structures that are compatible between transaction processors and support environment programs and also permits the transfer of data files between a TRAX system and other operating systems that support RMS.

On TRAX systems, RMS provides access to three main categories of files:

- **Sequential files.** Sequential files are accessed by starting with the first record in the file and proceeding to subsequent records. These files are good choices if an entire file is to be processed at one time or if the records are arranged in the file in the order they will be needed. (If individual records must be found at random within the file, another file organization would be better.)
- **Relative Files.** In these files, each record is identified by a number. The first record in the file is labeled "one," the next "two," and so forth. Records may be retrieved from such a file by specifying record numbers. Any record may be retrieved with low overhead, as long as its record number is known. This file is a good choice for retrievals and updates, so long as some scheme can be devised to calculate the record numbers. These files can also be read sequentially, like a sequential file.
- **Indexed Files.** This type of file is the most flexible file organization supported by TRAX. Each indexed file has one or more indexes, and each index enables the programmer to retrieve any record from the file once its index entry is known. For example, if a list of customers were placed in an indexed file, any customer's record could be retrieved once its customer number was known.

  The efficiency of retrieval varies from file to file within this group, depending on factors such as record size, index entry size, and file activity patterns.

  Indexed files can have one or more indexes. Each index has an entry for every record in the file, so different indexes provide different ways of finding the same records. For example, a customer file might have two indexes: one by customer number and another by customer last name.

The first index is called the *primary index.* Each record must have a *unique entry* in the primary index; it is this entry that identifies the record within the file. In our example, the index by customer number would be the best choice for the primary index.

Other indexes are called *secondary indexes.* The entries in these indexes *need not be unique;* that is, several records in the same file can have identical entries. For example, in a customer file the last name index would be a secondary index, and the file organization would allow several customers to have identical last names.

Records in an indexed file can be retrieved in two ways:
- By a key value in a specified index
- Sequentially, beginning *with a previously retrieved record and proceeding according to the index used for the previous retrieval.*

For example, we could retrieve a specific customer record from the customer file by specifying a customer number. Having done that, we could retrieve additional customers in customer number order.

Indexed files are good choices where random retrievals are necessary and record numbers (such as would be needed with a relative file) would be hard to calculate. They are also the best choice in situations requiring record insertion between existing records. But in situations where one of the other organizations will suffice, they would be a better choice because they will generally be more efficient.

For a detailed discussion of the features and capabilities of RMS, consult the RMS-11 reference manuals.

8.2 FILE ACCESS FROM TSTS

Each of the TRAX programming languages has its way of accessing RMS files:

- For COBOL and BASIC-PLUS-2, this interface consists of the language's standard file access verbs. That is, the application programmer can use the usual file access statements and clauses.
- For MACRO-11, a set of macro instructions is provided that generate the proper subroutine calls for linkage to RMS.

Tasks that use RMS to access files must include RMS support code in their task image. In a TRAX transaction processor, most of the RMS support code is kept in the transaction processor's common area. This minimizes code that must appear in each TST's task image. Only simple linkage routines appear in the TST, while the bulk of RMS remains in a separate task that supports all TSTs at one time. The simple linkage routines are inserted in each TST task image (if required) by the TSTBLD utility program.

For a detailed discussion of the interface between a specific programming language and RMS or for details of the linking procedures required for RMS, consult the appropriate language user manual and language reference manual.

8.3 WORK FILES

When a designer thinks of files and file access, he is usually thinking of permanent data files that are accessed and updated by many simultaneous users.

In many applications, another type of file is also required. This file, called a *work file*, is created and used by a single user in the course of a transaction instance. A transaction, for example, might require a large scratch area in which to assemble user input. If enough room is not available in the transaction workspace, a file could be used instead. This file would be created for the transaction instance, used as a scratch area, and then destroyed upon transaction termination.

Transaction processors allow TSTs to create, access, and delete work files much the same as permanent data files. Work files must conform to special rules, however:

- Work files must be defined with the FILDEF utility, just as permanent files are defined. Part of the definition declares the file to be a work file.
- When a work file is defined, it is given a logical file name and a physical file specification. The file specification, however, does not include a version number. *Each time a work file is created, a new physical file is generated having a unique version number.*
- After the first TST in a transaction instances creates a work file, other TSTs accessing that logical file will have access to the same physical file. In other words, only one physical work file of a given logical name can be created by a transaction instance.
- A TST cannot access any work file that its current transaction instance did not create.
- A TST can discover the file specification (including version) of a work file it is accessing by issuing an appropriate system call. (See the *TRAX Application Programmer's Manual* for further information.)
- After discovering the file specification for a work file, a TST can pass that specification to a support environment program through the usual interface methods (Chapter 11).
- The TST programmer uses the usual language verbs to create and delete work files.

8.4 RECORD LOCKING

For proper operation of a multi-user transaction processing system, some means of controlling access to individual records in shared files is essential. Without this control, users might attempt to update the same record simultaneously, and the effect of at least one of the updates could be lost.

TRAX transaction processors implement this control with a record lock facility. Any TST can lock a record for the transaction instance it is processing. When it does, access by any TST processing a different transaction instance (including even that same TST, if it subsequently begins to process another transaction instance) will be restricted. The restriction is removed when that TST or another TST, again processing the same transaction instance, explicitly releases the lock, or when the transaction instance terminates.

Depending on your specifications for a particular file, the restriction imposed by a lock is either:

- No TST processing another transaction instance can read or update the locked record, or
- No TST processing another transaction instance can update the locked record – but any TST can read the record.

Remember that the lock is imposed for the transaction instance and not for the TST itself. This allows one TST to read and lock a record, and a second TST in a later exchange to update and unlock the record for the same transaction instance. Conversely, the first TST cannot access the locked record for another transaction instance even though it locked the record for the first transaction instance.

Record locks are enforced and arbitrated by system software within the transaction processor. This means that access conflicts between TSTs in the same transaction processor will be correctly handled; but it also means that there is no way of resolving access conflicts between TSTs in two transaction processors or between a TST and a support environment program. For this reason, only a single transaction processor *or* a single support environment program can have update access to an RMS file at one time.

The application programmer controls the locking and unlocking of records by using LOCK and UNLOCK clauses in file access statements.

If a TST encounters a locked record, the unsuccessful file access operation will be retried after a short period. (This period is specified when the file is defined.)

If the record remains locked after the second try, the file access statement will return an error code. If the transaction was defined to include exchange recovery, and if the transaction instance has no outstanding record locks from other exchanges, the TST can issue an exchange recovery request to restart the processing of the exchange. This will release *all* the locks currently held by the transaction instance, and the exchange processing will start over.

8.5 STAGING

Staging is the delay of inserts, deletes, and updates to a file until the transaction instance requesting them terminates. TRAX supports staging, and each file used by a transaction processor must be specified as staged or unstaged.

- **Staged Files.** Staged files are files where update operations are automatically postponed until the transaction instance that requested them terminates normally.
- **Unstaged Files.** Unstaged files are files where update operations take effect immediately.

The principal benefit of staging is that staged operations will not be applied against a file if the transaction instance that requested them terminates abnormally – that is, if it aborts. An aborted transaction instance will have no effect on staged files, because the operations it has requested have not been done on the file.

Staging usually lengthens the time that records are locked. This is necessary because the records must remain locked until they are *updated,* not just until the update request has been made by a TST. In other words, the transaction processor must prolong a record lock during the time an update is being delayed. In some cases, this prolonging of record locks may be significant; in any event, you must consider this effect whenever you specify a staged file.

Because staging separates the *request* for an update operation from its *execution,* staging may confuse some application programmers unless you are careful to explain the use of staged files. You must explain that if a programmer updates a record in a staged file, there is *no way* to release the lock on that record before the transaction instance terminates. This includes explicit "release-lock" statements in the programming language. However, explicit "release-lock" statements *will* take effect, even on staged files, if the record in question has only been read and not updated.

Staged records take considerable space in the system workspace. Each staged record (that is, a record destined for a staged file) is saved in the system workspace until the end of the transaction instance. The staging of a large number of records during any transaction instance will require the definition of a correspondingly large system workspace, with a corresponding increase in the memory requirements of the transaction processor.

For efficient use of a system's memory, you should design transactions that stage a predictable number of records. This way, your system workspaces will always be as small as practicable, and they will be used to their greatest capacity.

8.6 DATA FLOW DURING FILE ACCESS OPERATIONS

When a TST issues a file access instruction, the flow of control and data is different from many other systems. This flow has been carefully designed to optimize flexibility, efficiency, and compatibility with other systems. As an application designer, a knowledge of this flow will help you to design more efficient files and file access procedures.

8.5.1 Data Flow During a Read

Figure 8-1 shows a typical flow of data during a simple RMS read operation requested by a TST. In this example, assume that a TST has requested a read operation and the data is to be placed in the transaction workspace. The read operation would proceed as follows:

- The TST makes the file read request; that is, an appropriate READ statement is encountered in the program.
- Linkage routines included in the TST task image handle the read request and transmit it to the transaction processor.
- The transaction processor system software includes a copy of RMS, and the request is given to the appropriate module.
- The RMS module reads the record from the file, using its own work buffers.
- When the record has been read and deblocked, it is copied into the designated destination buffer – in this case, the transaction workspace.

8.5.2 Data Flow During an Unstaged Update

Figure 8-2 shows the data flow that occurs when a TST requests an update operation on an *unstaged* file. The data flows exactly the reverse of the "read" example in Section 8.5.1, traveling from the source data structure to the file through the actions of the TST, the TST's resident linkage routines, and the common RMS code. The source data structure in this example is the transaction workspace but it could be any data structure in the TST.

8.5.3 Data Flow During a Staged Update

Figure 8-3 shows the data flow that occurs when a TST requests an update operation on a *staged* file. Like the previous example, the source of the data is the transaction workspace, although it could be any other data structure in the TST.

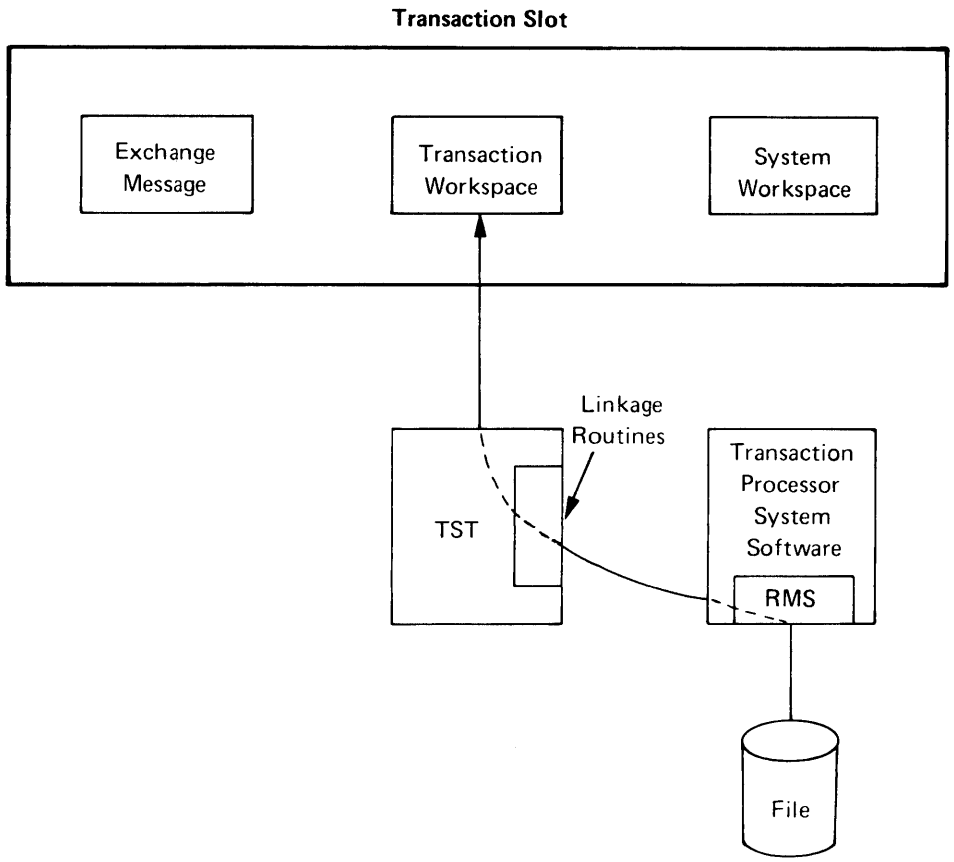


Figure 8-1 Flow of Data During Read

As you can see, the data flows from the source data structure to the file in two steps:

- In the first step, the TST issues the update request with an appropriate language statement. The TST's resident linkage routines pass the request to the transaction processor, as with the other examples. But the transaction processor notices that the file in question must be staged, and so the data is not given to RMS to be written in the file. Instead, it is saved temporarily in the transaction instance's system workspace. The TST is given a successful return code, as if the record had been written in the file.
- The second step occurs when the transaction instance terminates normally. When this happens, the transaction processor takes each record saved in the system workspace and gives it to RMS to be written in the file. This process is invisible to the TSTs in the transaction instance; they were not aware that the record was staged.

If a transaction instance terminates abnormally (that is, it is aborted for some reason), the second step never happens. The updates to the file are never applied, and the transaction instance has no effect on the staged files.

8.7 JOURNALING

For those applications that require a record of data file updates, or that require the reconstruction of certain data files after a system crash, TRAX provides a journaling facility.

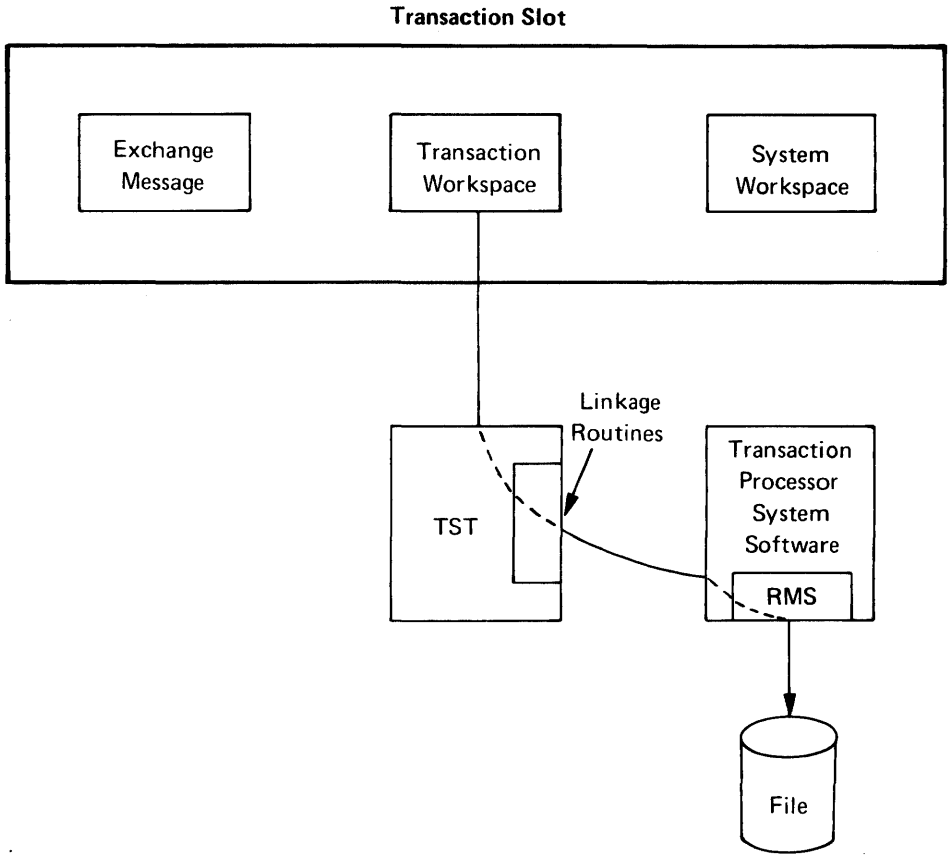


Figure 8-2 Flow of Data During Nonstaged Update

The mechanism of this journaling facility is simple and efficient. As you know, the transaction slot contains a complete picture of the state of its corresponding transaction instance. It contains the most recent exchange message, the transaction workspace, and the system workspace. As you have read, the system workspace contains all records that are destined for staged files. TRAX can therefore achieve a complete journaling facility in the following manner:

- Any file that is to be journaled is automatically staged. All updated records destined for this file will therefore be kept in the system workspace pending successful termination of the transaction instance.
- The transaction slot is written to a journal device at the end of the transaction instance, before staged records are written in their respective files. This preserves status information about the transaction instance at its termination. This includes, of course, copies of all staged records.

When you specify a transaction processor's files, you must specify whether each file is to be journaled or not. Each journaled file will automatically be staged (Section 8.5) so that updated records destinated for that file will appear in the system workspace at transaction instance termination.

Because of the staging requirement for journaled files, an aborted transaction instance has no effect on journaled files. The transaction has aborted before the journal entries could be made. The journal thus agrees with the actual updates to journaled files.

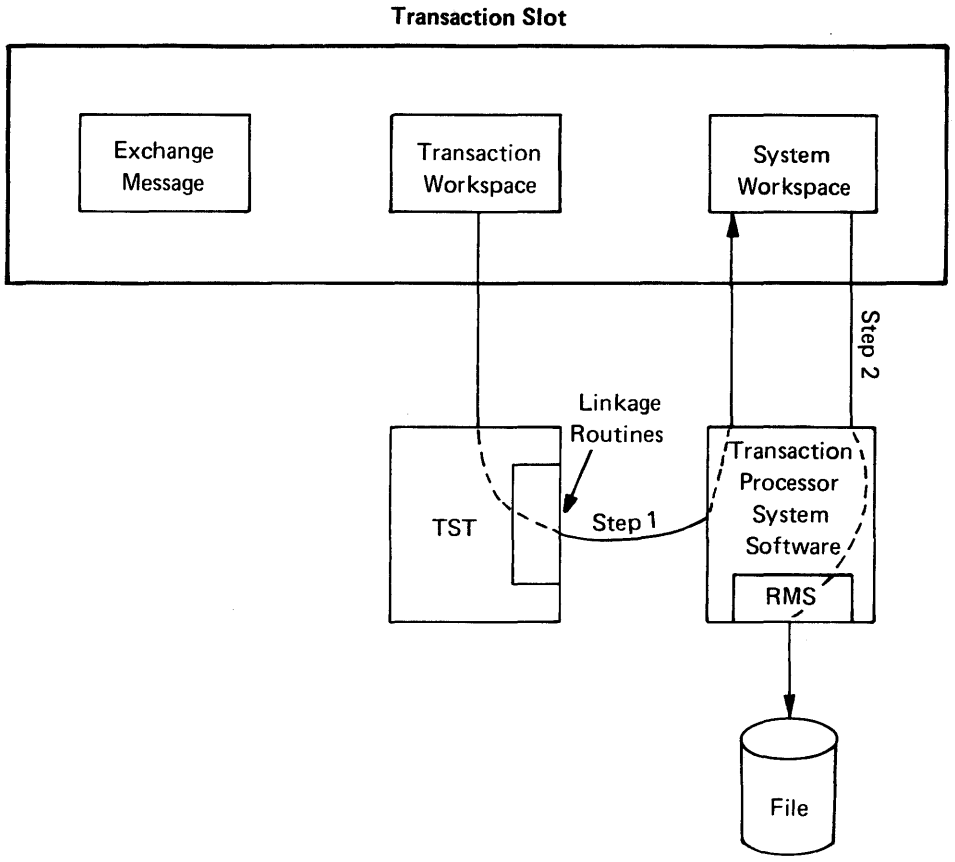


Figure 8-3 Flow of Data During Staged Update

Journal entries are only made for those transaction instances that make at least one update to a journaled file. Other transaction instances do not have their transaction slot journaled when they terminate.

Journal entries are written in the native mode of the journal device. That is, the journal data is written without logical file structure other than that provided by the hardware itself. This is to ensure the maximum device reliability; a logical file structure on the journal device would open the possibility of data loss should that file structure be corrupted.

The operator of a TRAX system can specify a primary and a secondary journal device:

- **The Primary Journal Device** is used first. When it is full, the system will shift automatically to the secondary journal device, if one has been designated. Otherwise, transaction processing will stop, while the journal device is taken off-line and another tape or disk mounted.
- **The Secondary Journal Device** stands ready to relieve the primary journal device when it becomes full. The secondary journal device accepts journal entries until it is full, then the primary device again takes over.

Both magnetic tape and disk devices may be used for the journaling devices. While it is designated as a journaling device, a device cannot be used for any other purpose.

8.7.1 Reconstructing Journals

A special utility program, RECOVR, is provided with each TRAX system. This utility program can reconstruct application data files, given a backup copy of that data file and all journals from that point to the time of a crash. It reads through the journals, locates the updated records in the system workspace of each transaction slot, and applies the updates.

8.8 LOGGING

A TST can write data into the system journal independently of the automatic journal entries made by the transaction processor. This process is called *logging.*

Log entries can have any format desired by the application programmer. When a log entry is ready, the TST must issue the appropriate system call. The log entry will be saved in a buffer and when there are enough log entries to warrant setting the journal device in motion, the blocked log entries will be written as a single journal entry.

8.8.1 Inspecting and Analyzing Log Entries

A special utility program, SHOLOG, is provided with each TRAX system as an aid to the inspection and analysis of log entries. This utility program reads a journal, selects specified log entry classes, and prints or displays them.

Log entries can be useful for system debugging and as an audit trail of system operation. When used for system debugging, log entries can provide a record of when TSTs are activated; they can also record any data that the TST programmer thinks might be helpful. When used as an audit trail, log entries can record the relevant data for each execution of a sensitive transaction. If properly done, logging can thus ensure that the transaction cannot be executed without creating a record of its execution.

CHAPTER 9
INITIATING TRANSACTION INSTANCES

Little has been said so far about how transactions are invoked – that is, how transaction instances are initiated.

There are several methods by which transactions can be invoked. Most of these involve application terminals, but there are methods that allow transactions to be invoked without any terminal. This chapter describes all of these methods.

9.1 INITIATING TRANSACTION INSTANCES FROM AN APPLICATION TERMINAL

Most transaction instances will be initiated from application terminals. This section describes the three methods by which this can be done.

9.1.1 Terminals That Can Invoke Only One Transaction

When a transaction processor is defined, some terminals are defined so that they execute only one transaction. When no transaction instance is active, the terminal displays the form from the first exchange of its transaction.

The display presented at a terminal when it is not involved in a transaction instance is called the *initial operating mode* of the terminal.

When you define a terminal station within a transaction processor, you must specify an initial operating mode for its associated terminal. If you specify a *transaction definition* as an initial operating mode, that terminal executes only that transaction and displays the form from the first exchange of the transaction when the terminal is idle.

A user initiates a transaction instance from this terminal by filling in the displayed form and sending the data to the system. An instance of the transaction is then underway.

When the transaction instance terminates, the terminal reverts to a display of the form from the first exchange of the transaction.

9.1.2 Terminals That Can Execute Several Transactions

This class of terminals can initiate any of a group of transactions. The transactions will be available to each user that sits at the terminal.

To get a terminal to operate in this way, you must specify a *form definition* as the initial operating mode when you define the terminal station. This form definition must include the SELECT clause in its FORM statement, thus designating one of the fields on the form to contain the selected transaction name. Such a form is called a *transaction selection form*, because its purpose is to collect the name of a transaction from the user.

The transaction selection form will not be part of any transaction definition. It is displayed whenever the terminal is idle (that is, whenever a transaction instance is not active at that terminal).

To invoke a transaction from such a terminal, the user must fill in the name of the desired transaction on the transaction selection form. This will be the short name (six characters or less) by which the transaction definition is known within the transaction processor. (If the form is defined with Menu fields instead of Input fields, the user may select one of the displayed transaction names rather than type in the transaction name.)

Each of these terminals is restricted to a list of transactions. These are the only transactions that can be executed from the terminal, irrespective of the transaction names listed on the transaction selection form or entered by the user.

A terminal is assigned this restriction list by specifying a *work class* for the terminal. A work class is a list of transactions, identified to the transaction processor by logical name of six or fewer characters. The terminal's work class is assigned when its terminal station is defined.

So, for example, a terminal might be assigned to work class CLASS2. If this work class contained the names of three transactions – ADDCUS, CHGCUS, and DYPCUS – these would be the transactions accessible from that terminal. Other transactions would not be accessible, whether or not they were listed on a transaction selection form or entered by the user.

In certain circumstances, you may want a transaction selection form to collect data in addition to the transaction name. You can do this by adding ordinary Input fields to the transaction selection form and using them to define an exchange message. If the transaction selection form generates an exchange message, this exchange message will be used for the first exchange of the selected transaction and that exchange's form will not be used.

In any case, once the transaction name has been entered or selected, the user sends it to the system. Except in the one situation described above, the system responds by displaying the form for the first exchange of that transaction.

The transaction instance will begin as soon as the user fills in this first form and sends the data to the system.

When the transaction instance ends, the transaction definition may specify either FIRST or INITIAL as the subsequent action. (See Chapter 4 for more information about transaction definitions and the subsequent action parameter.) If the parameter is FIRST, the terminal will display the first form again and prepare for another instance of the same transaction. If the parameter is INITIAL, the terminal will display the transaction selection form and the cycle will begin again.

9.1.3 Terminals That Require User Sign-On

This class of terminals can execute only a limited set of transactions until a user identifies himself to the system with a user identification and password. This process of identification is called *signing on.*

NOTE

The process of signing on, which involves an application terminal user and an application terminal, is not related to the procedure by which operating or programming staff identify themselves at support terminals. The latter process uses a different set of identifying codes and passwords.

This kind of terminal can be in any of three states:

- **Idle, Signed Off.** In this state, there is no transaction instance active at the terminal and a user has not identified himself to the system.
- **Idle, Signed On.** In this state, the terminal still does not have a transaction instance active, but a user has identified himself to the system.
- **Executing a Transaction Instance.** In this state, the user has selected a transaction and is executing it.

In the first state, idle but signed off, the terminal behaves in a manner identical to the terminals described in Section 9.1.2. The transaction selection form and the terminal's work class govern the set of transactions that can be executed. If a user selects a transaction that is permitted by the work class, the terminal moves directly to the third state – executing a transaction instance – without passing through the second state.

In most cases, the only transaction you will include in the terminal's work class is the SIGNON transaction. This transaction and all its supporting TSTs are supplied as part of each TRAX system. When the user executes this transaction, he is asked to enter a user identification and password. If he does this correctly, he will become signed on and the terminal will enter the second state.

While the terminal is in the second state, the terminal's work class does not apply. Instead, the user will have his own work class or classes, and he will be allowed to execute any transaction listed in those work classes. He will be able to execute these transactions even if the original terminal work class would have prohibited them.

The same transaction selection form remains on the screen after the user has signed on. The terminal has only one transaction selection form, and this form is used in both the first and second states – Idle and Signed Off, and Idle and Signed On. Only the applicable work classes change.

In either the first or second states, users initiate transaction instances just as was described in Section 9.1.2. The user chooses one of the available transactions by entering a transaction name or selecting its name on a menu; then the transaction processor displays that transaction's first form. The transaction instance begins when the user fills out the first of the transaction's forms and sends it to the system.

When a transaction instance terminates, the same two subsequent action parameters apply as in Section 9.1.2. If the parameter is set to FIRST, the same transaction is executed again. If the parameter is set to INITIAL, the transaction selection form will be displayed. In either case, the terminal will revert to the same state (signed on or signed off) that it was in before the transaction instance was begun.

If a user has signed on but wishes to leave the terminal, he executes the SIGNOF transaction. The SIGNOF transaction is also supplied with each TRAX system. When it is executed, the terminal reverts to the first state (idle but signed off) and the terminal's work class again applies.

NOTE

The terminal's work class *must* include the SIGNON transaction, and each user's work class *must* include the SIGNOF transaction. Otherwise, the terminal will not operate properly.

9.2 INITIATING TRANSACTION INSTANCES IN OTHER WAYS

Besides application terminals, there are three other ways of initiating transaction instances.

9.2.1 Spawned Transactions

A TST may initiate a transaction instance while processing an exchange message. This procedure, called *spawning,* creates a new and independent transaction instance. The original transaction instance (the one whose exchange message was being processed) continues.

The TST supplies an exchange message when it spawns the new transaction instance. The instance is limited to a single exchange.

The TST that spawns a transaction instance does not wait for the spawned transaction instance to terminate, and so the spawned transaction instance may not return a response message to the TST.

Spawning a transaction is a useful technique in a variety of situations. For example, a transaction might be spawned to execute a series of processing steps that require significant time. In this way, the user's terminal is not tied up for a long period.

9.2.2 Support Environment Programs

A program running in the support environment initiates a transaction instance in much the same way as a TST. As with transaction instances spawned by a TST, these transaction instances are limited to a single exchange. The exchange message for that exchange is supplied by the support environment program at the time it initiates the transaction instance.

Each active transaction instance initiated by a support environment program uses a *slave batch station.* This station serves the same purpose as a terminal station for transaction instances initiated by an application terminal. The number of transaction instances that can be simultaneously initiated by support environment programs depends on the number of slave batch stations in the transaction processor.

Because of the slave batch stations, transaction instances initiated through such stations may send response messages back to their source stations. These response messages are forwarded to the support environment program that initiated the transaction instance.

The support environment program is suspended while the transaction instance executes and is activated again when the response message is returned.

This technique is useful when a support environment program (a batch program, for example) must acquire some data via a transaction in a transaction processor. The technique also allows a support environment program to print data on a transaction processor's output-only terminals by activating an appropriate transaction.

Data communication between a transaction processor and programs in the support environment is discussed in detail in Chapter 11.

9.2.3 Other Transaction Processors

Another transaction processor, either within the same TRAX system or in another system, may request that a transaction instance be initiated. This technique is used to advantage, for example, in a distributed processing network. Each node of the network initiates transactions in other nodes. Transaction instances initiated in this way can have multiple exchanges. For each exchange, the initiating transaction processor supplies an exchange message.

Each exchange message is fed to a *slave link station* in the second transaction processor. The slave link station serves the same purpose as terminal stations, initiating each exchange and waiting for the corresponding response message.

When the response message is received, it is forwarded to the initiating transaction processor. That transaction processor terminates the transaction instance or sends another exchange message.

The number of slave link stations in a transaction processor determines how many transaction instances can be initiated at one time by remote transaction processors. The number of slave link stations associated with each remote transaction processor determines how many transaction instances that remote transaction processor can initiate at one time in the local transaction processor.

This technique is useful where one transaction processor acquires data held by another transaction processor, perhaps in a remote system. For example, a transaction on a local transaction processor may permit users to display inventory data. This may require the local transaction processor to interrogate several remote transaction processors to determine the inventory at remote locations. This technique can also be used to post local work to remote files or to print data on remote output-only terminals.

Data communication between transaction processors is covered in detail in Chapter 11.

CHAPTER 10
SECURITY, RELIABILITY, AND PERFORMANCE

Tight security, excellent reliability, and good performance are requirements in most transaction processing systems. In previous chapters, you learned many of the characteristics that help TRAX applications meet these requirements. In this chapter, these characteristics are reviewed and their contribution to system security, reliability, and performance is discussed. Several more features of TRAX are introduced that also contribute to satisfactory transaction processing.

10.1 SECURITY

A secure system is one where each user is restricted to the functions he is meant to use and can only access the data he is meant to see.

Several TRAX features contribute to the security of transaction processing applications built on it.

10.1.1 Application Terminals and Support Terminals

The most serious breach of system security occurs when a user gains unauthorized access to the operating system and its terminal commands. In TRAX, this is unauthorized access to the support environment. If a user gains access to the support environment, he can instruct the system to copy data files, alter programs, and even alter or halt the operating system itself.

TRAX prevents this by keeping application terminals and their machine interfaces strictly and physically separate from support environment terminals and their machine interfaces. Once the system has been properly generated and communication lines properly installed, it is impossible for an application terminal to gain access to the TRAX support environment. To do so would require the physical reconnection of the terminal to a different communication line and machine interface.

Everything that occurs at application terminals is under the control of transaction definitions and form definitions. Any function that has not been defined as a transaction simply cannot be executed from an application terminal.

This places TRAX apart from other on-line systems where a single terminal serves as both a programmer's console and an application terminal. In those systems, users can often "escape" from an application program and use the programmer's system command language. This is impossible under TRAX, because an application terminal has no control over the system.

Security against unauthorized access depends only upon the physical security surrounding a system's support terminals, their communication lines, and their machine interfaces. If physical access to these components is not allowed, security of the operating system and application programs is assured.

10.1.2 Work Classes and Signing On

Any application is likely to have transactions that need some security arrangement. For instance, a transaction that displays a customer's credit records should be more restricted than one that

displays outstanding orders. This means that some users should be allowed access to a transaction, while others should be denied access.

Chapter 9 discussed work classes, how they are assigned to particular terminals, and how they are assigned to particular users. These work classes, together with the SIGNON and SIGNOF transactions supplied with each TRAX system, give you the tools you need to implement almost any transaction security arrangement.

And, once these restrictions are in place, they cannot be defeated from an application terminal because of the distinction between application and support environment terminals. To change the work class assigned to a terminal or a user, someone must first gain access to a support environment terminal. Even the privileged, knowledgeable system programmer can not make such changes from an application terminal.

10.1.3 Terminals Running a Single Transaction

Chapter 9 also described a terminal restricted to a single transaction. No other transactions could be executed from that terminal, no matter how privileged or knowledgeable the user.

This feature is useful for terminals in insecure locations. A terminal placed in a bank lobby, for example, might allow customers to check their account balances. This terminal could be connected to a system that has sensitive customer data and privileged transactions. But so long as the initial operating mode of the bank lobby terminal limits that terminal to the one transaction, no breach of system security can occur. In fact, not even the system designer himself could defeat the security restrictions of the bank lobby terminal.

10.1.4 Logging

Although it will not *prevent* unauthorized access, the logging facility described in Section 8.8 can be used by an application programmer to *record* each use of a transaction and therefore detect unauthorized use.

The log entries are written on the system journal device and cannot be accessed from an application terminal.

10.2 RELIABILITY

A reliable system is available when needed, and its data loss is infrequent.

Several TRAX features contribute to the reliability of applications, in addition to the recognized reliability of the PDP-11 processors on which it runs. Some of these features are outlined in the following paragraphs.

10.2.1 Exchange Recovery

Because a picture of each transaction instance is maintained in its corresponding transaction slot (see Chapter 7), TSTs can recover from certain errors during the processing of an exchange.

If you specify exchange recovery for a transaction, the transaction slot will be copied to disk at the start of each exchange. Then this information can be used (at a TST's request) to restart the transaction instance from that point. For example, a long-term record lock encountered by a TST could prompt the TST to restart the exchange.

Exchange recovery is invisible to the user at an application terminal, except that response times may be slightly extended during exchange recovery attempts.

10.2.2 Crash Recovery

System software errors and hardware errors can cause a TRAX system to crash. When this occurs, the system is automatically reinitialized in the following way:

1. The operating system kernel restarts itself if necessary.
2. Each transaction instance that was active at the time of the crash is terminated. The effect on a transaction instance is the same as if an ABORT key had been pressed at the moment of the crash. These effects will vary depending on whether or not the transaction uses exchange recovery and whether it accesses staged or unstaged files.
3. Unstaging is completed for each transaction instance that was in the process of unstaging when the crash occurred.
4. The transaction processors that were active at the time of the crash are restarted.
5. The application terminals attached to each active transaction processor are reinitialized. They will be placed in the same state as when the transaction processor is first started. That is, each terminal will display either the first form of a transaction or a transaction selection form. At those terminals that require user sign-on, users must re-enter their identification codes and passwords.

Crash recovery is a feature that you select or decline for each transaction processor. If selected, it adds a slight overhead to transaction processing to maintain the transaction status table. The effectiveness of crash recovery depends on your use of two other TRAX facilities, staging and exchange recovery.

10.2.3 Data File Recovery

Rarely, a system interruption destroys application data files. This kind of failure is typically a severe hardware failure, such as a head crash on a disk.

In failures of this sort, it is imperative that data loss be held to a minimum and that the manual effort of reconstructing data be limited. The TRAX data file journaling facility, described in Section 8.7, meets this requirement. Using a backup copy of a data file and journals that represent the processing against that file since the time of the backup, a TRAX system can automatically recreate the data file as it appeared immediately before the failure.

The effectiveness of this facility depends on your file design and the selection of journaling for appropriate files. Journaling a file introduces a slight overhead in the processing of transactions.

10.3 PERFORMANCE

A system that performs well processes transactions quickly enough to accommodate workload and quickly enough to satisfy its users.

Several TRAX features contribute to performance in applications built upon it: some are outlined here. Good performance, of course, depends on good application design, as well as the facilities of the operating system.

10.3.1 Record Locking

TRAX enforces file access on a record basis. A transaction instance can lock one record in a file, while allowing other transaction instances access to other records. This contributes to system performance by reducing the effects of interaction between transaction instances.

10.3.2 Internal System Design

Many design features of TRAX were selected for their beneficial effect on system performance. For example, the messages that travel from station to station in the transaction processor do not really move in a physical sense; they remain in one area of system memory and they are "attached" to various processing modules by memory mapping techniques. This procedure (which is transparent to the application programmer) is more complex than moving the data from place to place, but it improves system efficiency.

10.3.3 Caching

TRAX transaction processors make effective use of the available disk access capability through caching, which is the intermediate storage of disk-resident data in buffer areas.

With the caching technique, a data structure fetched from disk is not discarded when it is no longer needed. It remains identified in a buffer. If the data structure is needed later and is still in the buffer, the buffer copy is used instead of reading the data structure from the disk.

When data structures are written, they are updated on the disk *and* in the buffer. The cached copy of the data structure and the disk-resident copy therefore match at all times.

Data structures are purged from the buffer as space is needed for other data structures. This purging is done according to a least-recently-used algorithm, so the data structures purged are least likely to be needed soon.

The caching technique applies to most data structures on the disk. These include:

- Transaction definitions
- Form definitions
- Data file records
- Data file indexes
- Messages needing long-term storage
- Transaction slots
- TST task images

CHAPTER 11
TRANSACTION PROCESSORS AND DISTRIBUTED PROCESSING

As you have read, transaction processors are well adapted for centralized on-line transaction processing applications. But the architecture of transaction processors also permits interfacing them to systems and subsystems for various kinds of distributed processing.

Interfacing a transaction processor to other systems or subsystems demands that you have a clear understanding of transaction processors. Distributed processing is a more advanced technique than the design of a simple, stand-alone transaction processor. For this reason, interfacing to other systems and subsystems has not been discussed in earlier chapters.

Three interfaces are possible between a transaction processor and other systems or subsystems:

1. An interface between a transaction processor and the support environment of the TRAX system where it runs
2. An interface between two transaction processors, either running on the same TRAX system or on two TRAX systems
3. An interface between a transaction processor running on a TRAX system and a similar subsystem running on a different operating system

This chapter discusses the general capabilities of these interfaces. Consult the *TRAX Application Programmer's Guide* and the *TRAX Support Environment User's Guide* for details about the exchange message formats, the support environment features, and the success and failure codes for each operation.

11.1 INTERFACE BETWEEN TRANSACTION PROCESSORS AND SUPPORT ENVIRONMENT

Figure 11-1 shows a TRAX system. This system has a transaction processor and a support environment. Within the support environment is a batch processor utility program and an application program (perhaps controlled by a different batch processor or a support environment terminal.)

There are two possible interface paths between a transaction processor and the support environment (Figure 11-1): one path is initiated by the transaction processor and the other by the support environment application program.

11.1.1 Path Initiated by the Transaction Processor

The interface shown in the upper portion of Figure 11-1 shows a path initiated by the transaction processor.

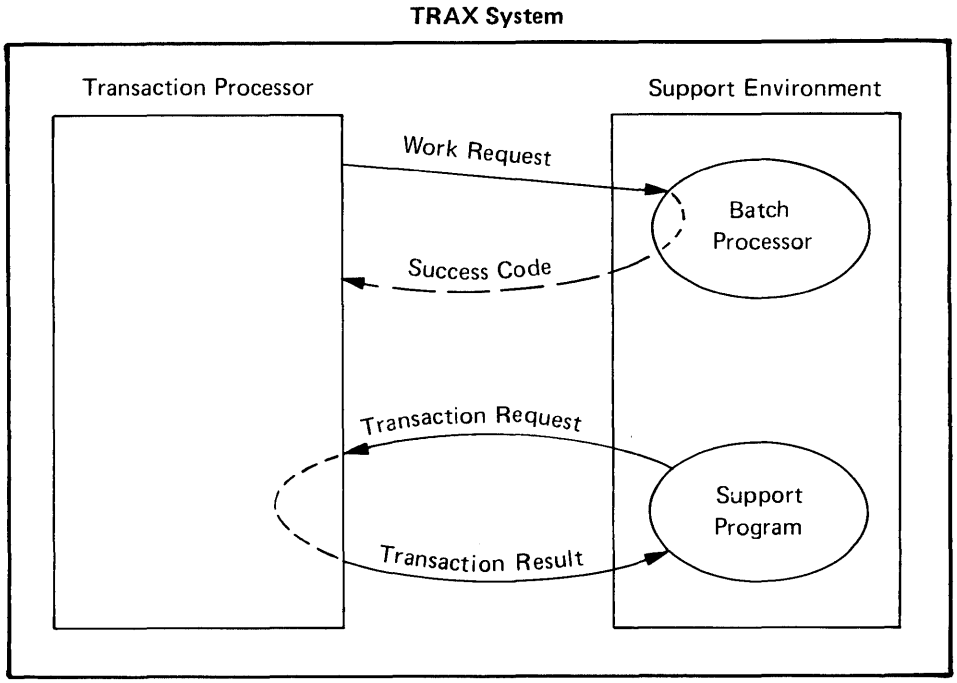


Figure 11-1 Interface Between a Transaction Processor and the Support Environment

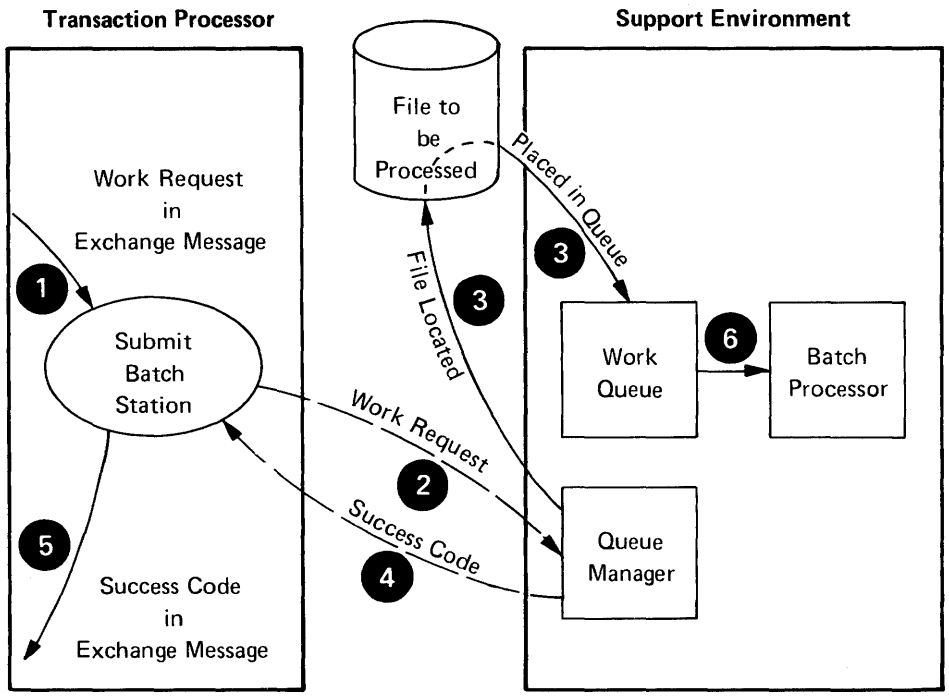
A transaction processor's only method of interface with the support environment is the submission of batch processor requests. (For information about batch processors, see the *Introduction to TRAX* and the *TRAX Support Environment User's Guide.*) This is a unidirectional interface; the support environment can return only a simple success code indicating that a request has been accepted.

The interface is accomplished via a *submit batch station* in the transaction processor and the queue manager utility program in the support environment. When an exchange message is routed to a submit batch station, the station looks for a legal SUBMIT command. (SUBMIT commands to enter batch or work requests are normally issued at a support environment terminal.)

The command is passed to the queue manager in the support environment, and the queue manager checks the validity of the command before entering the work request on the work queue. The queue manager passes a success code to the submit batch station, which places it in the exchange message. The success code indicates whether the work request has been successfully entered. (See more about the success code in the *TRAX Application Programmer's Guide.*) This code is usually inspected by a "downstream" TST where the exchange message is subsequently routed. Figure 11-2 shows the process.

NOTE

The success code does not indicate that the work request has been *executed*, only that it has been *entered* in the queue.



Processing Steps:

1. Work request arrives at a submit batch station in the form of an exchange message.
2. Submit batch station forwards request to the support environment queue manager.
3. Batch command file is located and placed in corresponding queue.
4. Success or failure code is returned to submit batch station.
5. Success or failure code is placed in exchange message, and exchange message is passed to next processing station.
6. Some time later, file works its way to head of batch processor queue and is processed.

Figure 11-2 Interface Initiated by a Transaction Processor

It may be some time before the request is processed; in fact, the support environment batch processor may not be running when the request is entered. The transaction processor waits only for the successful entry of the request in a queue, not for its execution.

11.1.2 Path Initiated by a Support Environment Program

The interface in the lower portion of Figure 11-1 shows a path initiated by a program in the support environment. This could be a program run at a support environment terminal or by a batch processor in the support environment.

This is a bidirectional interface. The support environment program requests that a transaction be executed in the transaction processor. The program awaits the results of the transaction, which are returned in a response message and then forwarded to the support environment program. This interface contrasts with that in Section 11.1.1, where the transaction processor *could not* wait for results from the support environment processing.

The support environment program makes the transaction initiation request by using a system call. At the end of the transaction instance, up to 24 bytes of the response message generated by one of the transaction TSTs are returned to the calling program.

Support environment programs may initiate only single-exchange transactions. The exchange message provided by the support environment program must be formatted properly for the transaction TSTs.

A support environment program can initiate the same transactions that can be initiated from an application terminal; when initiated from the support environment, form names in the transaction definition are ignored.

The process is shown in Figure 11-3. The station in the transaction processor that receives the exchange message contents from the support environment program, routes the exchange message to the processing stations, and forwards the response message to the support environment is called the *source station.* For these transactions, a *slave batch station* is always the source station. The maximum number of transaction instances initiated from the support environment that can be executing in a transaction processor at once depends on the number of slave batch stations in that transaction processor. Each transaction instance occupies one slave batch station during its execution.

11.2 INTERFACE BETWEEN TWO TRANSACTION PROCESSORS

The interface between two transaction processors is similar whether the transaction processors reside on the same TRAX system or on two remote systems. Two remote systems require a physical communication link; two transaction processors on the same TRAX system require a logical communication link. The only difference between the two situations is the planning and installation of the physical communication network.

11.2.1 Master and Slave Transaction Processors

The interface between two transaction processors requires one transaction processor to initiate transactions in a second transaction processor and to receive the results. The first transaction processor is called the *master* transaction processor; the second, the *slave* transaction processor.

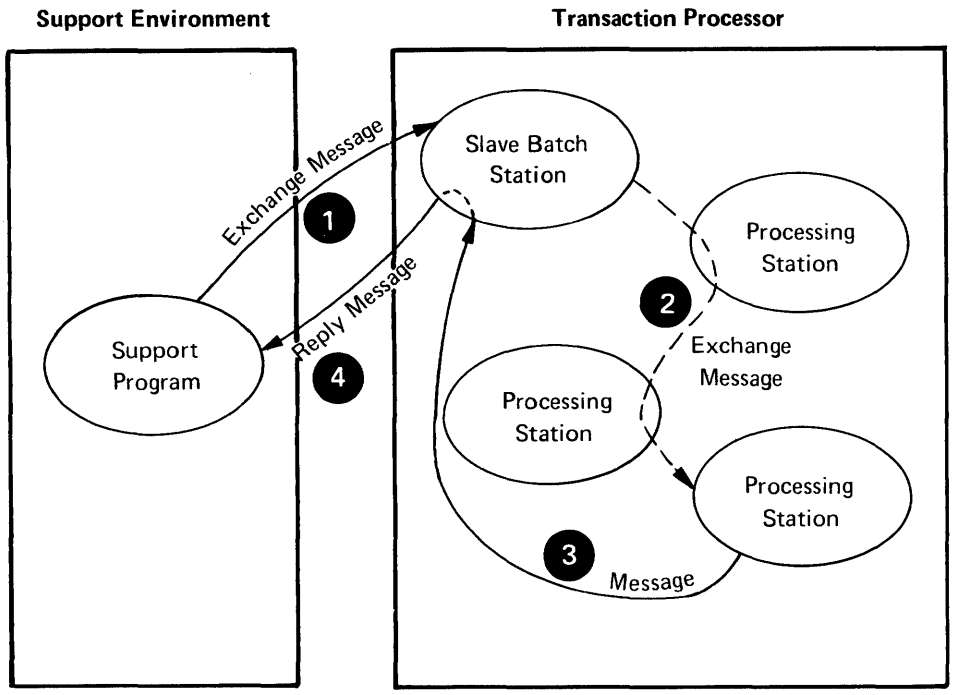
The terms "master" and "slave" are not fixed designations. Each of the pair of transaction processors can initiate transactions in the other. Therefore, both can be "master" or "slave." It is only in a request for transaction initiation that the terms "master" and "slave" have any significance.

11.2.2 How the Interface Works

Figure 11-4 shows how this interface works. In the master transaction processor, an exchange message arrives at a master link station. This station is associated with an external transaction processor, either:

- A local transaction processor having a logical communication link to the master transaction processor, or
- A remote transaction processor having a physical communication link to the master transaction processor.

The exchange message arrives at the master link station with control information and the text of an exchange message that is to be processed in the slave transaction processor. The master link station extracts the embedded exchange message and transmits it to the slave transaction processor.



Processing Steps:

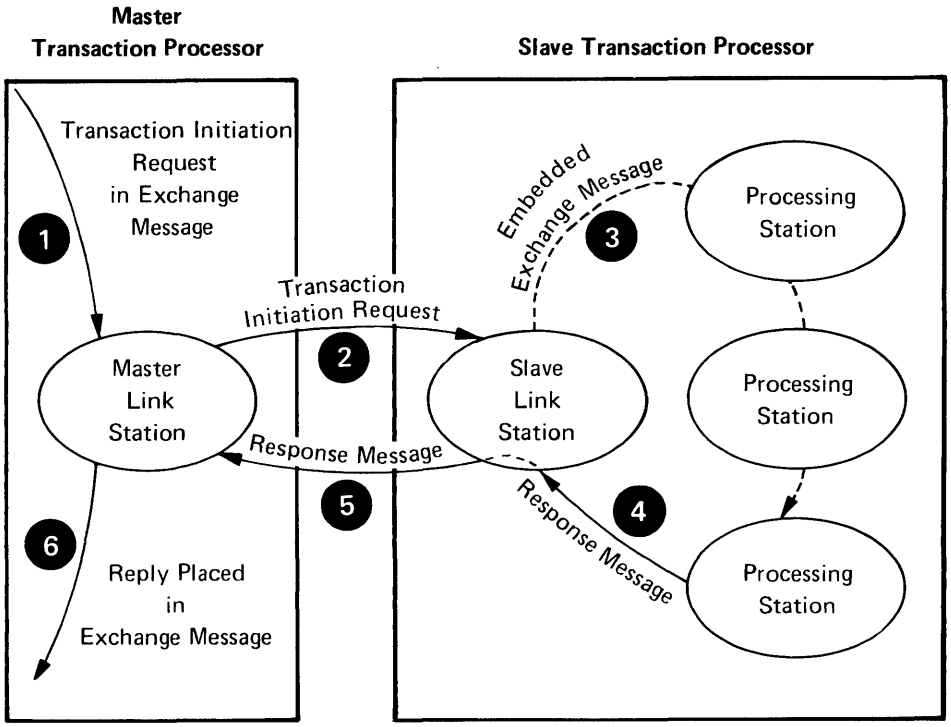
1. Support program issues a call which includes the text of an exchange message. The transaction processor selects a slave batch station which will act as the source station for the exchange message.
2. A transaction instance is initiated through the selected slave batch station, and the specified exchange message is routed to the list of stations named in the transaction definition.
3. One of the processing stations (usually the last) issues a response message addressed to the source station.
4. The response is forwarded to the support program.

Figure 11-3 Interface Initated by a Support Program

In the slave transaction processor, the extracted exchange message serves as the basis of an exchange. This could be the first (and possibly only) exchange of a newly initiated transaction instance, or it could be the second or subsequent exchange of a previously initiated transaction instance.

In the slave transaction processor, a *slave link station* is the source station for the exchange. That is, a slave link station is allocated (or retained, if this is a subsequent exchange) as the source station for the exchange message and the forwarding agent for the response message. The active number of transaction instances executing in a slave transaction processor initiated by master transaction processors depends on the number of slave link stations in that transaction processor.

When the exchange processing in the slave transaction processor is finished, a slave transaction processor TST sends a response message. This message returns to the slave link station in the slave transaction processor, which reformats and forwards the message to the master link station in the master transaction processor.



Processing Steps:

1. An exchange message arrives at a master link station in the master transaction processor containing a transaction request for another transaction processor.
2. The transaction initiation request is forwarded to the slave transaction processor, and a slave link station is assigned to initiate the transaction instance. (If the request is to continue a previous transaction instance, an initiating station will have already been assigned).
3. The exchange message embedded in the request is extracted and circulated to the appropriate stations in the slave transaction processor.
4. One of the stations which processes the exchange message (usually the last one) directs a response message to the initiating station.
5. The response message is forwarded to the master transaction processor.
6. The master link station in the master transaction processor places the response in the exchange message of the original transaction instance and sends the exchange message to its next destination.

Figure 11-4 Interface Between Two Transaction Processors

The master link station places the returned data in the original exchange message in the master transaction processor, and the transaction instance in the master transaction processor routes that exchange message to its next destination.

Meanwhile, the transaction instance in the slave transaction processor either is terminated or is suspended awaiting another exchange message from the master transaction processor. If it is terminated, its slave link station is released; if it is suspended awaiting further exchanges, its slave link station is retained. The suspended slave link station is accessible only to the transaction instance in the master transaction processor that allocated it, and the slave transaction instance is aborted if the master transaction instance aborts or terminates normally while the slave link station is allocated.

11.2.3 Cooperation Between Master and Slave

You have probably noted that this interface method can handle multi-exchange transactions in the slave transaction processor. The progress of the slave transaction is controlled by its transaction definition (as defined in the slave transaction processor) and the control information interpreted by the master link station. This control information serves much like application terminal function keys, interacting with the applicable transaction definition to determine the progress of the transaction instance.

The master and slave transaction instances cooperate so that the exchange messages are supplied at the proper times and are processed in the proper way. Mismatched messages and processing cause trouble; the design of both master and slave transactions is critical. Extreme situations are handled by the respective transaction processors; for example, the automatic abort of a slave transaction instance if the master transaction instance aborts. But the mundane and subtle aspects of transaction cooperation are handled with transaction definitions and TST processing logic.

The control parameters in the master exchange message are important to this cooperation. They are described in the *TRAX Application Programmer's Guide.*

11.2.4 Links and Sublinks

The data path between a master transaction processor and any of its slave transaction processors is called a *link.* A link includes a master link station within the master transaction processor.

The link is divided into one or more *sublinks.* While the links are permanent, sublinks are created and dissolved as transactions initiate and terminate in the remote transaction processor. Each sublink has a master link station at the master end, a slave link station at the slave end, and a logical data path between them.

Each remotely initiated transaction instance requires a slave link station and its corresponding sublink for the duration of the transaction instance. This includes the time between exchanges in the slave transaction processor, when that transaction processor awaits another exchange message from the master transaction processor.

Both links and sublinks are *logical* data paths; many links and their sublinks share a physical data communications facility. Each communication line between systems is point-to-point; multi-drop lines are not supported to join multiple TRAX systems. The interconnection of three TRAX systems requires three communication facilities, as shown in Figure 11-5.

Figure 11-6 shows two transaction processors on different TRAX systems. The figure shows a master link station in the left transaction processor, and two slave link stations in the right transaction processor. The left transaction processor, processor A, initiates up to two concurrent transaction instances in processor B with this arrangement. Transaction processor B can *not* initiate transaction instances in processor A; it can only respond to transaction requests made by processor A.

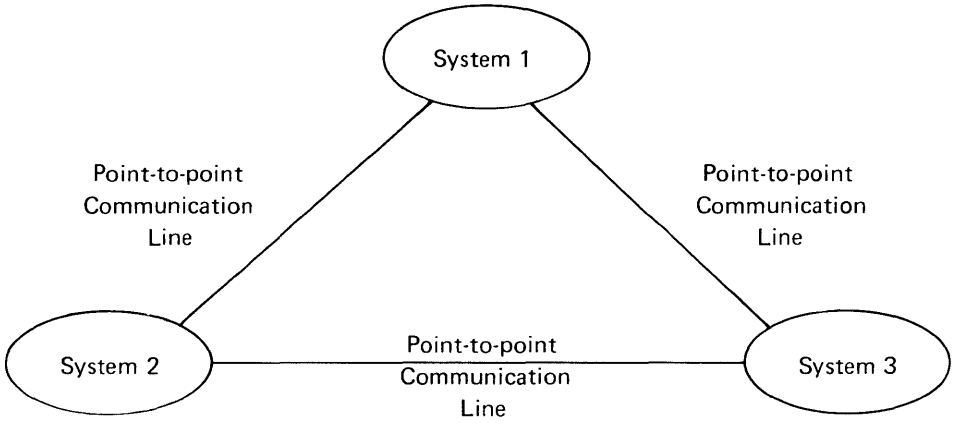


Figure 11-5 Interconnection of Multiple TRAX Systems

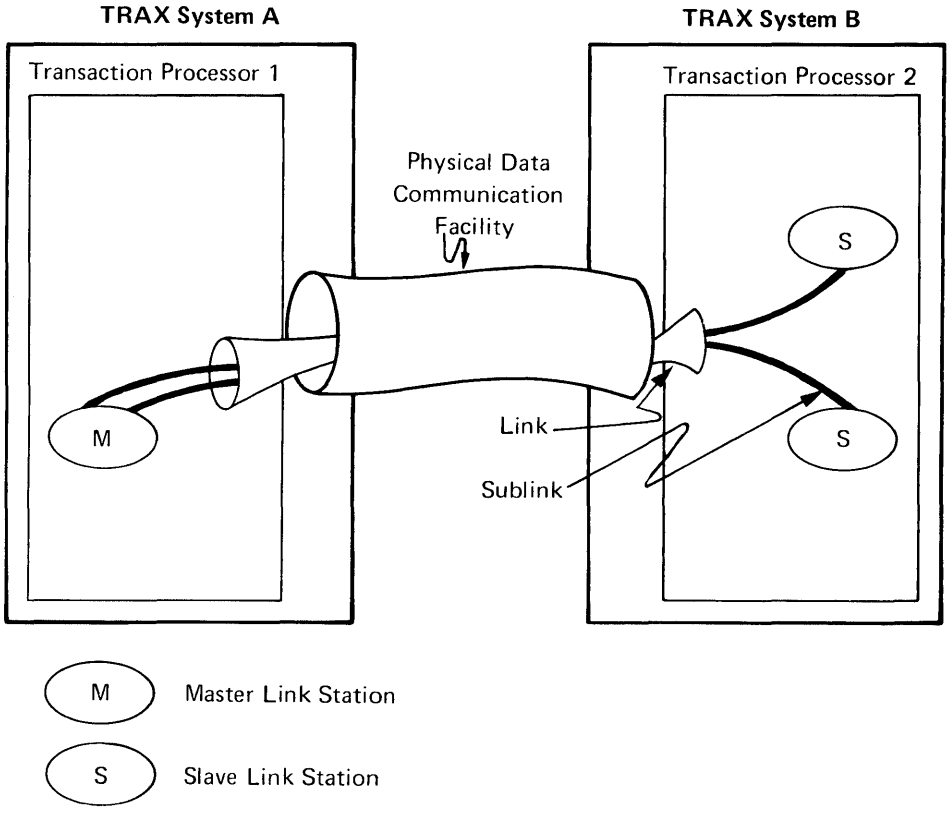


Figure 11-6 Links and Sublinks

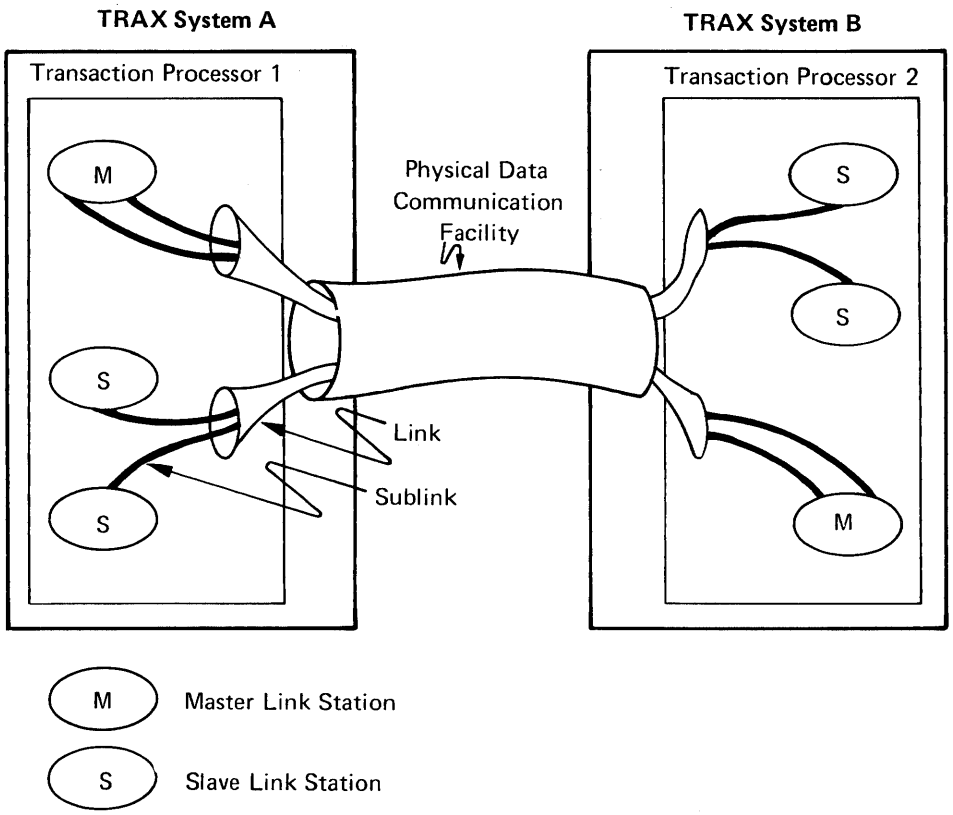


Figure 11-7 Duplexed Links and Sublinks

Figure 11-7 shows a duplexed arrangement where each transaction processor acts as a "master" transaction processor and initiates transaction instances in the other. Each transaction processor has a master link station, a link to the other transaction processor, and a pool of two slave link stations in the opposite transaction processor. Both links operate at the same time; that is, each transaction processor has active transaction instances processing in the other processor simultaneously.

The configuration of links, sublinks, and physical data communication facilities is determined in two steps:

- When the TRAX operating system is generated, the physical communication lines are configured. These communication lines are given identifiers for the second step.
- When the stations for a transaction processor are defined, the STADEF utility program will ask questions about each master and slave link station. The answers to these questions determine how the transaction processors will be connected, which physical communication lines (if any) are used by the transaction processor, and how many links exist on those communication lines.

See the *TRAX Support Environment User's Guide* for information about how to define master and slave link stations using the STADEF utility program.

11.3 INTERFACE WITH NON-TRAX SYSTEMS

TRAX transaction processors also interface with one non-TRAX system: a CICS application running on an IBM processor.

An application running under CICS operates much the same as a TRAX transaction processor, and this similarity makes interface possible. The CICS application receives input records from peripheral devices and processes them, generating one set of output records for each input record. This is similar to the way TRAX processes exchange messages and generates response messages.

TRAX treats a CICS application as a slave transaction processor, allowing TRAX master transaction processors to initiate transactions in the CICS application.

Several restrictions apply for this interface, when compared to the normal master-slave relationship between two TRAX transaction processors:

- A sublink, in TRAX terminology, corresponds to a device/control-unit address in IBM terminology. The appropriate parameters in each system are selected so that they correspond, and both systems know how many sublinks (DCU addresses) are active at one time.
- TRAX acts like IBM model 3270 terminal units for the CICS application. This applies only to the network architecture and device addressing, however. TRAX system software will not generate or process any 3270 control codes; this must be done by application TSTs.
- CICS does not support multi-exchange transactions, and therefore all transactions have only one exchange.
- CICS never acts as a master transaction processor and never originates transactions in the TRAX system.

Interfaces to CICS applications require special communication line configurations and station definitions. Configurations are described in the *TRAX System Generation Manual* and the *TRAX Support Environment User's Guide* (see the STADEF utility program).

PART TWO

Building a TRAX Application

CHAPTER 12
REVIEWING BUSINESS ANALYSIS TECHNIQUES

Before you design an application to run under TRAX, you must analyze the business environment that the application will serve. This chapter presents the analysis steps that you should perform. These are:

1. Identifying business activities
   a. Studying business procedures
   b. Studying data storage requirements
2. Developing system functional specifications
   a. Specifying system scope
   b. Choosing between fundamental system alternatives
   c. Specifying transaction processing functions
   d. Specifying batch processing functions
   e. Specifying data storage capabilities
   f. Specifying system reliability requirements

12.1 STUDYING BUSINESS ACTIVITIES

Your first task is to study the business activities that your automated system is to serve. This means that you must become familiar not only with computers and data processing, but also with the business operation itself.

Your study concentrates on two topics:

- The business procedures
- The business data kept, and how it is used

12.1.1 Studying Business Procedures

This portion of your study concentrates on activities and decision-making. It tells you *how*, *why*, and *when* certain business activities take place.

For example, an analysis of the business procedures in a wholesale distribution business (like that in the TRAX Sample Application) might answer these questions:

- How do orders arrive?
- How are picking lists generated for the warehouse staff?
- Are credit checks made and, if so, on what basis?
- How does the warehouse staff locate goods in the warehouse?
- Is there a separate packing staff, or do the pickers in the warehouse pack the goods too?
- Is the contents of each order double-checked by anyone?
- How are customers billed?
- How are backorders handled?
- Will partial orders be shipped if one or more items are unavailable or back ordered?
- How is stock procured from suppliers? What initiates orders for more stock?
- How is inventory managed?
- What procedures are used to accept incoming stock from suppliers?
- How are invoices for incoming stock approved and paid?
- Does the firm have catalogs? How are they mailed?

12.1.2 Studying Business Data Storage

To determine an application's data storage requirements, study the transient information the business uses each day and the permanent information it must keep on file.

Each business activity requires information. Your task is to determine the information needed by each business activity, and then find the common set of information needed for all business activities.

For example, an analysis of the data storage requirements of the TRAX Sample Application would attempt to answer these questions:

- Is the business using centralized files or distributed files? If distributed files, are they duplicates, or are certain data kept in different places?
- What different kinds of information does the business need?
- What data elements are in each kind of information – for instance, what data elements are included in a customer file?
- Which data elements appear in varying numbers or in varying sizes?
- What is the total quantity of data required?
- How quickly must the business retrieve each piece of information?
- Which data elements are used for computation? Which for decision making? And which are only displayed for general information?
- Which data elements are part of the business's official records?
- Which data elements have access restrictions or other security requirements?
- Is there a journal for accesses to certain data elements?
- How frequently is the data updated? Are there journals for these updates?
- How is data retrieved? For instance, are customer data retrieved by customer number, last name, or some other index?
- How long are data kept?
- Who decides when data can be discarded and on what basis?

12.2 DEVELOPING SYSTEM FUNCTIONAL SPECIFICATIONS

After you have studied the operation of the business, design a system that will address the business's need for automation. You do this primarily from the point of view of the business's requirements, although you must always keep in mind the limits of technical feasibility.

The result is a functional specification for the proposed system. A functional specification describes *what* the proposed system will do, but not *how* the system will be implemented. This does not mean that you can ignore technical feasibility; you should always have a feasible method in mind for accomplishing anything you propose in a functional specification. But, for the moment, the technical implementation of the system is not of interest. You must propose and document a system from a business point of view, so that you can prove its utility and receive authorization to proceed with its implementation.

The functional specification should cover these topics:

- System scope
- Fundamental system alternatives
- Transaction processing functions
- Batch processing functions
- Data storage requirements
- System reliability requirements

12.2.1 System Scope

First determine the scope of the proposed system. Often, you will not be automating the entire business. Many manual methods are adequate. Your analysis should identify the business areas that need automation now and in the immediate future. These areas should be confirmed by the business management.

Having chosen the system scope, try to stay with it. Build the originally-proposed system first; then add features later if it seems desirable. This will help you to meet development schedules.

12.2.2 Fundamental System Alternatives

You are planning a TRAX system for transaction-based, on-line processing. But, like most applications, your system will probably need some off-line or batch processing. One of the steps in a functional specification is choosing whether each business activity will be supported by on-line or off-line processing.

On-line processing is not always the best alternative. Many processes are schedule-oriented, that is, they are done at regular intervals with little human intervention. These are natural choices for off-line or batch processing.

Hybrid processing is also possible. For instance, you can use on-line processing to place data in a file, and off-line processing to print a report using that data.

12.2.3 Specifying Transaction Processing Functions

TRAX supports on-line functions with *transactions*. You must develop detailed functional specifications for each transaction. This involves the following design specifications:

- The purpose and function of the transaction – how it corresponds to manual business functions
- The information collected from its user
- The data retrieved from and stored in permanent data files
- The calculations done or decisions made
- The information presented to the user or to others
- The transaction volume expected and the speed with which the transaction must be executed
- Requirements for logging, journaling, and other security issues

12.2.4 Specifying Batch Processing Functions

The batch processing portions of the proposed system must also be specified.

NOTE

Because batch processing represents a more traditional form of data processing – one with which most readers will be familiar – the batch processing portion of a business application is not discussed in great detail in this manual.

As you specify the batch portion of your system, you will be considering these issues:

- The purpose and function of each batch processing sequence (or *stream*)
- The source and format of the data
- The schedule that controls the processing, or other initiating event that controls it
- The processing that must be done
- The destination and format of output
- The volume of data to be processed, and the time available for processing it.

12.2.5 Specifying Data Storage Requirements

Specify the data storage requirements of your application by outlining the set of data files needed to support the on-line and batch processing activities.

The file structures available under TRAX are:

- Sequential files
- Relative-record files
- Indexed-sequential files with multiple indexes

You can develop a functional file design by using these basic file structures. The most important design issues at this stage are the generic types of data to be stored and their methods of retrieval. Calculations of specific field and record sizes, recording methods, and other detailed technical decisions should be left until the detailed design phase.

Also consider such things as the performance impact that multiple on-line users will have on your file design, and the impact your file design will have on the throughput and response times of each transaction. Compare these performance estimates to the throughput and response time requirements you have determined from your business analysis. Resolve any conflicts.

12.2.6 Specifying System Reliability Requirements

System reliability requirements may or may not be important in your functional specification. Some applications are not sensitive to reliability issues; others are particularly sensitive. When specifying system reliability requirements, consider these questions:

- Must the system be "up" most of the time?
- What is the penalty if it is "down"?
- How much effort is it worth to improve reliability, thereby reducing "down" time?
- How essential and valuable is the data in the system's files?
- If files are destroyed by system accident, to what extent must they be reconstructed?
- How much time and effort can be invested in file reconstruction?
- How much system time (or user time) should be invested on an ongoing basis – that is, during system operation – to reduce reconstruction time and effort?

CHAPTER 13
AN INTRODUCTION TO TRAX TECHNICAL DESIGN

Three aspects of TRAX application design are significantly different compared to other implementation environments:

- The conversation between the system and its on-line users
- The sequence of processing applied to on-line user input
- Issues relating to system reliability and recovery of on-line processing results

For this reason, the remainder of this manual concentrates on the detailed technical design of the on-line portion of a TRAX application. These techniques will be new even to the seasoned application designer. If you need help with the off-line or batch portions of your application, review the *TRAX Support Environment User's Guide* (Order No. AA-D331A-TC) and perhaps a text on the design of batch processing systems.

The remainder of this manual will continue to use terms and concepts introduced in Part One of this manual. For example, the term "transaction processor" denotes the portion of the finished system that handles the on-line processing for the application. If you are not comfortable with these terms, stop from time to time to review the TRAX capabilities discussed in Part One.

13.1 DESIGN OF USER-SYSTEM CONVERSATION

As described in Part One of this manual, a TRAX application converses with its terminal users through forms. Forms are a transaction processor's only interface with its on-line users.

The extensive use of forms has an impact on both the system and its users:

- For the system, forms allow efficient processing. The user's completion of a form is similar to reading a unit record from a card reader or a record from a file. All data is received with a single operation.
- For the user, it is often easier to enter data when all data entry fields are available than respond to questions one at a time.

The concept of forms-oriented conversation affects TRAX application designs. Your application must collect a complete record of information from a user, rather than individual fields.

Two TRAX system concepts work together to determine the flow of user conversation:

- *Form definitions* specify the characteristics of each form. The form definitions are kept in a form definition file where they are available to the transaction processor.
- *Form sequence* depends on the definition of each transaction, as well as the actions taken by TSTs and the terminal operator's use of function keys.

When you design the user conversation for a transaction, you should begin with a mental image of the conversation process. Starting from this mental image, adjust the form definitions, the transaction definition, and the transaction's TSTs to achieve the desired sequence.

Many TRAX concepts interact closely during the execution of a transaction, and their interaction affects the flow of user conversation significantly. Here are examples of such interactions:

- The response messages sent by TSTs must agree with the transaction definition and its form definitions.
- Form definitions are a primary means of controlling the use of terminal function keys. Terminal function keys, especially the system function keys, significantly affect the flow of conversation.

13.2 PROCESSING DESIGN

With traditional implementation environments, one application program collects data through conversation with the user and then processes that data. But because TRAX uses forms to converse with the user, the application program (now called a TST) is free to concentrate on data processing rather than data acquisition.

The processing applied to a user's data in a TRAX application depends on three hierarchical factors:

1. At the highest level, processing depends on the *sequence of exchanges* executed within the transaction. This sequence is controlled by the transaction definition, which can be overridden by TST response messages and by the user via function keys.
2. At the middle level, processing for an exchange depends on the TST station *routing list* for the exchange message. The original routing list comes from the transaction definition, but the routing list can be modified by any TST processing the exchange message.
3. At the lowest level, processing at a TST station depends on each TST's programmed *sequence of operations*. In addition to controlling the flow of logic within the TST itself, a TST can alter the routing list for the exchange message being processed, sending it to a different series of stations; and the TST can also alter the sequence of exchanges for the transaction, directing execution of a particular exchange in the transaction definition regardless of the sequence specified in the definition.

An important phase in designing a TRAX application is partitioning its processing: partitioning the application into transactions; partitioning the transactions into exchanges; and partitioning the exchanges into the individual stations in the routing lists.

Separation of data processing from user conversation has many benefits. It helps structure the application design process. It simplifies application programs. And it improves system efficiency, because the user conversation is managed by resident system code. Since more time is generally spent in conversation than in processing, the time that application programs are in use is reduced.

This division between data processing and user conversation implies a formal interface between them. To accomplish this, TRAX uses exchange messages (a way for user input to be sent to processing programs) and response messages (a way for results and instructions to be returned to the user). As you design TRAX transactions, pay close attention to the content and timing of these messages.

Most of the stations that process exchange messages are TST stations. The processing done at these stations is specified when the corresponding TST is programmed. There are other stations, however, that may also process exchange messages. These stations (discussed in Part One of this manual) support such activities as communication with batch processing and communication between two transaction processors. The processing at these stations is done by software supplied with TRAX, and no additional code need be developed to use these stations.

13.3 APPLICATION RELIABILITY ISSUES

Designing a good on-line application for multiple users is a challenge. Multiple users must not interfere with each other, and the work processed by the application must be recorded so that it is not lost if the system fails.

TRAX has many features and facilities to assist you in implementing these necessary aspects of on-line applications. Staging, journaling, logging, exchange recovery, and crash recovery were presented in Part One of this manual. You should have a good understanding of these terms before proceeding with a TRAX application design.

However, TRAX can only offer capabilities. You must select from the available options and configure your application appropriately.

Experienced designers of commercial applications know that there are no "good" or "bad" solutions, only tradeoffs and compromises. Applications constructed under TRAX are no exception: one attractive feature can adversely impact another. Many times, the inclusion of all optional system features is the worst design strategy.

For instance, TRAX supports a sophisticated staging mechanism where file updates may be delayed until successful transaction completion. This feature is invaluable to you when transactions terminate unsuccessfully, and you wish them to have no effect on data files. The staging mechanism, however, extends the time that records are locked and therefore unavailable to other transaction instances. It also increases the size of the transaction's system workspace. By selecting the staging option to improve application reliability, you increase multiple-user interference – users lock each other out of data records and compete for available memory.

The final effect of these system reliability features depends on your design of the user conversation and processing portions of the application. For instance, if one transaction overlaps conversation with data processing, exchange recovery may not have the effect you desire.

As another example, consider the common practice of placing a control record at the front of a data file. This record might carry the next available customer number, as it does in the TRAX Sample Application. Adding a customer to the file requires that this control record be read, updated, and written before the new customer record can be inserted.

Such a record usually has high access frequencies, because adding customer records to the file requires access to the control record first. It is important, therefore, that each user read and update the record in the minimum time so that other users can access it.

But consider what happens if the staging facility is selected for this file. Once the control record is read and locked by a user, it cannot be unlocked until the entire transaction has been completed. If this process were to have other exchanges with their associated user interaction, considerable time might elapse before the record is unlocked. The result would certainly be unacceptable response time.

13.4 STEPS IN THE TRAX DESIGN PROCESS

Designing a TRAX application involves two principal activities:

1. Designing files to support the application's data storage requirements
2. Designing transactions and batch processing streams to support the application's processing requirements

File design for TRAX is like file design for other commercial, multi-user systems. This design process focuses on the structure of the files and the problems of shared access to those files.

Processing design for TRAX is significantly different from other systems, particularly for on-line or transaction-oriented processing. Design focuses on the transaction structure and the transaction components: transaction definitions, forms, station definitions, TST specifications, and so forth. These components, which are all interrelated, must be assembled to provide the desired transaction behavior.

File design and processing design usually proceed in parallel. As the content and structure of files are refined, transaction designs must be adjusted; as transactions are refined, file designs must be adjusted.

Processing design topics are covered in Chapters 14 through 20. File design topics are covered in Chapters 21 and 22. The topics are discussed in this order because the processing topics will be new to most readers, while the file topics will be familiar to many.

CHAPTER 14
DESIGNING THE OVERALL STRUCTURE OF A TRANSACTION

This chapter describes procedures for designing the overall structure of a transaction. It covers these major topics:

- Transaction structure diagrams
- Overlapped processing
- Transaction data structures
- Transaction access security techniques

The transactions identified during your preliminary analysis must be considered individually, and appropriate technical designs must be developed for each of them. Under TRAX, the implementation of each transaction will involve:

- An overall structure relating the user interactions, processing descriptions, and decision paths during transaction processing
- A detailed specification of each interaction with the user
- A detailed specification of each processing step that is required to process the data entered by the user

The first item, the overall structure of the transaction, is the best place to begin.

14.1 TRANSACTION STRUCTURE DIAGRAM

When you design TRAX transactions, you should develop diagrams that show the elements of a transaction and how they interact during transaction execution. These diagrams are called *transaction structure diagrams*. You should develop one for each transaction in your application.

When you develop these diagrams, try to concentrate on two goals:

- A working transaction design. That is, doublecheck your work against the material in Part One of this manual to make sure that the transaction can be implemented under TRAX and that it will operate as you intended.
- A sensibly organized and easy-to-use transaction. That is, imagine what it would be like to execute your transaction from an application terminal.

Your finished diagram should reflect both these aspects of transaction design – technical feasibility as well as ease of use.

14.1.1 An Example Transaction

The Change Customer transaction from the Sample Application provides a good example of a transaction structure diagram.

This transaction reads a record from the customer file and allows a terminal user to change the record and replace it. The processing steps in the transaction are:

1. When the transaction begins, the user is shown a form on which he enters the number of the customer whose record is to be changed.
2. The user either presses the CLOSE key, terminating the transaction, or enters a customer number as requested. If the customer number is entered to the form, the user presses the ENTER key.

3. Processing begins to locate and read the specified record.
4. If the record is not found, the user is so advised. The form is left on the screen, including the user's input, and the user either presses the CLOSE key or tries a new customer number. In other words, the user is returned to Step 1.
5. If the record is located, the user sees a new screen display showing the data from the record.
6. The user then either presses the CLOSE key, terminating the transaction, or he alters the displayed data and presses the ENTER key.
7. If the user alters the data and presses the ENTER key, processing begins to place the modified data in the file. This processing is done in two stages – data validation and then file update.

14.1.2 Diagram Symbols

A transaction structure diagram can show all of the important components of a transaction design. Figure 14-1 shows a transaction structure diagram for a typical transaction, the change customer transaction from the TRAX Sample Application.

These components of a transaction design can be shown on a transaction structure diagram:

- *Exchange Boundaries.* Each page of the diagram represents one exchange. Each page is labeled for the exchange it represents.
- *Exchange Activities.* There are two principal activities within each exchange: terminal conversation and exchange message processing. The former is shown on the left portion of the page; the latter, on the right.
- *Initial Display of Form.* The initial display of the form is shown by a "video display" symbol. The symbol contains a description of the form's purpose or effect.
- *Reply Definitions.* Each reply definition is represented by another "video display" symbol. Each reply symbol contains a description of the reply's purpose or effect, and is labeled with the appropriate reply number. Arrows designating REPLY messages pass through the appropriate reply symbol and then to the exchange's user action symbol.
- *User Action.* After the initial display of the form and each reply, the terminal user must take one of two actions:

  – Press a system function key. This causes an immediate transfer to a new exchange specified by a combination of the transaction definition and the key used.

  NOTE

  Certain of these keys can also cause the transaction instance to be terminated.

  – Enter data and then press the ENTER key or some other user function key. This creates an exchange message and sends it to the stations on the exchange routing list.

  The user action symbol (there can be only one in each exchange) shows the point where the exchange awaits the user action. The arrows leading from this symbol show the various function keys the user can press, and what happens when he presses each.

  System function keys and user function keys are treated differently in a transaction structure diagram because they have different effects:

  – *System function keys* repeat the exchange, enter a new exchange, or terminate the transaction instance. Each key is shown by a single arrow, usually directed to the margin of the page with an annotation of the key's effect. Each arrow is labeled with the name of its function key.
  – *User function keys* generate an exchange message and cause it to be routed to a series of processing stations. All enabled user function keys therefore share a common series of arrows on the transaction structure diagram. This series of arrows represents the path of the exchange message. It begins at

the user action symbol and proceeds through the processing stations on the exchange routing list. The segment of the arrow between the user action symbol and the first processing station should be labeled with the names of the enabled user function keys.

Because some function keys may be enabled or disabled by a reply definition, some function key arrows may require footnotes explaining the circumstances in which the user may employ them. For example, a function key arrow might carry a footnote explaining that the key is only enabled after the invocation of Reply 1.

- *Processing Stations*. These symbols, at the right side of the diagram, show the stations that receive the exchange message; usually they represent TST stations. Each symbol represents a separate station.
- *TST Actions*. TSTs may take several actions in processing an exchange message. These actions are shown in the transaction structure diagrams as arrows originating from within the TST, in addition to the exchange message arrows. When a TST takes an action, exchange message processing is not necessarily terminated; the TST can take an action and still process the exchange message.

  The TST actions that can appear on a transaction structure diagram are:

  - *Response message to activate reply*. If the TST modifies the present screen display and allows the terminal user to take another action, it must send a REPLY response message. This message is shown as an arrow coming from the TST to a reply symbol on the left side of the same exchange. The arrow is annotated to explain the purpose and content of the REPLY message and the reply number it will invoke.
  - *Response message to begin a new exchange*. A TST can send several kinds of response messages to cause the user's terminal to proceed to a new exchange. These messages are:

    PRCEED
    STPRPT
    TRNSFR

    These messages are also represented on the diagram as arrows coming from the issuing TST. The arrows proceed to the margin of the page and are labeled with the destination exchange's exchange label. Each arrow should be annotated with the message purpose, content, and message type.
  - *Response message to terminate the transaction instance*. A TST can send two kinds of messages that terminate the transaction instance:

    1. A CLOSE message terminates the transaction instance immediately. This message is represented by an arrow pointing to the margin of the page, with an annotation for the message and its effect. (This will usually be a return to the transaction menu.)
    2. An ABORT message is similar to a REPLY message and is drawn like that message.

  - *Report Messages*. If a TST must print data on an output-only terminal, it sends a report message with the data to that terminal's station. This action is shown by an arrow from the issuing TST to a standard flowchart "document" symbol at the right side of the page. The arrow is labeled with the report's purpose, content, and destination station name.
  - *Mailbox messages*. If a TST must deposit data in a mailbox station, it sends a mailbox message to that mailbox station. When a TST interrogates a mailbox to see how many messages it contains or retrieves a mailbox message from a mailbox, a TST issues the appropriate library call. The mailbox is represented by a standard flowchart "transmittal tape" symbol and the flow of mailbox messages by suitable arrows.

- *Exchange Parameters*. The effect of response messages and function keys depends to an extent on two parameters in each exchange of the transaction definition: the REPEAT parameter and the subsequent action parameter. These parameters are shown at the bottom of Figure 14-1 (Sheet 1 of 2), so that the correct paths for response messages and function keys can be determined and easily verified.

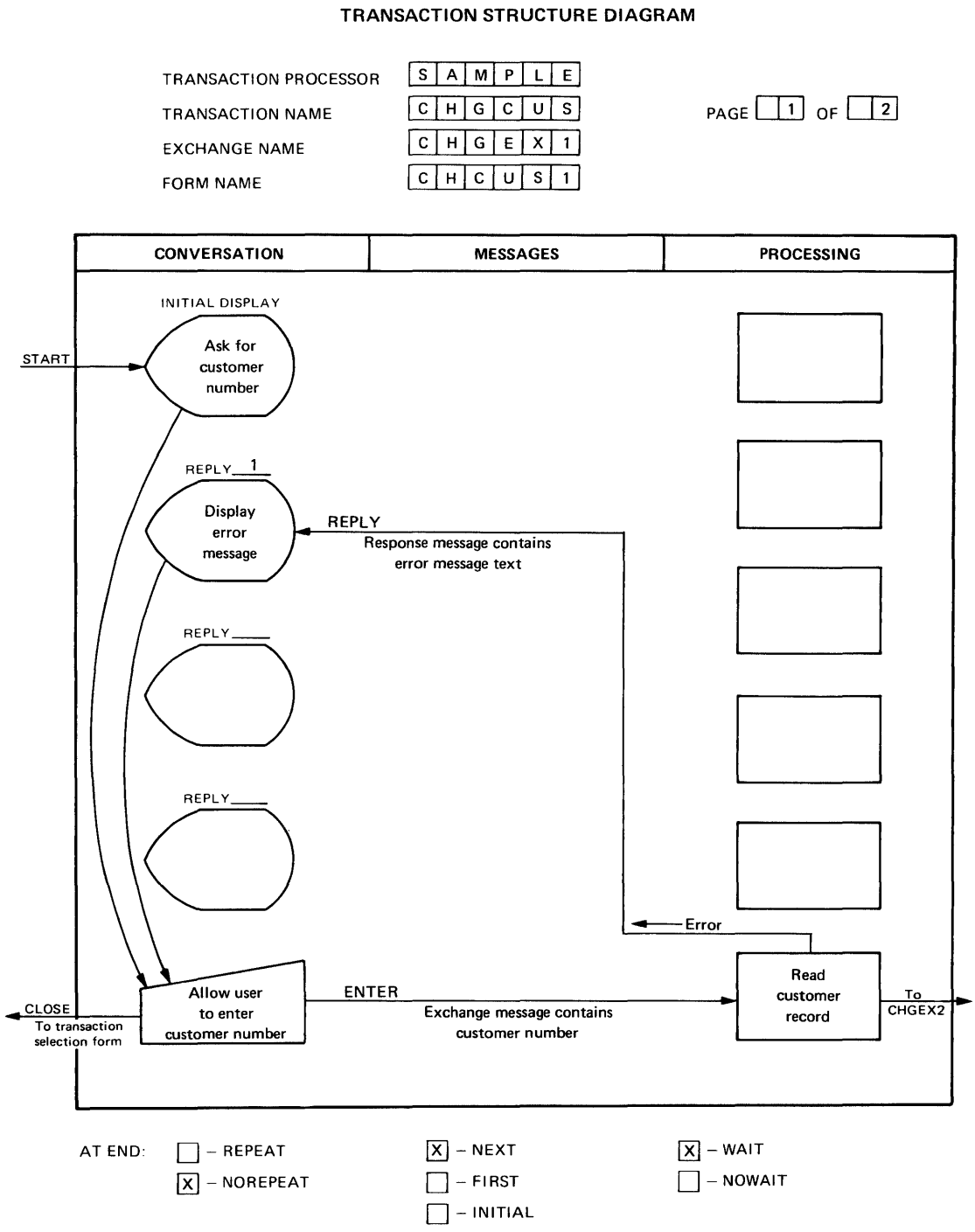


Figure 14-1 A Transaction Structure Diagram

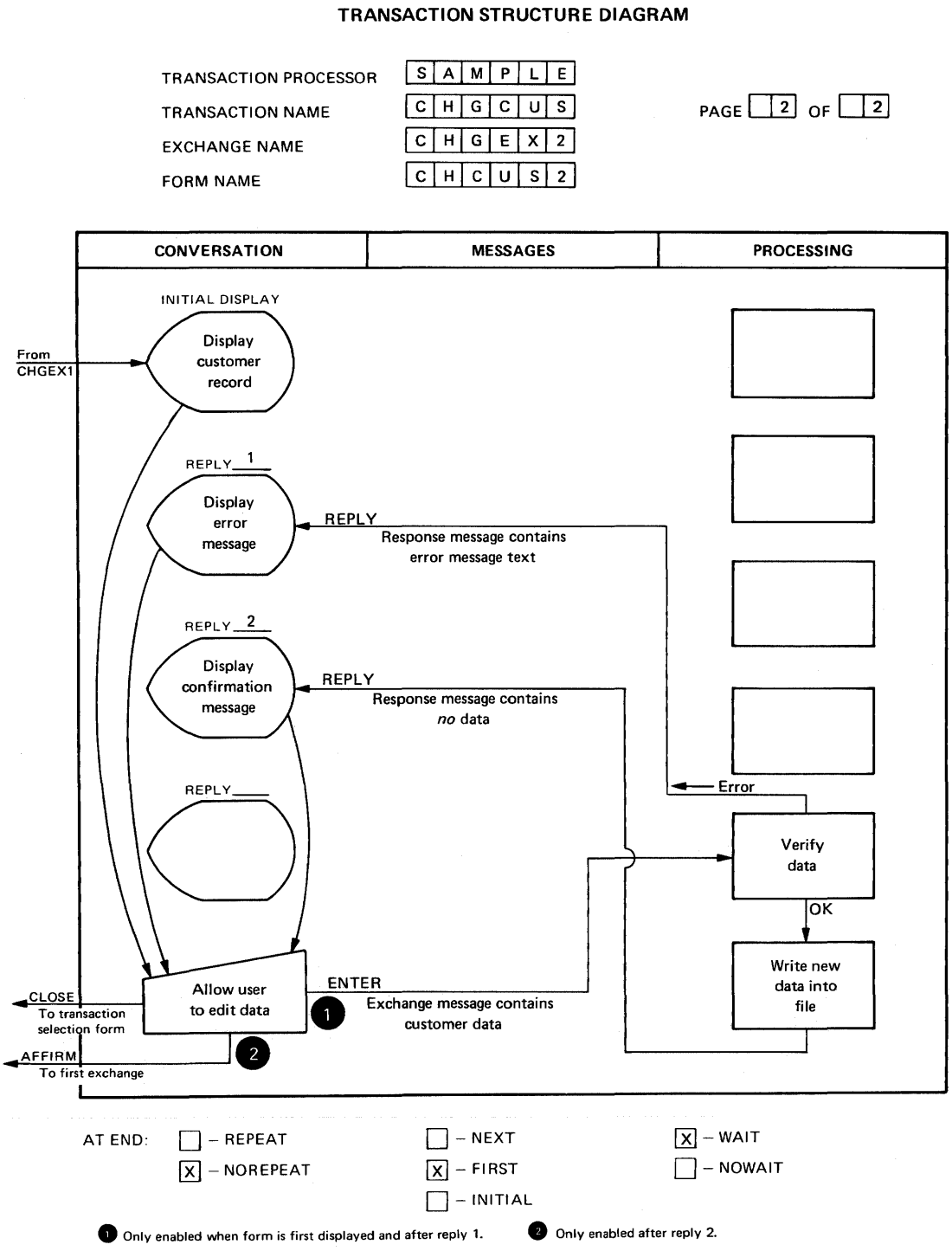


Figure 14-1 (Cont.) A Transaction Structure Diagram

Stop here to be sure that you can identify the following transaction components in Figure 14-1:

- Two exchanges
- Three TSTs
- The entry point from the transaction menu
- Exit points from the transaction
- The conversation and processing phases of each exchange
- The path of the exchange message in each of the two exchanges
- The initial display of each exchange's form
- The reply defined in the first exchange's form definition
- Two replies defined in the second exchange's form definition
- The point in each exchange where the user must enter data or press a function key
- The function keys enabled in each exchange
- A REPLY response message for errors in the first exchange's processing phase
- The PRCEED response message that causes the transition between the first and second exchanges
- A REPLY response message for errors in the second exchange's processing phase
- A REPLY response message that causes a transaction completion message to be displayed on the terminal screen

14.1.3 Transaction Control Flow

Three kinds of arrows on a transaction structure diagram are important: those representing exchange messages, response messages, and the effects of system function keys. These components of a transaction determine the transaction's flow of control – that is, the sequence in which steps of the transaction are executed.

As you construct transaction structure diagrams, be sure that these three kinds of arrows conform to TRAX transaction processor architecture. Make sure, too, that your arrows reflect your specifications for the REPEAT and subsequent action parameters. For example, if an AFFIRM key is shown going to the next exchange via the NEXT option, a PRCEED message must be shown as having the same destination.

Be sure you understand why each step of the transaction will occur the way you designed it. You cannot throw together a flowchart of a transaction without regard for the rules of transaction processor architecture and expect that transaction to work properly under TRAX. Your design must consider from the outset the capabilities of transaction processors.

Refer to Part One of this manual when you have specific questions about transaction processors and their capabilities.

14.2 OVERLAPPED PROCESSING

In typical transaction designs, response messages are sent by the last TST in each exchange's routing list shortly before the TST terminates. This causes the transaction instance to alternate user conversation and TST processing. First, the user enters data; then the system processes it; the user enters more data; and so forth.

It is possible, however, to overlap user conversation with the processing of exchange messages. In certain circumstances, this technique can improve a transaction's apparent response time. On the other hand, it is easy for an application designer to *unintentionally* overlap the conversation and processing phases of two exchanges by choosing the wrong timing for response messages or the wrong transaction definition parameters. A good understanding of the concepts in overlapped processing helps you to choose situations where it may be useful. It also helps you appreciate how transaction processors work, even if you never use overlapped processing in your transaction designs.

14.2.1 Overlap via Response Messages

One method of overlapped processing results from the distinction between *when a TST sends a response message* and *when the TST terminates execution*. Any TST in a routing list can send the response message; this is independent of the TST termination and other TSTs that must still process the exchange message.

By sending a response message, a TST releases the user's terminal for another conversational phase. The response message can:

- Invoke a reply and invite further data entry on the same form
- Repeat the exchange, using a new copy of the form
- Transfer to a new exchange and display the exchange's form
- Terminate the transaction instance, and follow with the display of the transaction's first form
- Terminate the transaction instance, and follow with the display of a transaction menu

If a response message is sent before exchange message processing is finished, the remaining processing phase of that exchange and the conversational phase of the new exchange proceed in an overlapped or parallel fashion.

You might use this technique in transactions where the user enters data that must be written into a file. Data entered this way usually has two processing steps:

1. The data entered by the user is checked for accuracy and consistency. If the user has made any errors, the processor stops and an error message is generated.
2. If the data is acceptable, it is written into the file. By this point, the only errors that can occur are system or hardware errors; while they may abort the transaction, the user cannot correct or avoid them.

The user is interested only in the results of the first processing step. If the data passes the consistency checks, the user can enter the next set of data. Meanwhile, the previous set of data can be written into the file.

You can get this effect under TRAX by sending the response message as soon as the data is successfully checked, leaving the same TST or another TST to continue processing in parallel with the conversatonal phase of the next exchange.

14.2.2 Overlap via the NOWAIT Option

If you wish to completely overlap exchange message processing with the conversational phase of the next exchange, TRAX allows a second overlap technique: the NOWAIT option in the transaction definition. If you specify this option for an exchange, the transaction processor will not wait for a response message from a TST before allowing the user's terminal to proceed to the conversational phase of the next exchange. The transfer will happen as soon as the exchange message is constructed.

You can only use this technique in transactions where a user enters data that can be written into a file without programmed edit checks. This mode of operation is frequently called "blind data entry." By using the NOWAIT option, you can design transactions with the best response times. But remember, the user receives no feedback for the data he enters.

NOTE

The NOWAIT option is not equivalent to having the first TST in the routing list send an immediate response message. Before a TST can send a response message, it must begin to process the corresponding exchange message. This means that if the exchange message waits in the TST's station queue, the response message will be delayed and optimum response times will not be secured.

14.2.3 Restrictions on Overlapped Processing

There are three restrictions on overlapped processing.

1. No further communication is possible between TSTs and application terminals once overlap begins.
2. Overlapped processing cannot be achieved in transactions for which exchange recovery is selected.
3. There is a restriction on the duration of overlap.

14.2.3.1 No Communication with Terminal during Overlapped Processing – Remember that a series of TSTs can issue only one response message to each exchange message. This means that when you use a response message to achieve overlap, no further response messages are possible as part of that exchange. When you use the NOWAIT option, you must not allow the exchange message processing to issue any response messages whatsoever.

This restriction makes sense in another important way. As soon as a TST issues a response message or a NOWAIT option is encountered, the display on the terminal screen is modified and the user's keyboard is unlocked. Further response messages – even if the transaction processor allowed them – would be pointless, because the old form that collected the exchange message data is no longer on the terminal screen. Response messages would disturb the terminal user as he worked on the new form.

14.2.3.2 No Overlap Possible if Exchange Recovery Selected – Overlapped processing cannot be achieved if you select exchange recovery for the transaction. Overlapped processing can cause the user and the transaction processor to be at two different points in the transaction definition. Exchange recovery is impossible when two such asynchronous events must be recorded and preserved. To avoid this problem, TRAX transaction processors delay messages issued by TSTs until the processing phase is completed. So even if you attempt overlapped processing in exchange-recovery transactions, no overlap occurs. Instead, you are using up storage space in the transaction processor to hold the delayed messages.

14.2.3.3 Restriction on the Duration of Overlap – A final restriction involves the extent of overlap between processing phases and conversational phases. Only one exchange message can exist for a transaction instance at a time. Therefore, if a user has begun overlapped conversation, the processing phase for the subsequent exchange cannot begin until the processing phase for the previous exchange ends. To have two processing phases active for a single transaction instance would require two exchange messages.

If a user completes an overlapped conversational phase before the previous exchange's processing phase has completed and he presses the ENTER key or other user function key to transmit the entered data to the transaction processor, the transaction processor will not accept the data until the prior processing phase terminates. The transaction processor remembers that the user tried to send data and automatically interrogates the terminal later to acquire the data. No additional user action is required. But the transaction instance cannot proceed until the prior processing phase ends.

14.3 TRANSACTION DATA STRUCTURES

Once you diagram the flow of each transaction, the next step is to lay out the data structures to be passed between elements of the transaction processor when the transaction is executed. You must determine the size and contents of the following data structures:

- *Exchange Messages.* Each exchange needs an exchange message format. Remember, the exchange message carries information entered by the user or derived from the form displayed on the user's terminal. If you allow the terminal user to choose between more than one user function key, you must reserve space in the exchange message for the identifier of the selected key. This is the way a TST determines which user function key was used.
- *Transaction Workspace.* You must specify a transaction workspace format that will satisfy all exchanges of the transaction. The workspace should accommodate all data passed between the TSTs that work on the transaction.
- *Response Messages.* You must specify the format of each of the response messages the TSTs can send. Some response messages need not contain any data other than the required parameters for the corresponding library call; in many cases, the call itself is sufficient. But if a response message contains data which will be displayed on the user's terminal, the format of this information must be specified.
- *Other Messages.* If any TSTs in the transaction send report or mailbox messages, these must also be specified.

Each of these data structures should be defined on specification sheets similar to those shown in Figures 14-2 through 14-6.

EXCHANGE MESSAGE SPECIFICATION SHEET

Transaction Processor ☐☐☐☐☐☐

Transaction Name ☐☐☐☐☐☐

Exchange Label ☐☐☐☐☐☐

Field No.	Starting Byte	Length (Bytes)	Contents
1			
2			
3			
4			
5			
6			
7			
8			
9			
10			
11			
12			
13			
14			
15			
16			
17			
18			
19			
20			
21			
22			
23			
24			
25			

Figure 14-2 Specification Sheet for Exchange Messages

TRANSACTION WORKSPACE SPECIFICATION SHEET

Transaction Processor ☐☐☐☐☐☐

Transaction Name ☐☐☐☐☐☐

Field No.	Starting Byte	Length (Bytes)	Contents
1			
2			
3			
4			
5			
6			
7			
8			
9			
10			
11			
12			
13			
14			
15			
16			
17			
18			
19			
20			
21			
22			
23			
24			
25			

Figure 14-3 Specification Sheet for Transaction Workspace

RESPONSE MESSAGE SPECIFICATION SHEET

Transaction Processor ☐☐☐☐☐☐

Transaction Name ☐☐☐☐☐☐

Exchange Label ☐☐☐☐☐☐

Type of Message

☐ – REPLY (Activates reply no. ☐☐☐)

☐ – PRCEED

☐ – STPRPT

☐ – CLSTRN

☐ – ABORT (Activates reply no. ☐☐☐)

☐ – TRNSFR (To exchange ☐☐☐☐☐☐)

Field No.	Starting Byte	Length (Bytes)	Contents
1			
2			
3			
4			
5			
6			
7			
8			
9			
10			
11			
12			
13			
14			
15			
16			
17			
18			
19			
20			
21			
22			
23			
24			
25			

Figure 14-4 Specification Sheet for Response Message

REPORT MESSAGE SPECIFICATION SHEET

Transaction Processor ☐☐☐☐☐☐

Transaction Name ☐☐☐☐☐☐

Report Form Name ☐☐☐☐☐☐

Field No.	Starting Byte	Length (Bytes)	Contents
1			
2			
3			
4			
5			
6			
7			
8			
9			
10			
11			
12			
13			
14			
15			
16			
17			
18			
19			
20			
21			
22			
23			
24			
25			

Figure 14-5 Specification Sheet for Report Message

MAILBOX MESSAGE SPECIFICATION SHEET

Transaction Processor ☐☐☐☐☐☐

Mailbox Station Name ☐☐☐☐☐☐

Sending Transaction Name ☐☐☐☐☐☐ TST ☐☐☐☐☐☐

Receiving Transaction Name ☐☐☐☐☐☐ TST ☐☐☐☐☐☐

Field No.	Starting Byte	Length (Bytes)	Contents
1			
2			
3			
4			
5			
6			
7			
8			
9			
10			
11			
12			
13			
14			
15			
16			
17			
18			
19			
20			
21			
22			
23			
24			
25			

Figure 14-6 Specification Sheet for Mailbox Message

The system workspace should also be considered at this time. You need not specify its exact size or format, but you should consider what data structures the transaction processor will have to store in the system workspace. If your transaction design places unreasonable demands upon the system workspace, such as staging a large number of lengthy records, system performance may suffer and you should redesign the transaction. The size required for the system workspace is computed in the transaction design documentation process described in Chapter 16.

14.4 TRANSACTION ACCESS SECURITY TECHNIQUES

When you design each transaction, you must consider which users or classes of users are to be allowed access to it. In many commercial applications, this consideration is as important as the operation of the transaction itself.

Two techniques are available to control access to TRAX transactions:

1. *Terminal-Based Access.* Anyone using a particular terminal (or a particular dial-up port) is allowed access to a common set of transactions. With this technique, a user does not identify himself to the system.
2. *User-Based Access.* The set of transactions available from any particular terminal or dial-up port depends on the user requesting access. With this technique, each user identifies himself to the system when he begins work at a terminal and informs the system when he leaves the terminal.

Both of these access control techniques depend on work classes, which are defined sets of transactions. Work classes are described in Chapter 9.

14.4.1 Terminal-Based Access

Terminal-based access can be implemented either with or without a transaction selection form:

- If more than one transaction is to be available from the terminal, a transaction selection form is needed to allow the user to make his choice. The terminal should also be assigned a work class that specifies which transactions the terminal can execute.
- If only one transaction is to be available, the transaction selection form and work class are unnecessary; the terminal always displays the first form of its only transaction when it is idle.

14.4.2 User-Based Access

User-based access is implemented with the following special transaction processor components, supplied with each TRAX system generation kit:

- A SIGNON transaction that identifies the user to the system
- A SIGNOF transaction that informs the system that the user is leaving the terminal
- Utility programs that maintain the AUTOEF file, which contains the names and attributes of the system users.

Those terminals that require user identification must be assigned to the SIGNON work class. This work class must contain the SIGNON transaction. It can also contain other transactions you wish to be accessed from such terminals without further user identification.

At least one of each user's work classes *must* include the SIGNOF transaction. When a user executes this transaction, the terminal reverts to its original SIGNON work class. Another user can subsequently identify himself with the SIGNON transaction.

14.4.3 Access Control Design

Use a two-step process to select access control techniques.

1. Divide the application's permanent terminals, dial-up ports, and users into groups by access requirements.
2. Select groups of transactions appropriate for each.

You may find that terminal-based access techniques are adequate, or that user-based access techniques are adequate. Usually, a combination is best.

CHAPTER 15
SEVERAL TRANSACTION DESIGN EXAMPLES

This chapter provides several examples of the transaction structure diagram as you might use it to design a transaction. The chapter begins with a simple transaction design and progresses to sophisticated design.

15.1 THE APPLICATION PROBLEM

The transaction described in this chapter is the Display Customer transaction from the TRAX Sample Application. This transaction is one of several that are used to maintain a customer master file for the application. The purpose of the transaction is to allow users to inspect the contents of the customer master file.

The customer master file is an indexed file whose primary index is the customer number. The file also has a secondary index by customer name.

Besides the fields of primary and secondary indexes, the file has the following fields:

- Address
  Street
  City
  State Code
  Zip Code
- Full Telephone Number
- Contact Name
- Credit Limit
- Current Balance
- Purchases to Date
- Next Order Number
- Next Payment Number

15.2 A SIMPLE TRANSACTION DESIGN

Figure 15-1 illustrates, perhaps, the simplest transaction design that solves the application problem.

The transaction requires two exchanges, because the transaction requires the entry of a customer identification number before that record can be shown on the terminal screen.

1. The form for the first exchange collects a customer number from the user, and the customer number is sent as an exchange message to a TST. This TST reads the specified record and returns the data from the record as a response message.

   The response message is a PRCEED message and the subsequent action for the exchange is NEXT, so the transaction moves to the second exchange.

2. This exchange displays the customer data from the response message and solicits a function key from the user. This gives the user time to read the displayed data.

   There is no processing for the second exchange. After reading the data, the user presses the AFFIRM function key if he wishes to see a new customer record.

During the conversation phase of both exchanges, the CLOSE function key is enabled. Pressing this key terminates the transaction and returns the user to the appropriate transaction selection screen.

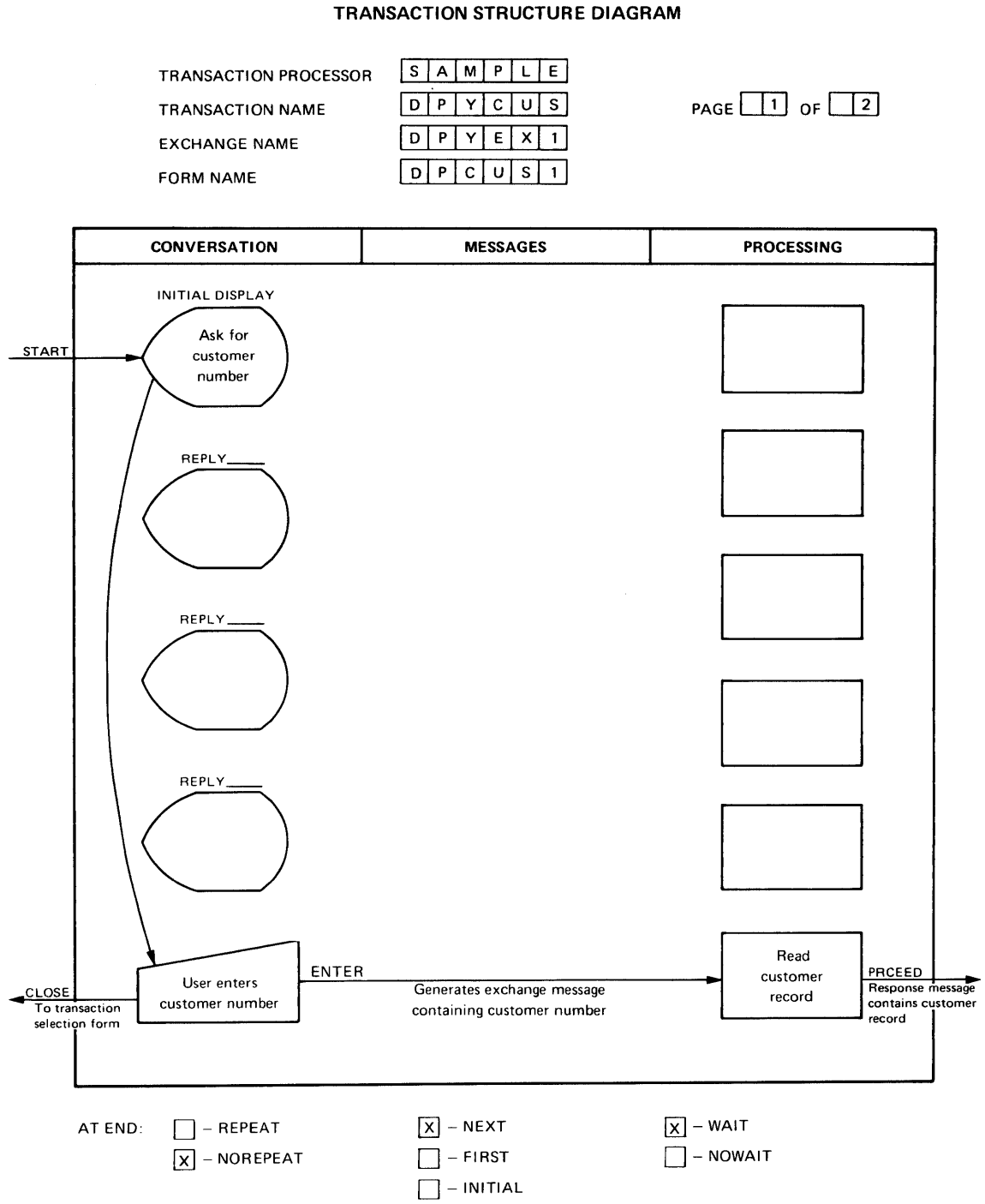


Figure 15-1 A Simple Transaction Design

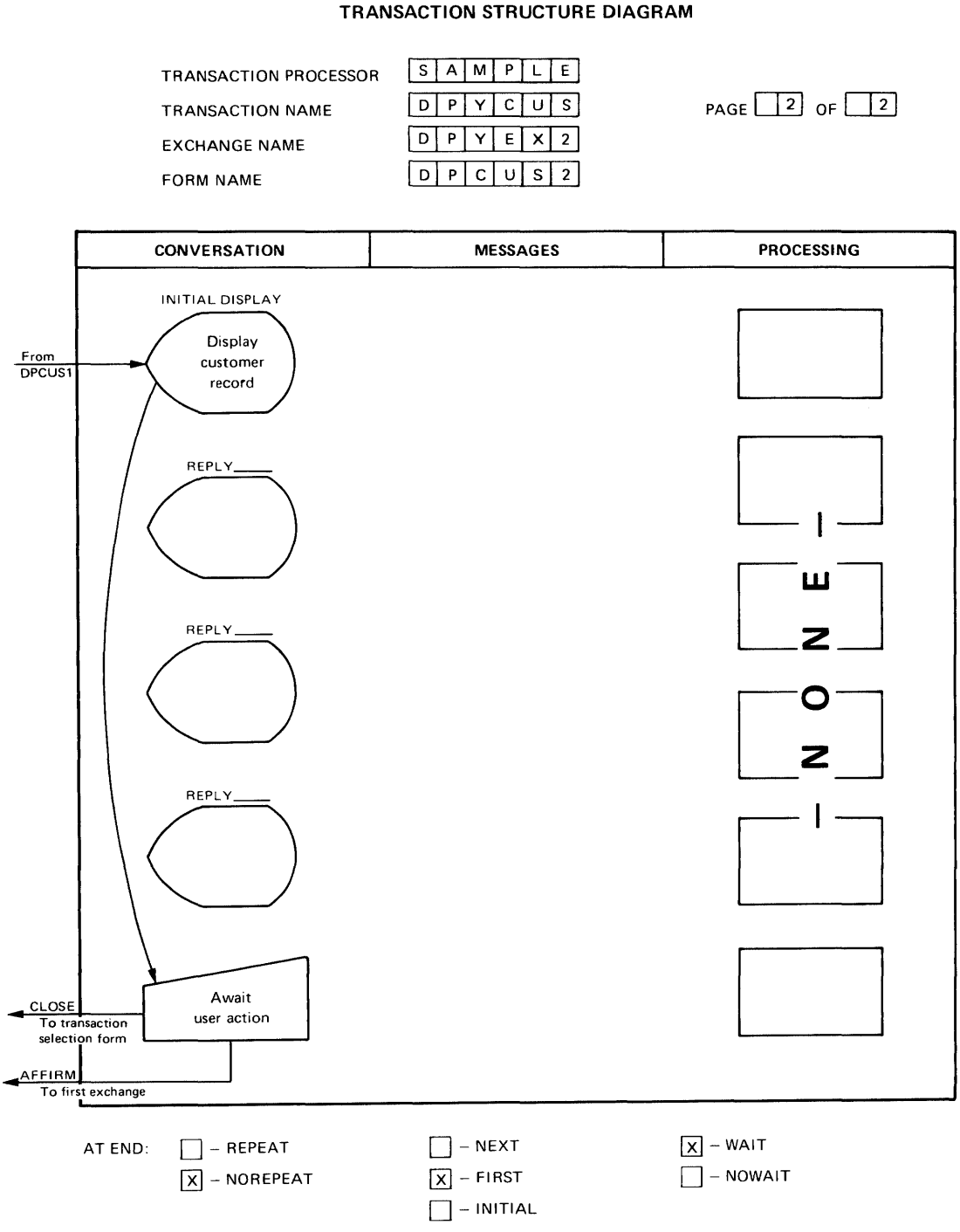


Figure 15-1 (Cont.) A Simple Transaction Design

Conversation and processing are not overlapped in this design: both exchanges use the WAIT option rather than NOWAIT, and the TST in the first exchange's processing waits until its processing is done before sending the response message.

Neither exchange uses the REPEAT option; each is executed once during the transaction.

The subsequent action for the first exchange is NEXT, indicating that when a PRCEED response message is received, the transaction should move to the conversation phase of the next change.

The subsequent action for the second exchange is FIRST. This indicates that when the user presses the AFFIRM key in response to the second exchange, the transaction instance will be terminated. After the current transaction instance is terminated, the transaction processor will display the form from the first exchange and prepare for another instance of the same transaction. If we were to select instead a subsequent action of INITIAL, the AFFIRM key would still terminate the transaction – but the transaction processor would display the appropriate selection form instead of the first form for the transaction. If the user is expected to execute several of these transactions in a row, the FIRST option is more appropriate; if the transaction is normally executed only once, then the INITIAL option is more appropriate.

An implementation of this transaction would require two forms and one TST. The second exchange does not need a TST, because no processing is required. The exchange exists only to allow a function key to be pressed when the displayed data has been read.

15.3 AN ALTERNATIVE DESIGN WITH ONLY ONE EXCHANGE

The transaction design shown in Figure 15-1 has two exchanges. The user interaction which collects the customer number is in one exchange, and the user interaction which displays the data and confirms that the data has been read is in a second exchange.

Figure 15-2 shows the result of this transaction design: alternating conversation and processing, where the conversation phase and processing phase of each exchange are entered exactly once.

Figure 15-3 shows an alternative transaction design using a single exchange. However, two user interactions are still possible, because the conversational phase of the exchange is entered twice during each execution of the transaction.

1. During the first entry, the user enters a customer number and presses the ENTER key.
2. During the second entry, the user presses the AFFIRM key to acknowledge that the data has been read.

Between these two entrances to the conversational phase, there is a processing phase where a TST reads the selected record. The data from this record is returned in a REPLY response message, rather than in the PRCEED response message used in Figure 15-1. *The REPLY message causes the transaction to re-enter the conversational phase of the exchange, rather than proceed to the next exchange of the transaction.*

The REPLY message activates a defined reply in the form definition and supplies the customer data used by that reply definition. The reply definition designates the positions on the screen where the data will be displayed.

Figure 15-4 shows the result of this transaction design. Where the simple design in Figure 15-1 resulted in alternating conversational and processing phases for each exchange, the design for a single exchange in Figure 15-3 results in a looping flow where each conversational phase is visited twice and each processing phase, once.

The transaction designs shown in Figures 15-1 and 15-3 appear almost identical to a user at an application terminal. The only apparent difference between the two designs is that in the simple design (Figure 15-1), the entrance to the second exchange causes the terminal screen to be erased and a new form to be constructed. *This is always the case*

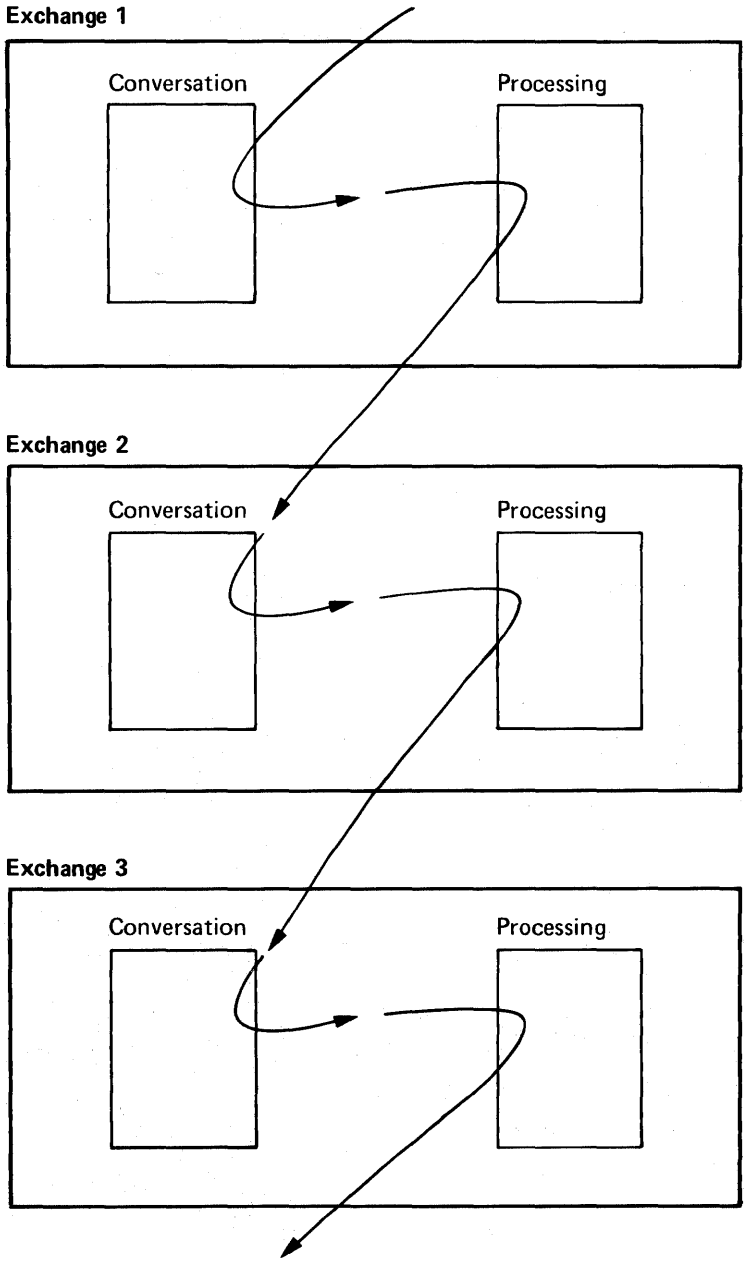


Figure 15-2 A Transaction Where Each Exchange is Entered Only Once

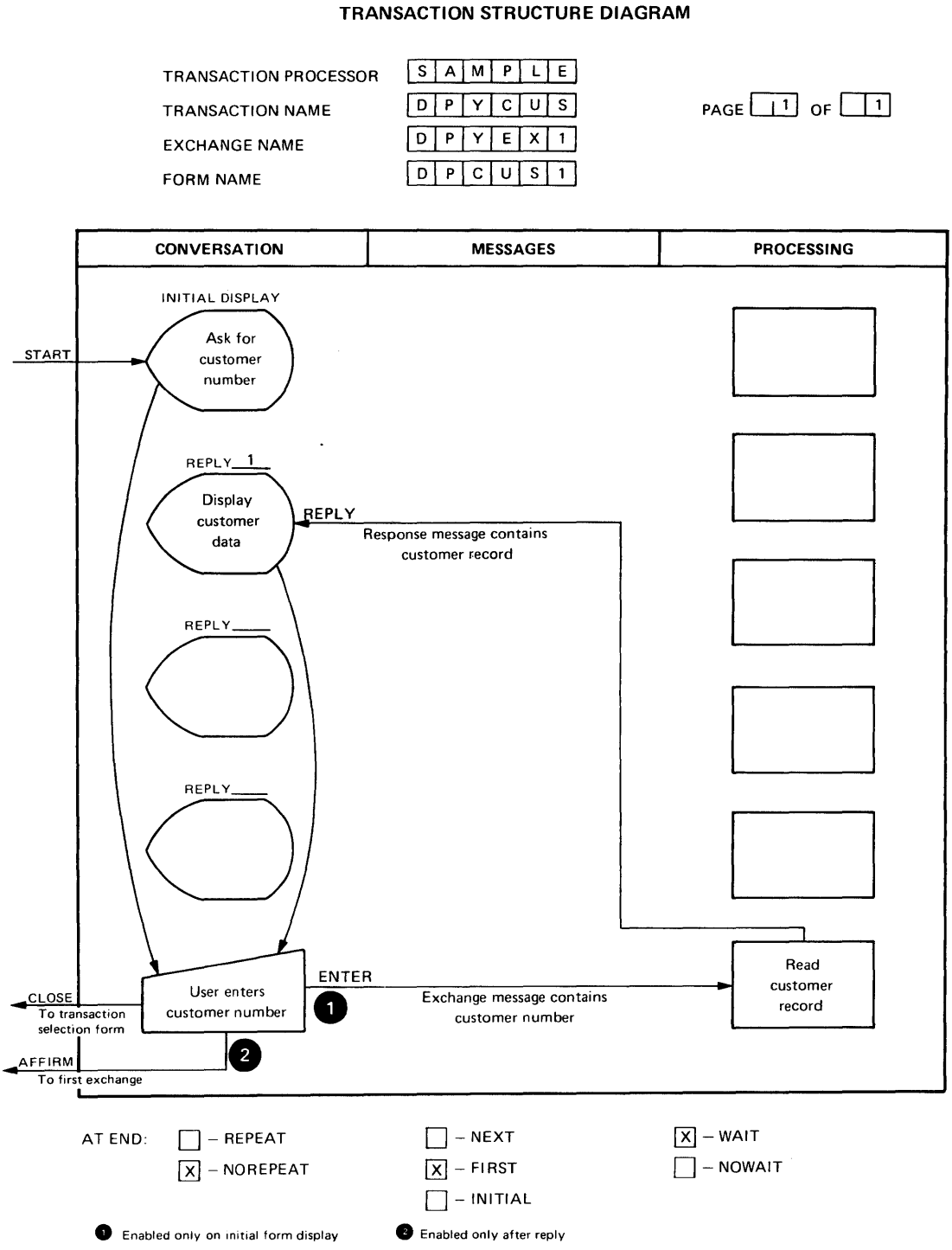


Figure 15-3 A Transaction Having Only One Exchange

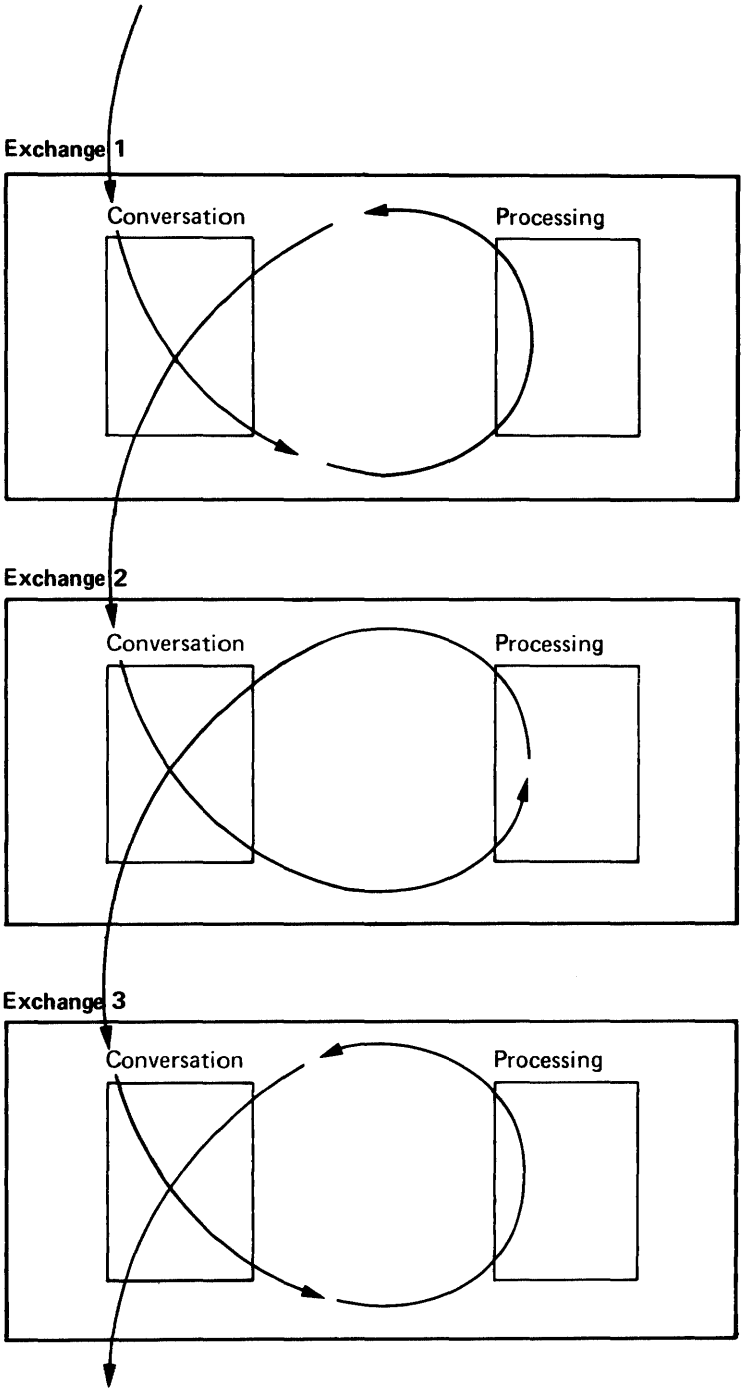


Figure 15-4 A Transaction Where Each Exchange is Entered Twice

when a new exchange is entered. In contrast, the simple exchange design in Figure 15-3 does not erase the screen; activating a reply affects only those fields named in the reply definition. (In fact, activating a reply is the *only* way the conversational phase of an exchange can be re-entered without erasing and redisplaying the exchange form.)

Although the single-exchange design (Figure 15-3) achieves economy by reducing the number of forms required, this savings may have undesirable side effects. For example, the single-exchange design gives you considerably less flexibility in the ways you can change the screen display between the first and second steps of the transaction. Remember that the characteristics of each field are determined when a form is first displayed; from that time on, the content of the field may be changed by replies but the field's characteristics cannot be changed. Because the design in Figure 15-3 uses only one form (using replies as a substitute for a complete second form) the characteristics of each field must remain the same during the transaction instance. For instance, if a field is displayed in reverse video in the first step of the exchange, it must remain in reverse video for the second step.

An implementation of the single exchange transaction requires one form and one TST. The form, however, must have a reply definition so that a REPLY response message containing customer data can display that data in the appropriate fields of the form. The REPLY message must also disable the ENTER key, which was enabled on the first entrance to the conversational phase, and enable the AFFIRM key for use during the second entrance to the conversational phase. It is the enabling and disabling of function keys that allows the exchange to operate differently for these two entrances.

15.4 THE EFFECT OF THE REPEAT OPTION

The single-exchange transaction design shown in Figure 15-3 activates a transaction instance each time a customer record is displayed. This happens because the exchange definition specifies NOREPEAT and the subsequent action parameter specifies FIRST. Therefore, the user's terminal is set for the execution of a new transaction instance after each display operation is finished.

Consider what happens when this transaction design is modified by substituting REPEAT instead of NOREPEAT in the definition of the exchange. This change is shown in Figure 15-5.

To the user, this transaction design is identical to that shown in Figure 15-3. The only difference is in the system's handling of the transaction.

With the specification of REPEAT, the AFFIRM key no longer terminates the transaction instance and prepares for another. Instead, the AFFIRM key continues the same transaction instance with another execution of the exchange just completed. The exchange starts from the beginning, with a fresh copy of the exchange form.

It must be emphasized that *the user sees the same screen display* in Figures 15-3 and 15-5, after pressing the AFFIRM key. The difference is that in Figure 15-3 a transaction instance terminates and another is ready to begin; in Figure 15-5, the transaction instance continues. This distinction is invisible to the user, and affects only the application design, its implementation, and its execution within the system.

Some effects of the continuing transaction instance are:

- Staged records continue to be staged and are not written to the files.
- Staged records continue to be locked.
- No journaling occurs at this time.
- The same transaction workspace continues to be used.

None of these effects is important in the display customer transaction, because that transaction does not use record locking or updating and uses no transaction workspace. In other transactions, however, these effects are important.

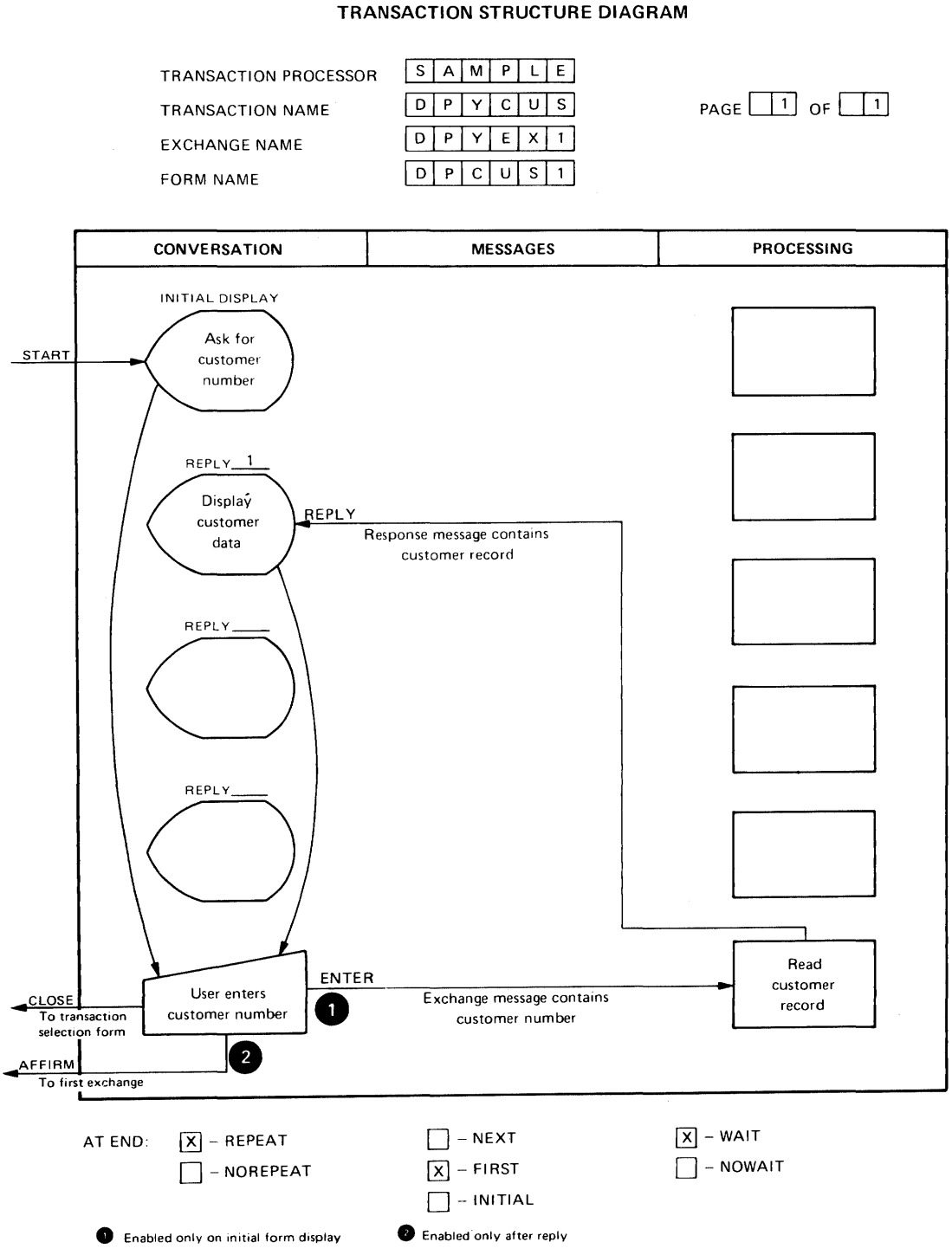


Figure 15-5 Using the REPEAT Option

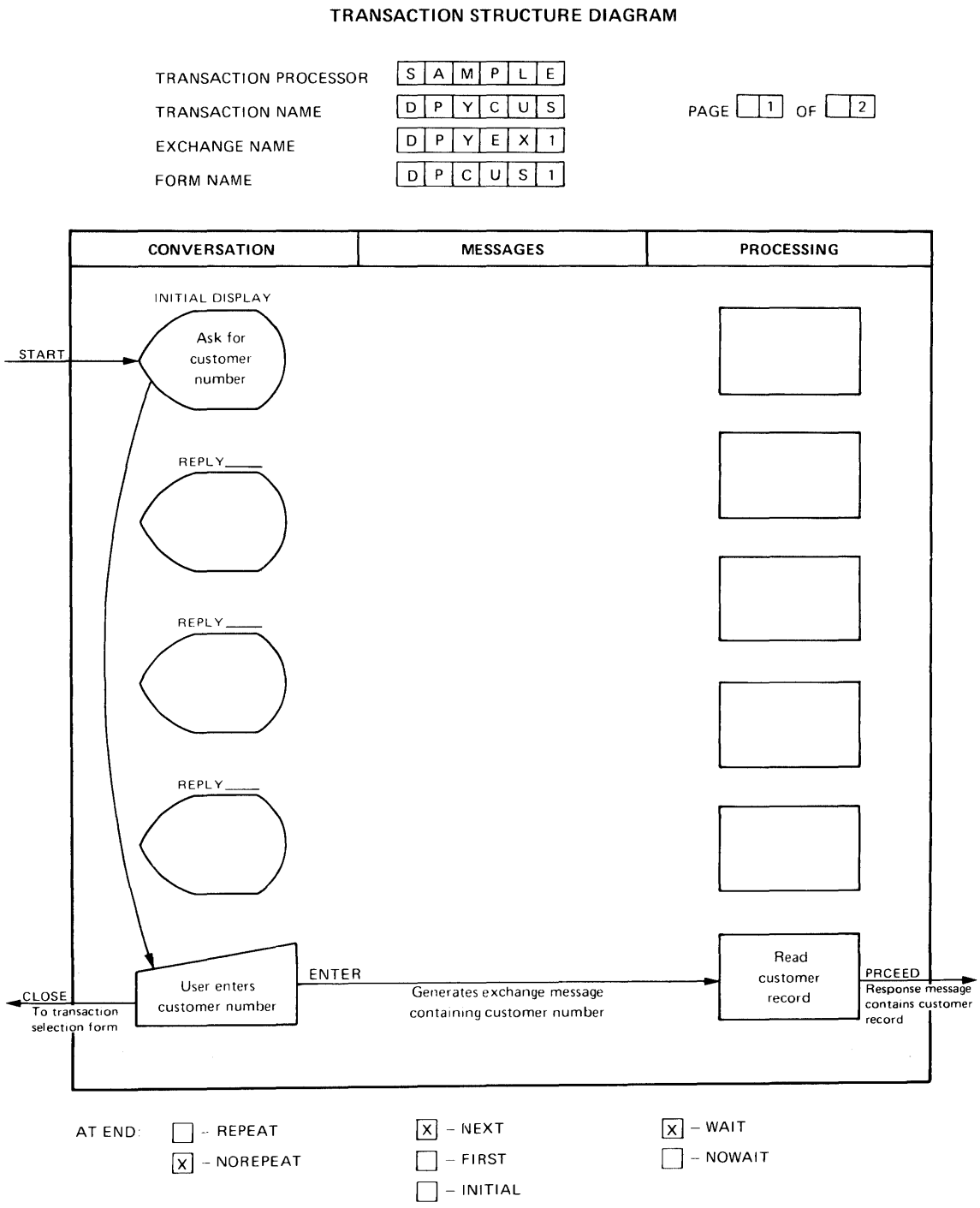


Figure 15-6 Allowing the User to Browse

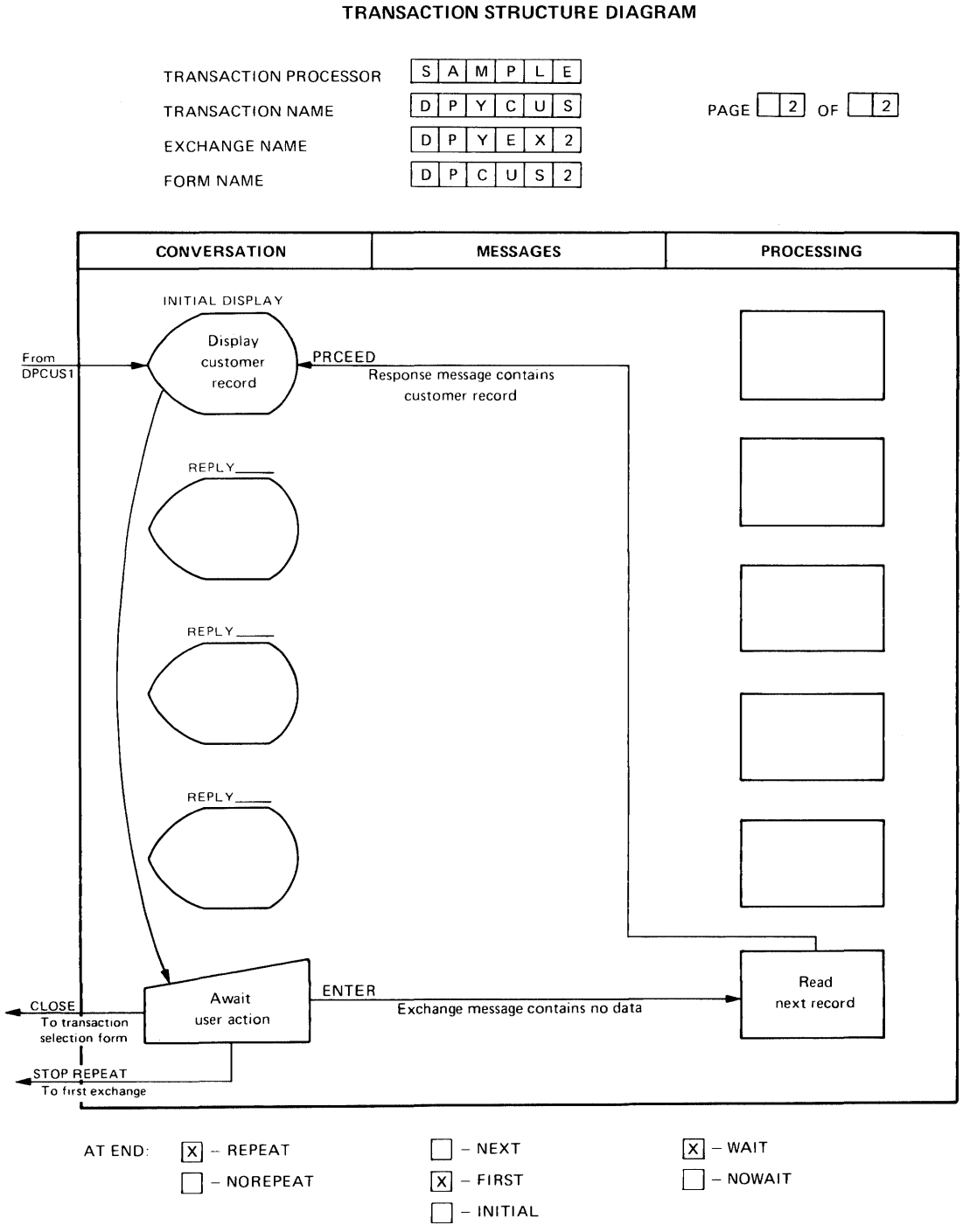


Figure 15-6 (Cont.) Allowing the User to Browse

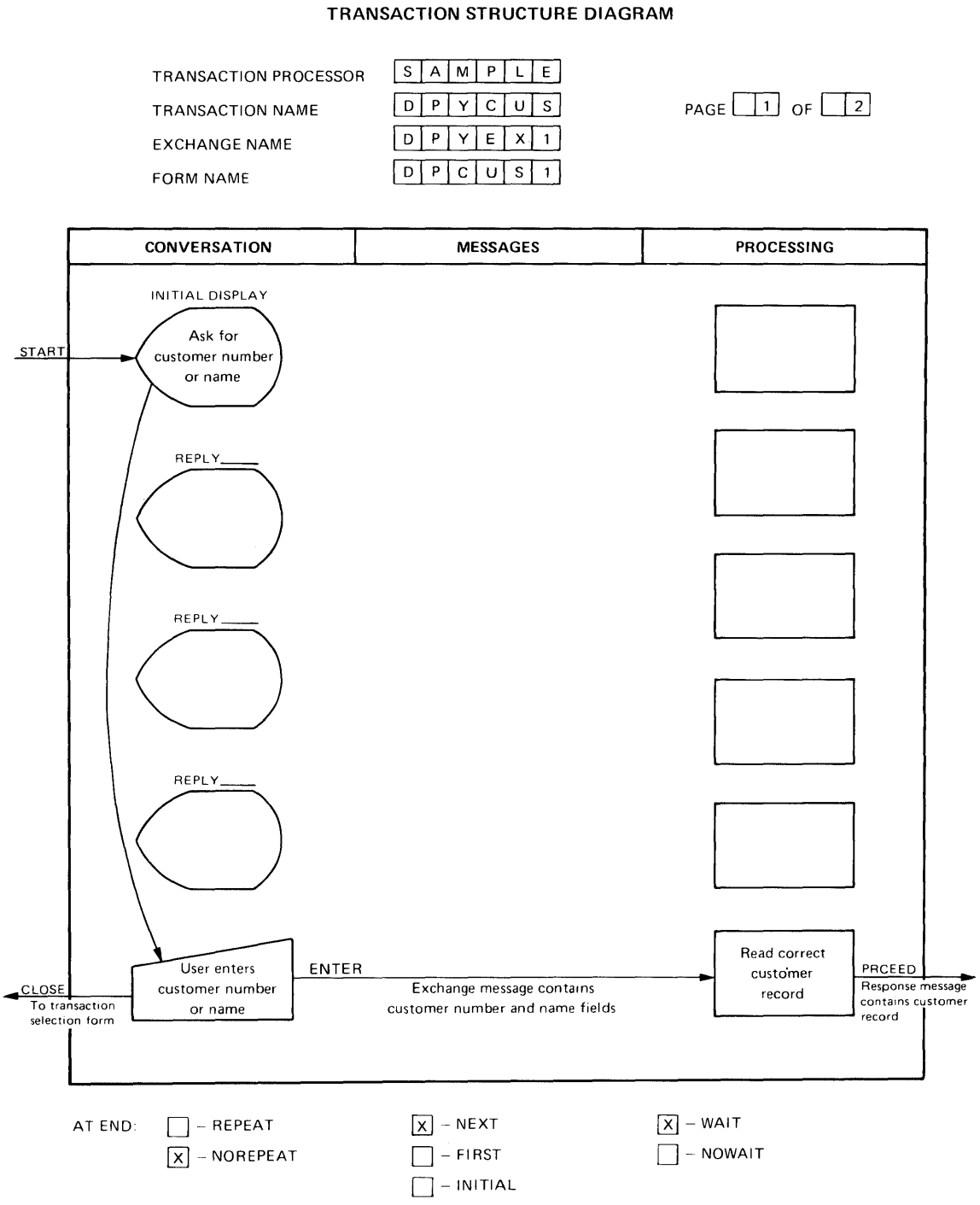


Figure 15-8 Browsing with Two Indexes

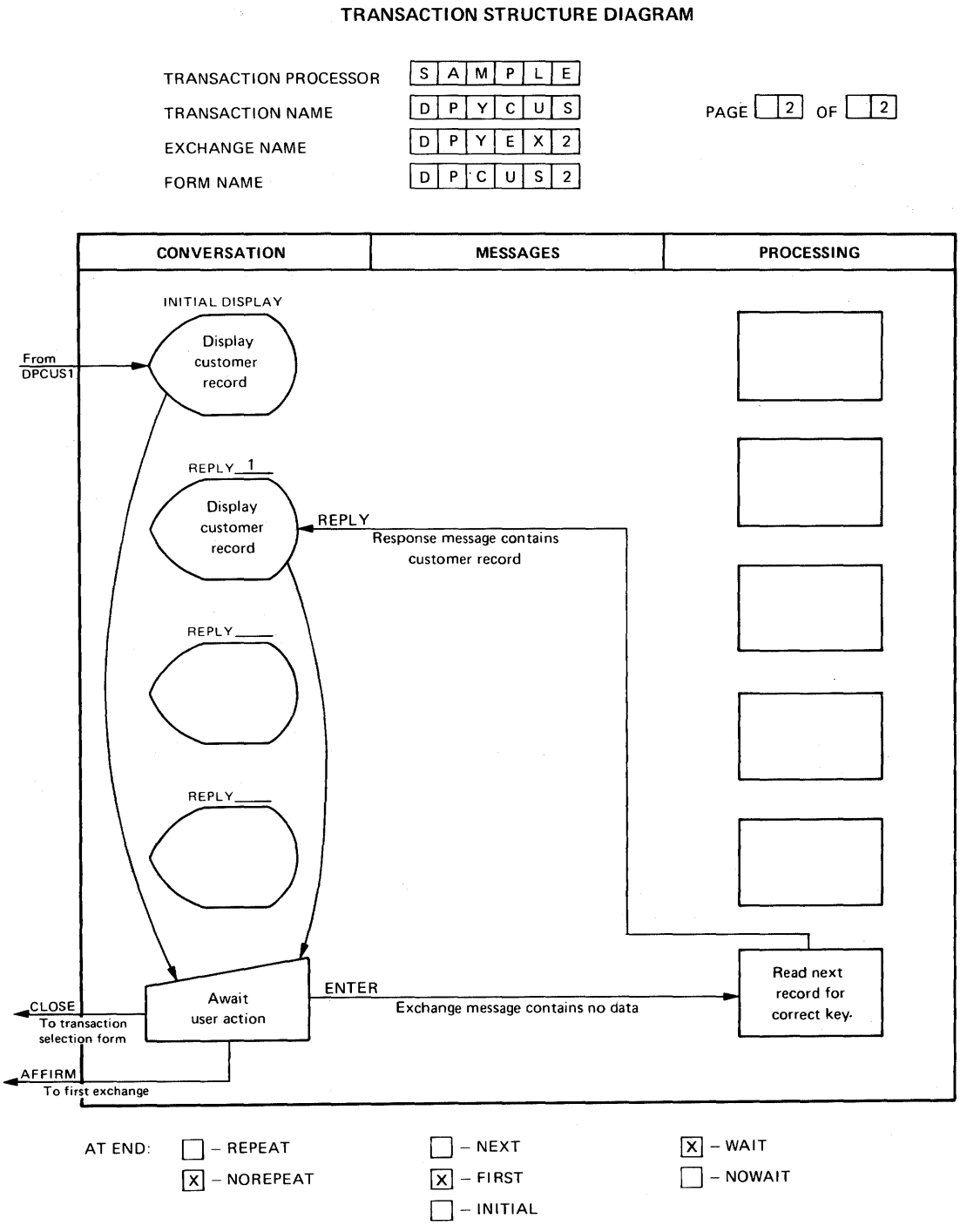


Figure 15-8 (Cont.) Browsing with Two Indexes

15.8 ERROR MESSAGES

Error messages are required in the preceding designs before they can be used in a commercial application. Consider the previous example with error messages included. This final design is shown in Figure 15-9.

Error messages are almost always implemented with REPLY response messages, because the user must usually correct his input and enter it again.

The text of an error message can be generated in three ways:

1. The text can be specified as the VALUE clause in the definition of a DISPLAY field. (This field should not include the NOBLANK option.) Naming this field in the WRITE clause of a REPLY definition causes the text to be displayed. The text is automatically removed from the screen when another reply is activated.
2. The text can be specified directly in the WRITE clause of a REPLY statement. In this way different messages can be made to appear in a field depending on the activated reply definition.

   The field is usually a DISPLAY field without the NOBLANK option, so the message is erased when another reply is activated. But if you wish, the error message can appear in any field and in either the display or forms area of the form.
3. The error message text can be inserted into the response message by a TST and moved to a field on the screen by a REQUEST keyword in the WRITE clause of a REPLY definition. This method gives the TST programmer flexibility in specifying error message text, but it makes the documentation of error messages more important – they are no longer in the form definition.

Method 1 requires as many fields on the screen as there are different error messages. Method 2 can place several alternative messages in the same field, but it requires a separate reply definition for each different message text. Method 3 requires only a single field and a single reply definition to handle an unlimited variety of error messages.

The design in Figure 15-9 uses Method 3.

The form used by the first exchange in Figure 15-9 must therefore have a reply definition added. This reply is activated by a TST, via a REPLY response message, in instances where the specified record cannot be located or read.

The form used by the second exchange has a reply definition, but it is inappropriate for the display of error messages. It treats the incoming REPLY message as a data record and divides it in fields to be displayed on the screen. A second reply is defined in this form to handle the error message. It is activated in situations like an end-of-file condition, where there are no further records for display.

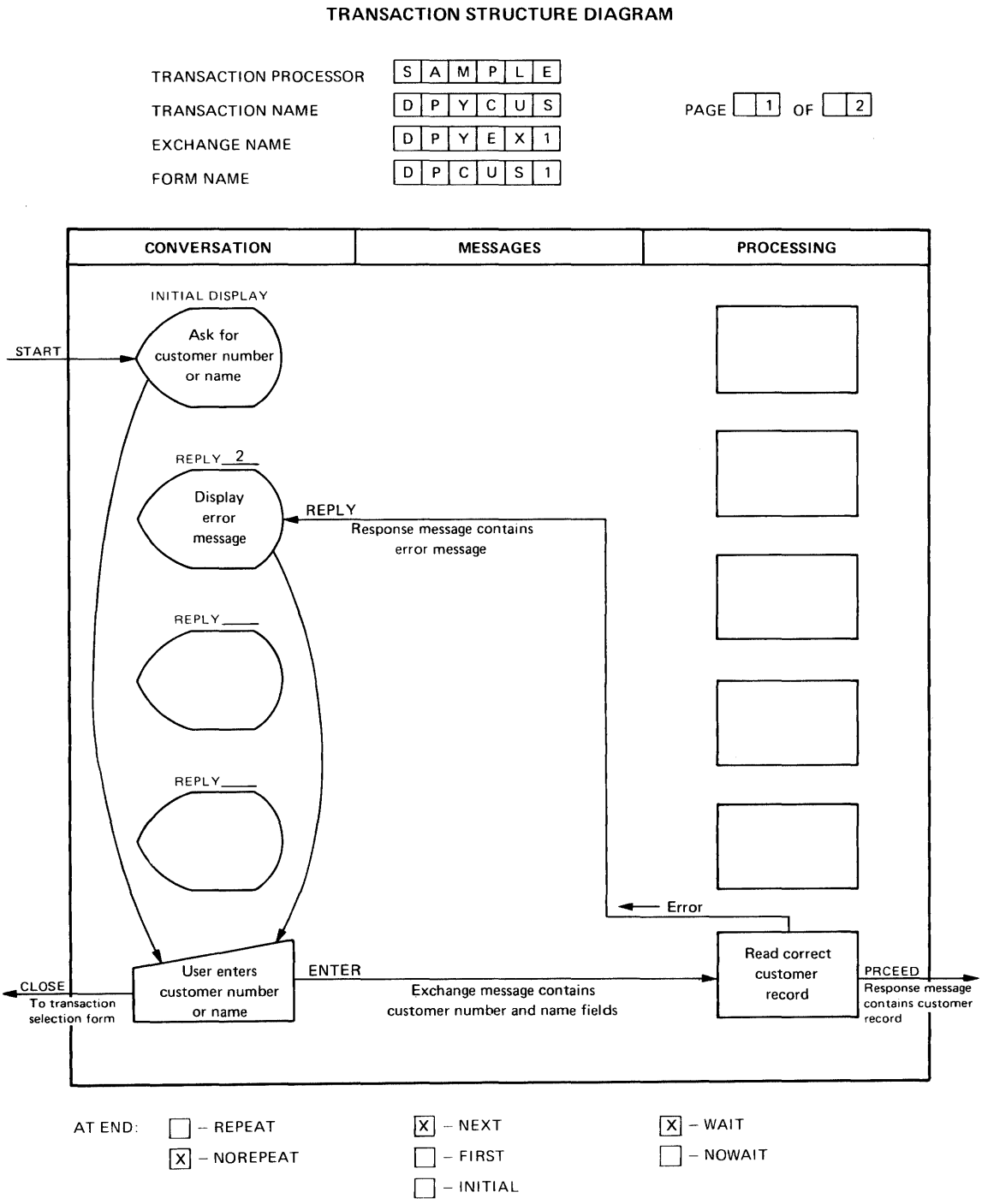


Figure 15-9 Adding Error Messages to Figure 15-8

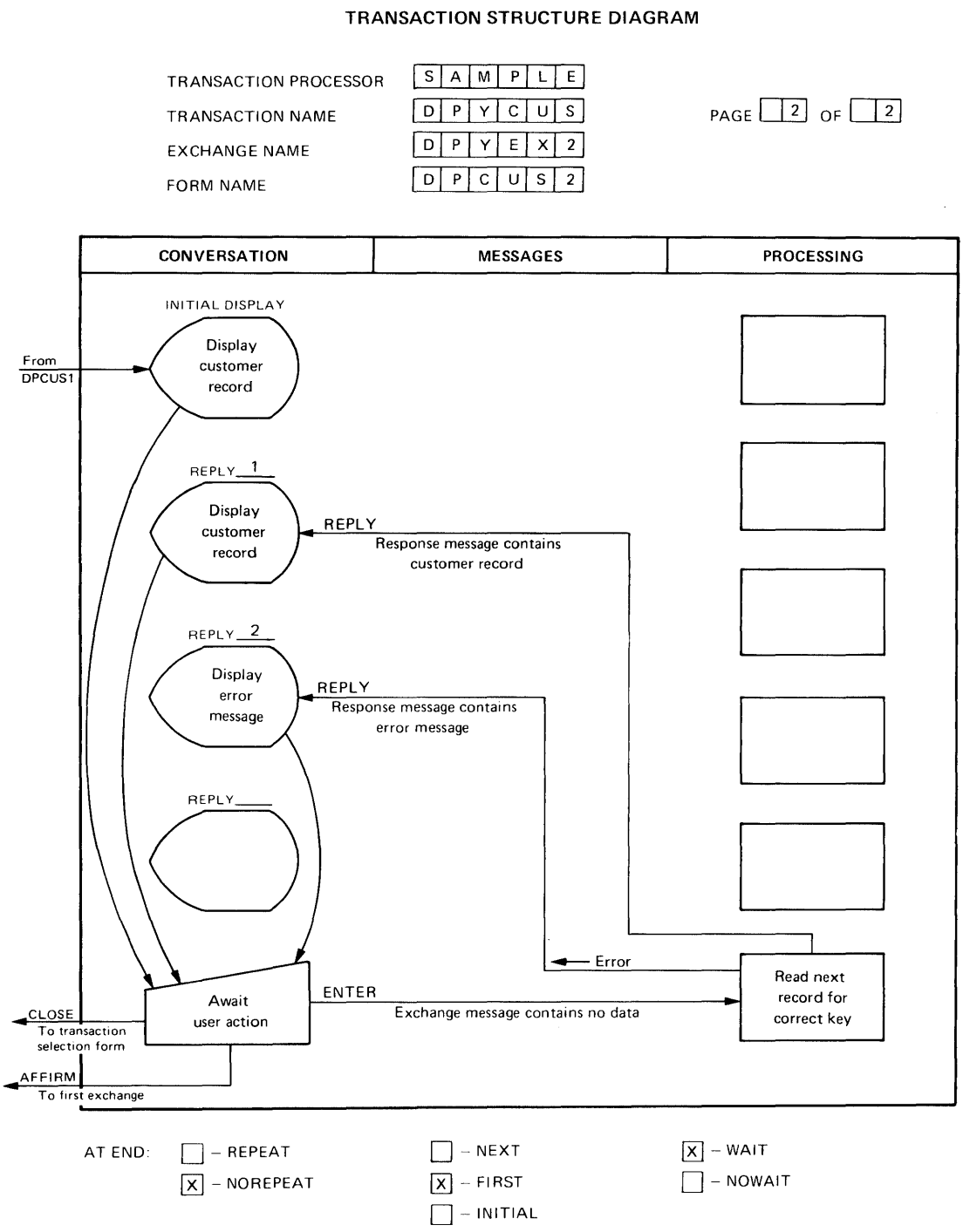


Figure 15-9 (Cont.) Adding Error Messages to Figure 15-8

CHAPTER 16
DOCUMENTING THE TRANSACTION DESIGN

16.1 STANDARDIZING TRANSACTION COMPONENTS

After laying out the overall flow of each transaction and defining transaction data structures, it is time to compare the transactions you have designed. The object of this comparison is to identify similarities between transactions. Similarities sometimes allow transaction components to be used for several transactions or portions of one component to be used in another component.

Similarities are found often in these transaction components:

- *TSTs.* Frequently, a portion of a TST is useful in another TST. Using the copying features of the TRAX text editor or the programming language can often save significant programming effort. See Section 19.5.
- *Forms.* Sometimes parts of a form can be used in two or more transactions. This is likely when several transactions operate on the same set of data – for instance, a set of data maintenance transactions operating on a master file. Such a set probably includes add, change, delete, and display transactions for the basic data record. With careful design, the forms for these transactions can share a considerable portion of their form definitions.
- *Data Structures.* Consistent data structures save considerable development time and make software maintenance easier. For example, standardizing data structures such as exchange messages and response messages allows their declarations in form definitions and TSTs to be coded once and used many times. This is such a time-saver that it helps to add filler fields to some data structures to make them compatible with others, and therefore sharable without modification.

Carefully document your list of required transaction processor components. You will refer to this list frequently as the implementation proceeds.

Give each form a six-character name. This name identifies the form within the transaction processor, so be careful to select meaningful mnemonic names.

Each TST and its corresponding TST station should also be assigned a six-character name. (You can use the same name to identify both the TST and its station as long as the name ends with a letter.) Again, attempt to select meaningful mnemonic names.

16.2 DEFINING STATIONS

Next, you must define the set of stations needed by the transaction processor.

Usually, you will have two types of stations in your transaction processor: terminal stations and TST stations. Occasionally, you may have other types of stations.

16.2.1 Terminal Stations

Your transaction processor needs a terminal station for each application terminal it serves. This rule also holds for clustered or multi-dropped terminals. Each terminal device, video display or printer, requires a station.

Use the form shown in Figure 16-1 to define the terminal stations. This form can later be used to enter the terminal station definitions to the transaction processor via the STADEF utility program.

For each terminal station, you must make the following design decisions:

1. *Station Name.* You must choose a station name (up to six characters) that uniquely identifies the station within the transaction processor. If you need a series of terminal stations with similar characteristics, you may use a station name of four characters and two asterisks – for instance, TERM**. The STADEF utility program expands this dummy name to a set of terminal station names by replacing the asterisks with a series of 2-digit numbers.
2. *Device Name.* You must associate a terminal device identifier with the station you are defining. A terminal device identifier of up to six characters is assigned to each attached application terminal during the TRAX system generation. The designated terminal will be served by this terminal station.
3. *Device Type.* If the terminal station is to initiate transactions, choose BOTH. If it is to print reports produced by transactions executed at other terminals, choose OUTPUT.
4. *System Messages.* If you want operating system messages (such as operator broadcast messages) to appear on this application terminal, enter YES.
5. *Work Classes* (Interactive Terminals Only). The work class you assign to this terminal determines the set of transactions that can be executed from the terminal. Definition of work classes is discussed in Section 16.3.
6. *Initial State* (Interactive Terminals Only). You may choose one of two displays to be used when the terminal is idle:
   - If you wish the terminal to display the first form of a transaction, choose the TRANSACTION option and enter the name of the transaction. This will be the only transaction executable from the terminal.
   - If you wish the terminal operator to select from a list of transactions, choose the INITIAL FORM option and enter the name of a transaction selection form. *Be sure the names of transactions displayed on the transaction selection form agree with the transactions the terminal's work class allows.*

The work class and initial state options for interactive terminals are important because they affect system security. Refer to Part One of this manual for a description of your choices and their effects.

16.2.2 TST Stations

Your transaction processor will need one station for each TST. If you allow multiple copies of a TST to execute at the same time, multiple station definitions are not necessary. Instead, use the "Number of Active Copies" parameter discussed later in this section.

Use the form shown in Figure 16-2 to define the TST stations. This form can later be used to enter the TST station definitions to the transaction processor via the STADEF utility program.

For each TST station, you must make the following design decisions:

1. *Station Name.* Choose a station name that identifies the station within the transaction processor. These names may be up to six characters and must end with a letter.
2. *Task Image File Specification.* Enter the name of the task image file that contains the TST's executable task image. The task image file need not exist at this time; just assign the name that will be used when the TST is coded and linked. The file specification can contain a device name, a UIC, a filename and extension, and a version number if desired. (See *TRAX Support Environment User's Guide* for discussion of file specifications.) If you do not enter portions of the file specification, the following default specifications will be used:

   Device: SY:
   UIC: [1,300]
   Extension: .TSK
   Version: Latest

TERMINAL STATION SPECIFICATION SHEET

Transaction Processor Name: ☐☐☐☐☐☐

Station Name	Device Name	Device Type	System Messages	Work Class	Run A Dedicated Transaction?
☐☐☐☐☐☐	☐☐☐☐☐☐	☐ – 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: ☐☐☐☐☐☐
☐☐☐☐☐☐	☐☐☐☐☐☐	☐ – BOTH ☐ – OUTPUT	☐ – YES ☐ – NO	☐☐☐☐☐☐	☐ Yes – Transaction Name: ☐☐☐☐☐☐ ☐ No – Initial Form Name: ☐☐☐☐☐☐

Figure 16-1 Terminal Station Specification Sheet

TST STATION SPECIFICATION SHEET

Transaction Processor Name: ______

Station Name	Station Priority	Task Image File Specification	No. Active Copies	Serially Reusable?
		:[,] . ;		☐ – 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
		:[,] . ;		☐ – YES ☐ – NO
		:[,] . ;		☐ – YES ☐ – NO

Figure 16-2 TST Station Specification Sheet

3. *Station Priority.* The normal priority of a TST station is 128. You may choose another priority; larger priority numbers indicate higher TST priority and vice-versa. When TSTs are waiting for execution, the transaction processor activates the TST of the highest priority among those which can be satisfied by the available resources. Once they are activated, TSTs are not interrupted until they terminate of their own accord. You must keep this scheduling rule in mind as you modify TST priorities to adjust transaction processor behavior.
4. *Number of Active Copies.* Specify the maximum number of copies of a TST that can be active at a time. Multiple copies are only activated if there are a sufficient number of exchange messages arriving to keep multiple copies busy. If you specify a large number of copies, exchange messages will not have to wait at the TST station before TST copies are activated to process them. But you may want to specify a single copy, if you want to exchange messages to be processed one at a time or if multiple TSTs might encounter such bottlenecks as contention for memory or contention for data file records.
5. *Serially Reusable.* This choice depends on the programming language selected for the TST and how your programmers use features of that language. If you tell the transaction processor that a TST is serially reusable, it may not fetch a fresh copy of the TST task image to begin processing each exchange message. This means that a serially reusable TST must completely initialize its own variables and data structures. Variables and data structures that are initialized only when the TST is linked do not allow a TST task image to be reused.

   A problem often develops with COBOL programmers because they do not realize that VALUE IS clauses in the DATA DIVISION of their programs are initialized at link time, rather than execution time. To ensure that a COBOL program is serially reusable, the programmer should only use VALUE IS clauses for data items that are never changed during program execution. COBOL programmers should use MOVE statements at the start of the PROCEDURE DIVISION to initialize data items that may be changed during program execution.

   BASIC-PLUS-2 variables are automatically initialized to zeros and null strings at execution time rather than link time. BASIC-PLUS-2 programs are therefore *always* serially reusable.

16.2.3 Special Station Types

Besides terminal and TST stations, you may need special purpose stations:

- *Batch Submit or Batch Slave Stations.* If your transaction processor needs an interface to TRAX batch processing facilities, you must define batch submit or batch slave stations.
- *Master Link or Slave Link Stations.* If your transaction processor must exchange data with other transaction processors, you must define some master link or slave link stations.
- *Mailboxes.* If your transaction processor needs facilities for the deposit and recall of mailbox messages, you must define one or more mailbox stations.

Part One of this manual provides you with additional details concerning these special types of stations. If you require any master link stations, define them on the form shown in Figure 16-3. If you require any slave link stations, submit or slave batch stations, or mailbox stations, define them on the form shown in Figure 16-4. This data can then be entered via the STADEF utility program, just as for the other station types.

16.3 WORK CLASSES AND USER AUTHORIZATIONS

When you define each interactive terminal station (Section 16.2.1), you specify one of three initial states:

1. If only one transaction is to be executed from the terminal, you specify the name of that transaction.
2. If anyone at the terminal is to be allowed access to a common set of transactions, you specify two things: the name of a transaction selection form and the name of a work class.
3. If various people sitting at the terminal are to be allowed access to different sets of transactions, you specify the same two parameters. But the terminal's work class must be SIGNON and one of the transactions permitted by the SIGNON work class (possibly the *only* available transaction) must be the SIGNON transaction.

MASTER LINK STATION SPECIFICATION SHEET

Transaction Processor Name: ☐☐☐☐☐☐

Station Name	Connected to Slave Link Type		Number of Sublinks		
☐☐☐☐☐☐	☐ – Node Name:	☐☐☐☐☐☐	☐☐☐	Slave TP Name	☐☐☐☐☐☐
	☐ – Local		☐☐☐	Slave TP Name	☐☐☐☐☐☐
	☐ – 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	☐☐☐
☐☐☐☐☐☐	☐ – Node Name:	☐☐☐☐☐☐	☐☐☐	Slave TP Name	☐☐☐☐☐☐
	☐ – Local		☐☐☐	Slave TP Name	☐☐☐☐☐☐
	☐ – 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	☐☐☐
☐☐☐☐☐☐	☐ – Node Name:	☐☐☐☐☐☐	☐☐☐	Slave TP Name	☐☐☐☐☐☐
	☐ – Local		☐☐☐	Slave TP Name	☐☐☐☐☐☐
	☐ – 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	☐☐☐
☐☐☐☐☐☐	☐ – 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 16-3 Master Link Station Specification Sheet

SPECIAL PURPOSE STATION SPECIFICATION SHEET

Transaction Processor Name: ☐☐☐☐☐☐

Station Name	Station Type
☐☐☐☐☐☐	☐ – 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
☐☐☐☐☐☐	☐ – 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
☐☐☐☐☐☐	☐ – 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
☐☐☐☐☐☐	☐ – 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 16-4 Special Purpose Station Specification Sheet

If you used the *second* option, you must define the work classes that you assigned to the terminals.

If you used the *third* option, you must define the SIGNON work class assigned to idle terminals, the user identifiers and passwords used by the SIGNON transaction, and the work classes assigned to each user.

Use the specification sheet shown in Figure 16-5 to define work classes. List all of the transactions defined in the transaction processor down the left side of the sheet. List the work classes you wish to define across the top. Then place an "X" in the appropriate row and column to indicate the transactions included in each work class. The data on this sheet can be entered later into the WORDEF utility program, which installs it in the transaction processor.

Use the specification sheet shown in Figure 16-6 to define user identifiers and passwords and to assign work classes to each user. This specification sheet is similar to the work class form in Figure 16-5, except you must enter user identifiers and passwords on the left and work classes across the top. Again, place an "X" in the appropriate row and column to indicate users who can access each work class. A user can access at most 64 work classes. This data can later be entered into the transaction processor via the AUTDEF utility program.

16.4 TRANSACTION DEFINITIONS

Now that the components of each transaction are defined, you can assemble the transaction definitions that provide an overall framework.

To define a transaction within a transaction processor, you must supply the data itemized in Figure 16-7. Use one of these specification sheets to define each transaction.

There are two parts to the transaction definition sheet. The first part asks you to specify parameters for the entire transaction. In the second part, you define each of the exchanges for the transaction. (If your transactions have more exchanges than can be defined on this sheet, use the continuation sheet shown in Figure 16-8.)

The data on these sheets is entered into the transaction processor via the TRADEF utility program.

16.4.1 Overall Transaction Parameters

Refer to the upper portion of Figure 16-7. On this part of the sheet, you must specify the following transaction parameters:

1. *Transaction Name.* You must assign a six-character name to the transaction. This name will appear on transaction selection screens, and users will select transactions with these names. Transaction names must be unique within the transaction processor.
2. *Exchange Recovery.* Specify YES if you want the transaction to use exchange recovery. (Exchange recovery is discussed in Part One of this manual.) Remember that exchange recovery has side effects:
   - System overhead increases during the execution of the transaction.
   - Messages generated by TSTs are delayed until an exchange's processing phase terminates.
   - Staged files are usually required.

   Use exchange recovery only after you fully understand its benefits and side effects.
3. *Log Exchange Messages.* Specify YES if you wish the transaction processor to record each exchange message from this transaction as a log entry in the system journal. You can use this feature for system debugging and as an audit trail during system operation.
4. *Log Independent Messages.* Specify YES if you wish to log response, mailbox, and report messages. You can use this feature for system debugging and as an audit trail during system operation.
5. *Maximum Size of Exchange Message.* Determine this parameter by inspecting the transaction data structures you have prepared. Select the largest exchange message in the transaction and record its length here.
6. *Transaction Workspace Size.* Enter the size of the workspace that you defined for this transaction.

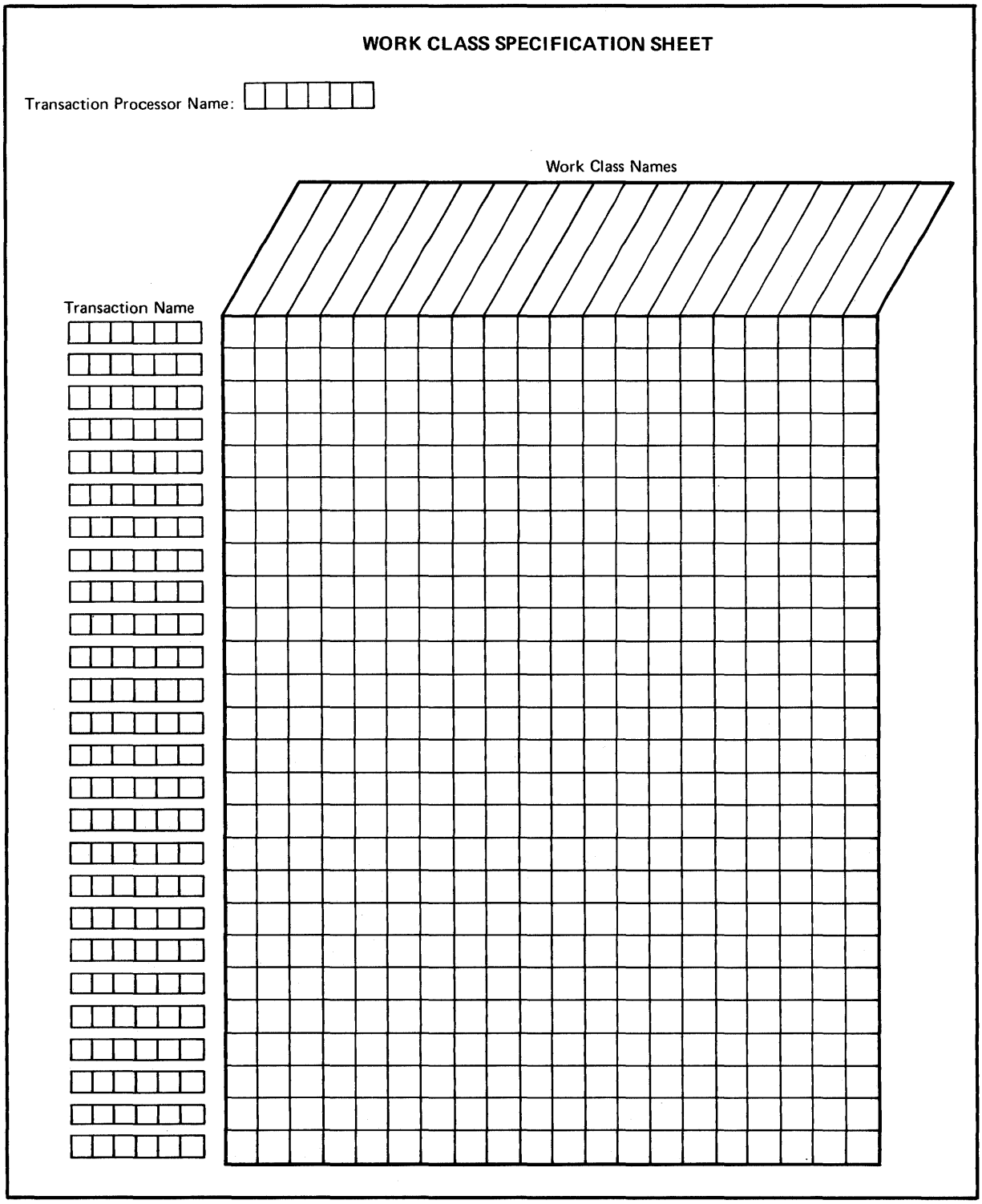


Figure 16-5 Work Class Specification Sheet

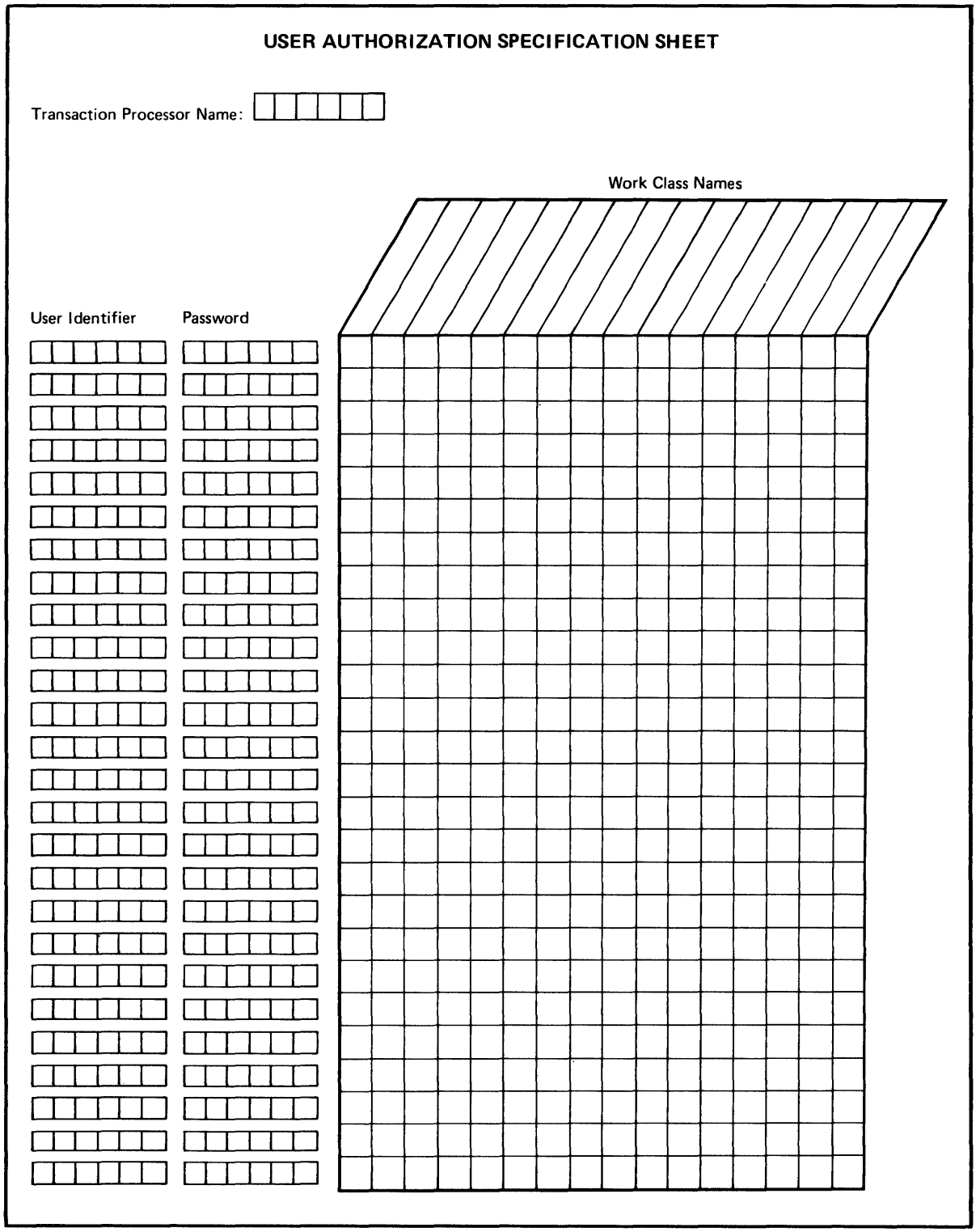


Figure 16-6 User Authorization Specification Sheet

TRANSACTION SPECIFICATION SHEET

Transaction Processor Name: ☐☐☐☐☐☐

Transaction Name: ☐☐☐☐☐☐

Exchange Recovery? ☐ – YES ☐ – NO

Log Exchange Messages? ☐ – YES ☐ – NO

Log Other Station Messages? ☐ – YES ☐ – NO

Maximum Size of Exchange Message: ☐☐☐ 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: ☐☐☐☐ blocks

Divide Transaction Workspace Size by 64 and round up: ☐☐☐☐ blocks

Enter System Workspace Size: ☐☐☐☐ blocks

Add to find Transaction Slot Size: ☐☐☐☐ 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
☐☐☐☐☐☐	☐☐☐☐☐☐	☐☐☐☐☐☐	☐ WAIT	☐ REPEAT	☐ INITIAL	☐☐ MINS
		☐☐☐☐☐☐	☐ NOWAIT	☐ NOREPEAT	☐ FIRST	
		☐☐☐☐☐☐			☐ NEXT	
		☐☐☐☐☐☐				
		☐☐☐☐☐☐				
		☐☐☐☐☐☐				
		☐☐☐☐☐☐				
		☐☐☐☐☐☐				
☐☐☐☐☐☐	☐☐☐☐☐☐	☐☐☐☐☐☐	☐ WAIT	☐ REPEAT	☐ INITIAL	☐☐ MINS
		☐☐☐☐☐☐	☐ NOWAIT	☐ NOREPEAT	☐ FIRST	
		☐☐☐☐☐☐			☐ NEXT	
		☐☐☐☐☐☐				
		☐☐☐☐☐☐				
		☐☐☐☐☐☐				
		☐☐☐☐☐☐				
		☐☐☐☐☐☐				

Figure 16-7 Transaction Specification Sheet

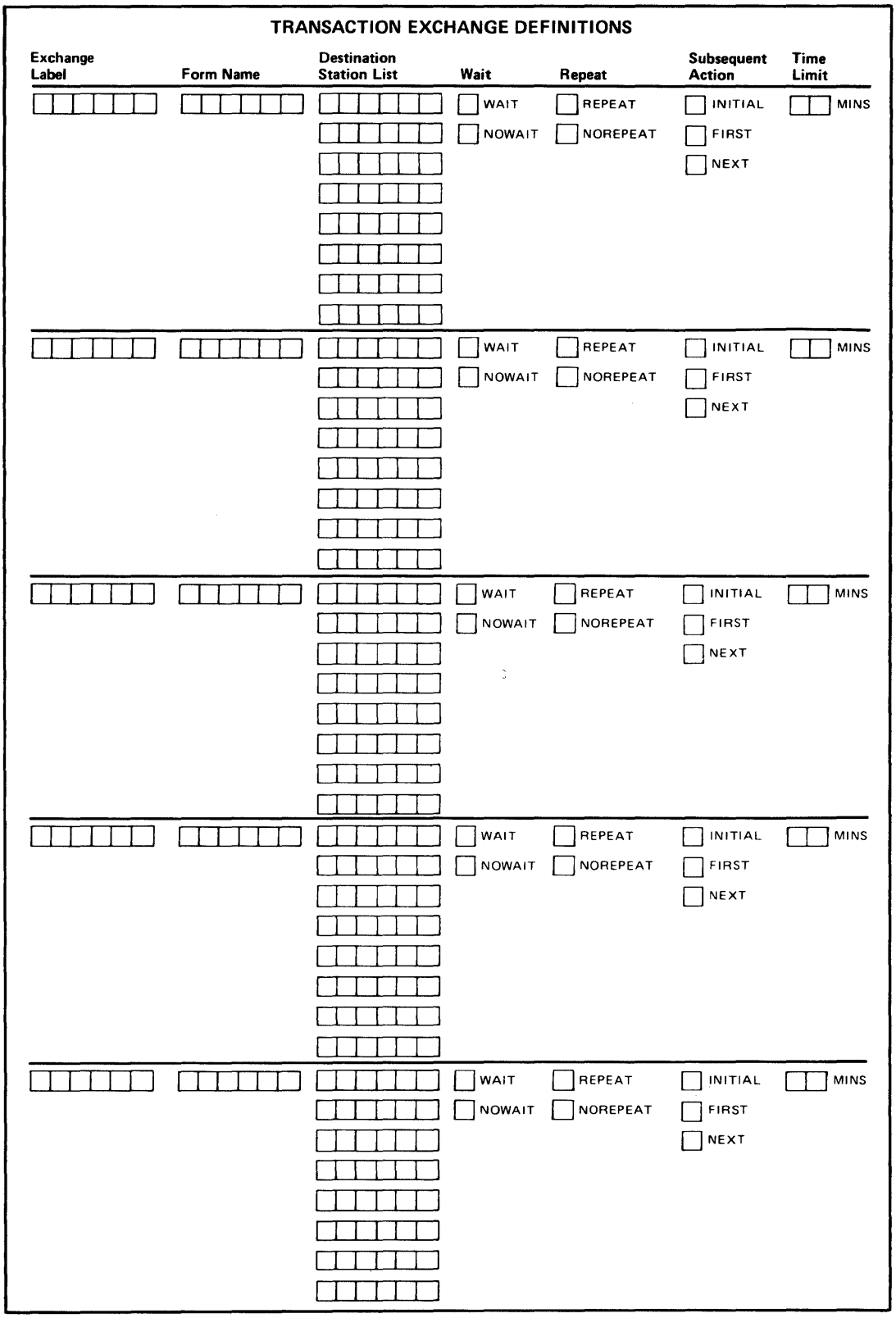


Figure 16-8 Transaction Specification Sheet Continuation

7. *System Workspace Size.* A system workspace is necessary only if your transaction uses exchange recovery or staged files. This is the area where the transaction processor keeps messages and updated data base records until successful exchange or transaction completion. Compute the system workspace size using the worksheet shown in Figure 16-9. Then enter the result here.

16.4.2 Exchange Definitions

Refer to the lower portion of Figure 16-7. In this part of the specification sheet, you must define the exchanges that make up the transaction. For each exchange, you must specify:

1. *Exchange Label.* You must give each exchange in the transaction a unique label. This name must be unique within the transaction but may be identical to exchange names in other transactions. TST programmers use these labels to select successor exchanges.
2. *Form Name.* Most transactions are initiated from application terminals, and most exchanges in these transactions must have a form. If the exchange you are defining needs a form, enter the six-character name of the form.

   Exchanges do not need a form in the following circumstances:

   - *None of the exchanges in a transaction* need form names if the transaction is initiated only by source stations other than application terminal stations – for instance, by a TST.
   - *The first exchange of a transaction* does not need a form name if the transaction selection forms supply exchange messages and if the exchange is not entered from within the transaction.

3. *Routing List.* Enter the list of stations to which the exchange message must be sent. Each time the exchange is executed, its exchange message is assigned this routing list. The routing list may subsequently be changed by the TSTs that process that message.
4. *WAIT Option.* The normal selection for this option is WAIT, meaning that a response message is expected from one of the TSTs processing the exchange message. Be sure you thoroughly understand this option before selecting NOWAIT.
5. *REPEAT Option.* This is one of two important options that determine which exchange follows the exchange being defined. This option is tested when the terminal operator presses an enabled AFFIRM key or when a TST sends a PRCEED response message. If you specify REPEAT, the exchange is followed by another execution of the same exchange.
6. *Subsequent Action.* This is the second of the two options mentioned before. You have three choices for the subsequent action option:

   - NEXT selects the next exchange in the transaction definition as the successor to this exchange.
   - FIRST terminates the transaction after the current exchange is finished. Then the transaction processor prepares to execute the transaction again by displaying the form from the first exchange.
   - INITIAL also terminates the transaction after the current exchange is finished, but the transaction processor causes the terminal to revert to its initial state. As you recall, this may vary depending on the definition of the corresponding terminal station.

   This option is tested in the following four cases:

   - When a terminal user presses an enabled AFFIRM key and the REPEAT option is not specified.
   - When a terminal user presses an enabled STOP-REPEAT key, regardless of whether the REPEAT option is specified.
   - When a TST issues a PRCEED response message and the REPEAT option is not specified.
   - When a TST issues a STPRPT response message, regardless of whether the REPEAT option is specified.

CHAPTER 17
DESIGNING FORMS

Once you determine the overall structure of the transactions in your application, you can design the forms for the transaction.

Before you begin designing TRAX forms, you should review the information in Chapter 5 and read the *ATL Language Reference Manual.*

Remember that design requirements will vary depending on the purpose of the form and the terminal where it will be used. Entry forms, transaction selection forms, and report forms require different design.

This chapter covers several factors involved in designing a form:

- The functions of entry forms
- Basic form layout
- Initial field values
- Building the exchange message
- Designing reply definitions
- Special purpose forms

17.1 REVIEWING THE FUNCTIONS OF ENTRY FORMS

Entry forms are the most complex kind of form to design. This is because they must serve many purposes during transaction execution:

- They must provide a functional, pleasing design that allows the user to enter and read information easily.
- They must accept data in response messages and include this data in the display for the user.
- They must build exchange messages in a specified format from data entered by the user.
- They must accept reply messages that instruct them to alter the data displayed for the user and change the function keys available to the user.

Sections 17.1 through 17.6 discuss issues that will help you satisfy these requirements for entry forms.

17.2 THE BASIC FORM LAYOUT

Your first step is to devise a basic layout of fields on the form. These fields may be either data entry fields or prompt fields. You must consider not only the fields' positions and sizes, but also their attributes.

Details of typographic layout and field positioning are important, just as with ordinary forms. A form is more readable and easier to use with careful graphic design.

Don't become entranced with cleverness. Clever forms design sometimes leads to better forms, but more often it leads to complex forms that are harder to understand and use. Clever arrangements of fields, for example, are counterproductive if they do not correspond with the user's train of thought as he enters data. Default rules are worse than none if the user cannot remember them. A reduction of user keystrokes may be detrimental to overall productivity if the design requires more thought from the clerical user.

Field attributes are powerful tools for controlling user data entry. Among other things, you can:

- Protect fields against data entry, or open them to data entry.
- Require entries in certain fields or make them optional.
- Restrict the characters that can be entered in a field.
- Cause a field to be right- or left-justified.

You should arrange data entry fields so that frequently used fields are near the top of the form and optional fields are near the bottom; otherwise the user must skip past unused fields to reach required fields further down the form.

Make sure that each data entry field is visible to the user. It is difficult to enter a field that is indistinguishable from its surroundings and whose position and size are unclear. TRAX has three ways of making fields visible:

1. *Reverse Video.* TRAX video display terminals can display fields in either white-on-black or black-on-white. The black-on-white style is called "reverse video." Designating data entry fields as reverse video fields makes them visible without introducing other data editing complications.
2. *The CLEAR Character.* A CLEAR character fills the field when the form is first displayed. It is also placed in the field (or in a portion of the field) when the user employs a DELETE-type editing key. Thus the CLEAR character effectively marks the position and size of the field. Periods, hyphens, and underscore characters are excellent choices for use as a CLEAR character. (Remember, though, that any CLEAR characters remaining in the field when the exchange message is built are included in the message, and the exchange's TSTs must edit them out.)
3. *Initial Values.* You might also use initial values to mark a field's size and position. Text inserted in a field as an initial value, though, is not replaced on the screen when the user uses a DELETE-type editing key. Initial values override any CLEAR character that you may have defined for a field.

When you are satisfied with your basic form design, sketch it on the specification sheet shown in Figure 17-1. This sheet gives you a grid the size of a video terminal screen, with the rows and columns labeled for reference.

At the top of the sheet, fill in the name of the form. Mark the box labeled "Initial Display" to distinguish this sheet from ones you may later create for replies.

The sheet also includes questions such as the set of function keys to be enabled, whether the terminal warning bell should be sounded when the form is first displayed, positioning of the cursor, and so forth. Check the appropriate "enabled" or "disabled" boxes for the function keys, and fill in the other parameters where required.

17.3 INITIAL FIELD VALUES

Many fields must contain text when the form is first displayed. This text is called "initial field values" and can come from two sources:

1. *Form Definition.* Text can originate in the form definition, as either literal text or built-in variable data items recognized by ATL. The latter include the name of the terminal station being used, the transaction being used, the time of day, and so forth. Text specified in this way is always displayed without modification.
2. *Response Message.* Text can originate in the response message that causes the form to be invoked. For instance, a PRCEED response message from a preceding exchange could contain data destined for display as part of the current form. The format of these response messages was discussed in Section 14.2.1. During forms design, you must decide where this data is to be placed on the form. This placement is done on a field-by-field basis.

NOTE

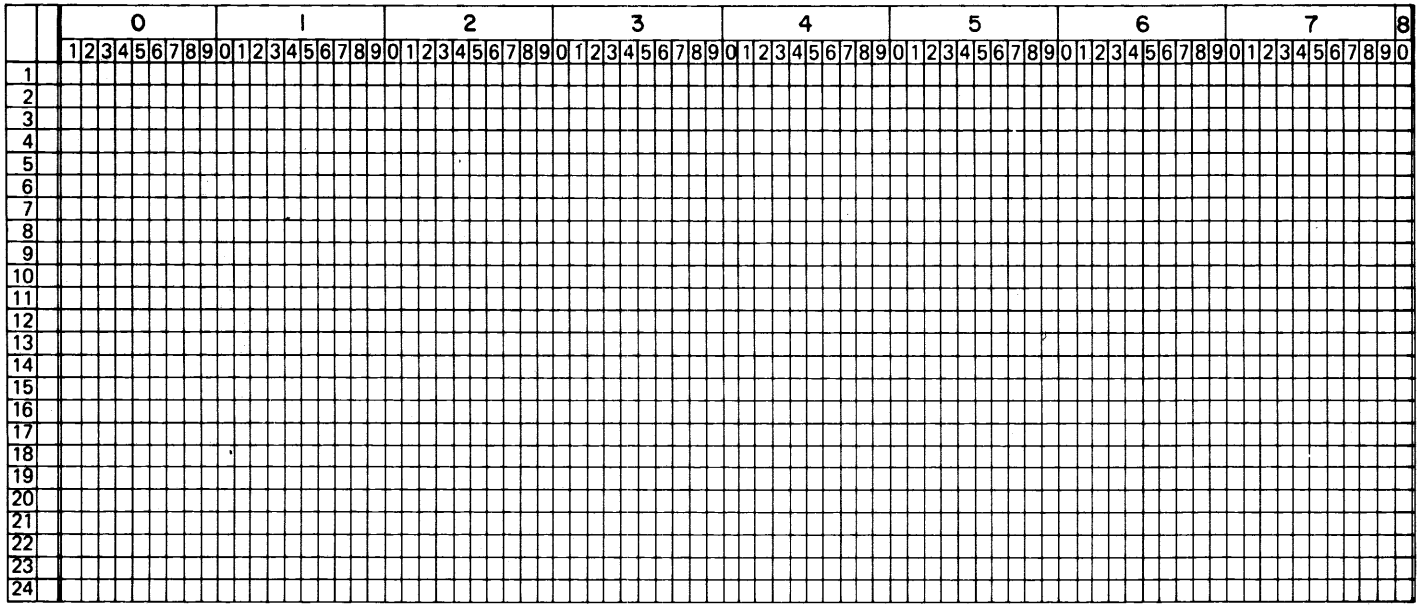
The second option is only available in exchanges entered as a result of a response message – not in exchanges entered as a result of a system function key.

FORM SPECIFICATION SHEET

Transaction Processor: ☐☐☐☐☐☐ ☐ – Initial Display ☐ – Reply Number ☐☐☐ Form Name: ☐☐☐☐☐☐

Split:

Lines Display Area



Bell: Rung for ☐☐.☐☐ periods

Cursor: Positioned at field ☐☐

System Function Keys	Enabled	Disabled
ABORT	☐	☐
CLOSE	☐	☐
AFFIRM	☐	☐
STOPREPEAT	☐	☐

User Function Keys	Enabled	Disabled
ENTER	☐	☐
KEYDOT	☐	☐
KEY00	☐	☐
KEY01	☐	☐
KEY02	☐	☐
KEY03	☐	☐

Use blue ink to give instructions, field types etc.
Use red ink to show actual text that appears on screen

Figure 17-1 Video Terminal Forms Specification Sheet

17.4 BUILDING THE EXCHANGE MESSAGE

Working from your exchange message layout (Figure 14-2), decide which fields on the form are the sources of the data in the exchange message.

Not all the data in the exchange message needs to come from fields entered by the user. You may also include the following data in the exchange message:

- Text in protected fields (that is, those not accessible to the user)
- Literal text coded into the form definition
- Identifiers assigned to user function keys
- ATL built-in variable data items such as those discussed in Section 17.3.

17.5 DESIGNING REPLIES

Replies are a powerful, flexible feature of TRAX forms. Chapter 14 discussed the role of replies in a transaction structure. Having already devised a transaction structure diagram for each of your transactions, you know the replies each form needs. Now you must decide the details of how each reply operates.

Identify the replies in each form by number. You may use any numbers you wish, but it is better if you use low reply numbers. (The space occupied by a form definition depends, to a small extent, on the *highest* reply number you use in that form.) You should also define as few replies in each form as possible. Replies significantly increase the size of a form definition.

A response message invoking a reply can carry data supplied by the sending TST. (Message format is discussed in Section 14.3.) Specify where this data is to be placed on the form. This placement can only be done on a field-by-field basis.

A reply can also cause literal text or ATL built-in data items to be written in specific fields. If you wish this, you must include it in your reply design.

You may document a complex reply by drawing the form as it will appear after the reply is activated. Use another form specification sheet like the one you used to sketch the original form (Figure 17-1). Complete the extra information fields at the bottom of the sheet. At the top of the sheet, label your sketch as a reply and fill in the reply number.

17.6 SPECIAL PURPOSE FORMS

The design of special purpose forms is similar to those we have just discussed. Special considerations are presented in the following sections.

17.6.1 Output-Only (Report) Forms

Output-only (report) forms print data on a printer. The data is sent to the printer's station as a report message, with a specification of the form to be used.

Definitions of output-only forms usually have two sections:

1. Definitions for each field
2. Specifications for the portion of the report message used to fill each field

Output-only forms have no data entry fields and never generate exchange messages. Consequently, they never have replies. Their role is to take data from the incoming report message and print it in the format specified.

17.6.2 Transaction Selection Forms

Transaction selection forms are used to select a transaction. These forms are never part of a transaction, although they can collect data for the first exchange of a transaction.

Transaction names are usually presented in a set of menu fields for user selection. Alternatively, you can designate one of the fields on each transaction selection form where the user can enter the transaction name.

Transaction selection forms do not usually generate exchange messages. For special purposes, however, you can add ordinary data entry fields to a transaction selection form, and use these fields to generate an exchange message. If you do this, the first form in the chosen transaction is skipped: the exchange message generated by the transaction selection form is used in the processing phase of the first exchange.

17.7 WRITING THE FORM DEFINITIONS

When you have made the design decisions discussed in Sections 17.2 through 17.6, you are ready to write the form definitions.

Remember the form roles (Section 17.1) and write your form definitions so that they have a clearly defined section for each of those roles. This simplifies enhancement and maintenance of the forms.

When you compile each form with the ATL utility program, the compiler output includes much of the application documentation you need for the development effort. For example, the compiler prints the formats of exchange and response messages, as well as a mockup of the form as initially displayed and after each reply. This output saves you a significant documentation effort.

Once you are satisfied with a form definition, subsequent adjustments can be left to application programmers.

CHAPTER 18
EXAMPLES OF FORM DESIGN

This chapter presents two examples of form design. These two forms are taken from the change customer transaction in the TRAX Sample Application.

These examples are based on the following design documentation for that transaction:

- The overall transaction structure — Figure 18-1
- The format of the first exchange's exchange message — Figure 18-2
- The format of the first exchange's REPLY response message — Figure 18-3
- The format of the first exchange's PRCEED response message — Figure 18-4
- The format of the second exchange's exchange message — Figure 18-5
- The format of the second exchange's first REPLY response message — Figure 18-6
- The format of the second exchange's second REPLY response message — Figure 18-7

18.1 THE RELATIONSHIP BETWEEN THE TRANSACTION AND ITS FORMS

The relationship between the transaction and its form is shown in the transaction structure diagram, Figure 18-1.

18.1.1 Requirement for Two Forms

The most important relationship is the requirement for two forms. Notice that the transaction has two exchanges; it will therefore need two forms. These two exchanges and their corresponding forms serve the following purposes:

1. The first exchange asks the user to identify the customer whose record is to be read from the file and presented for change. The form for this exchange asks for a customer identification number.
2. The second exchange presents the existing data for the customer, allows the user to inspect and modify this data, and then places the updated data in the file. The form for this exchange must display the data retrieved by the first exchange and then allow the user to change the data.

An inspection of the transaction structure diagram can determine further characteristics of these two forms.

18.1.2 Characteristics of the First Form

The transaction structure diagram (Figure 18-1) shows four characteristics of the first form:

1. No initial values must be retrieved from a response message.
2. The ENTER and CLOSE keys are enabled.
3. A reply is required for displaying an error message.
4. An exchange message and a response message must be defined.

The first form does not require initial values for its fields. That is, the data displayed on the first form does not depend on the results from prior processing.

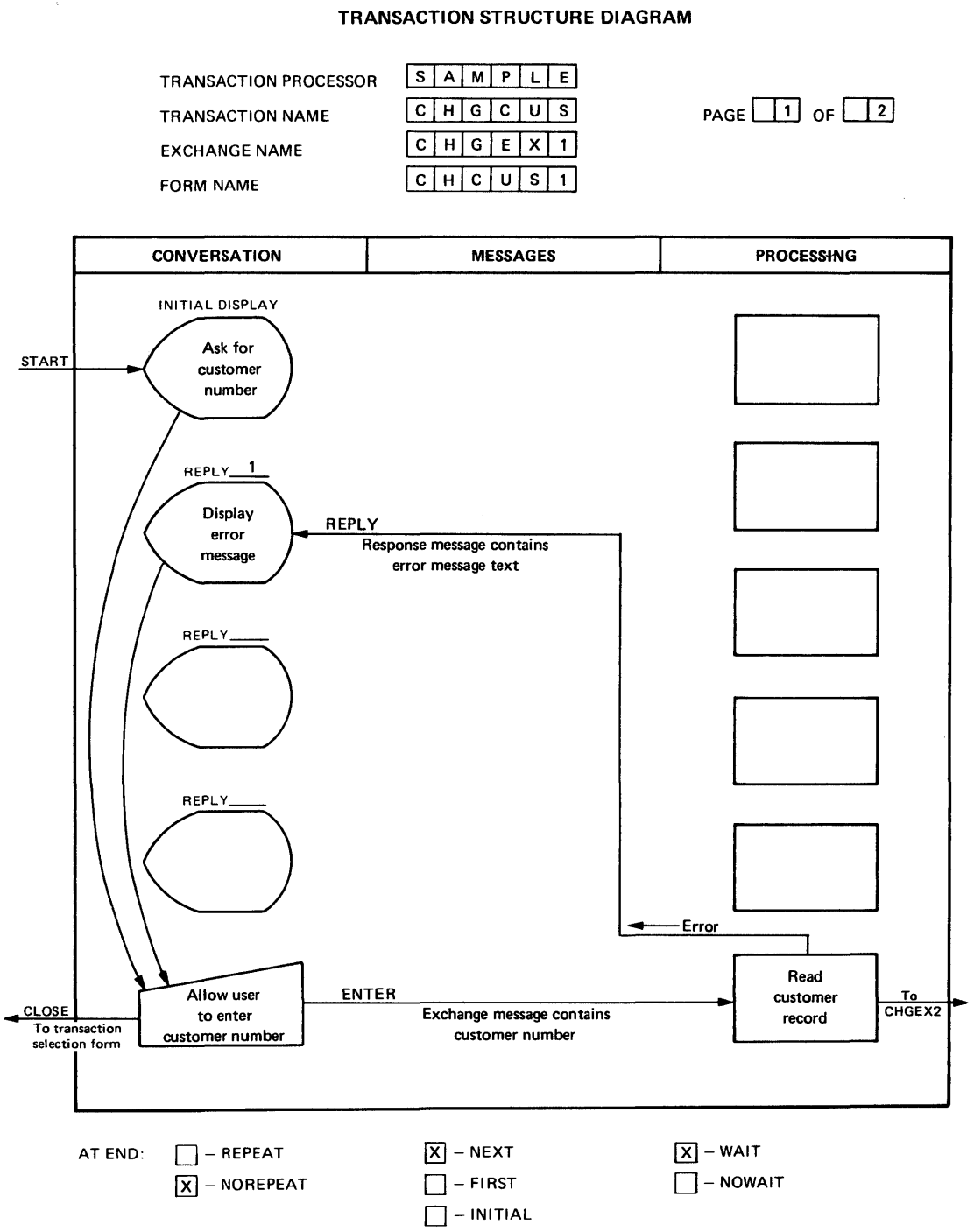


Figure 18-1 Structure of Change Customer Transaction

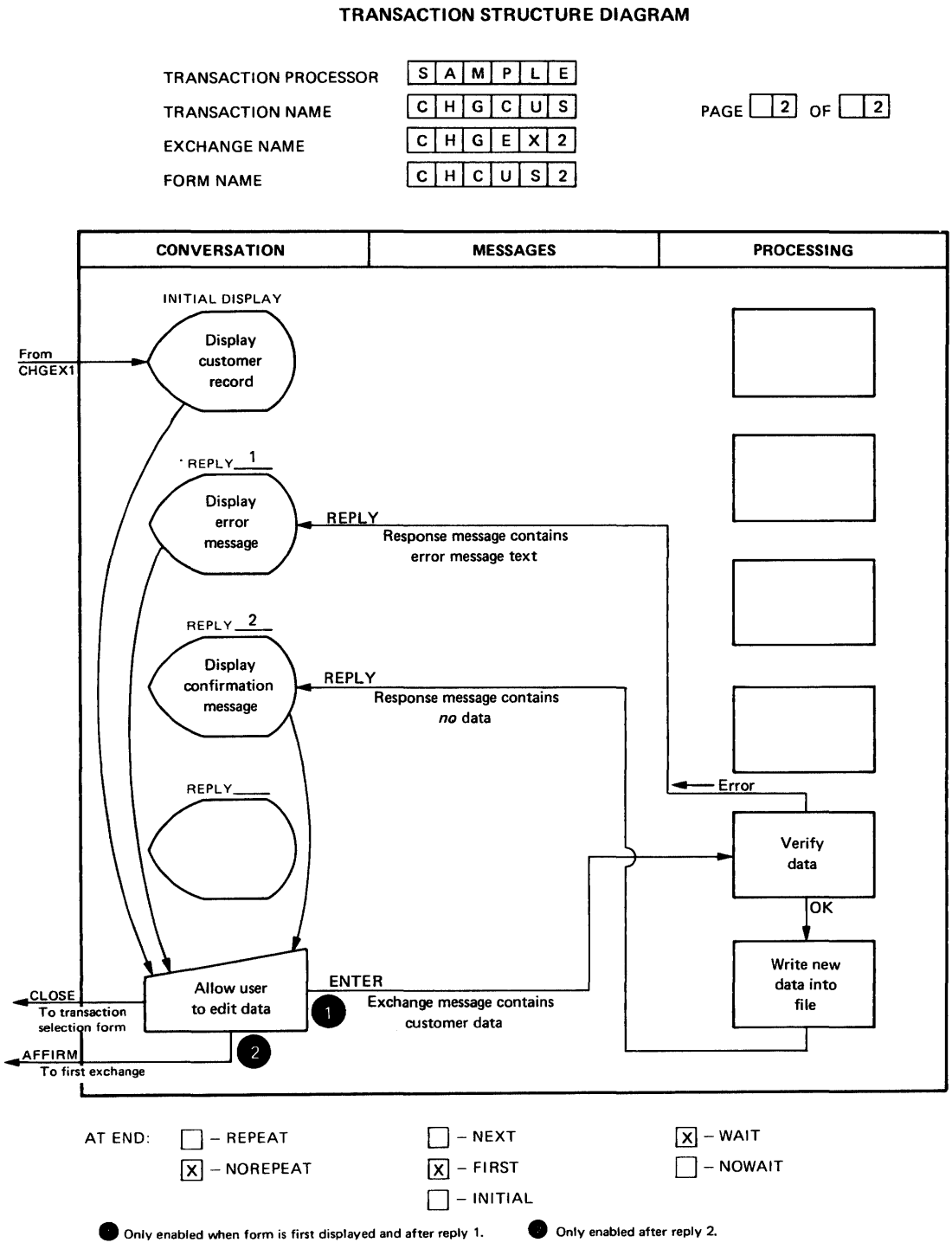


Figure 18-1 (Cont.) Structure of Change Customer Transaction

RESPONSE 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]

Type of Message

[X] – REPLY (Activates reply no. [ ][ ][2])

[ ] – PRCEED

[ ] – STPRPT

[ ] – CLSTRN

[ ] – ABORT (Activates reply no. [ ][ ][ ])

[ ] – TRNSFR (To exchange [ ][ ][ ][ ][ ][ ])

Field No.	Starting Byte	Length (Bytes)	Contents
1	1	80	Error Message (Part One)
2	81	80	Error Message (Part Two)
3			
4			
5			
6			
7			
8			
9			
10			
11			
12			
13			
14			
15			
16			
17			
18			
19			
20			
21			
22			
23			
24			
25			

Figure 18-3 Reply Message for Exchange 1

processing phase. The text of the displayed message is always the same and can be specified as part of the reply definition. It is necessary, however, for this second reply to disable the ENTER and CLOSE keys and enable the AFFIRM key.

The second form must process or generate four messages:

1. *The response message from the prior exchange* (Figure 18-4) provides the data displayed in the form's data entry fields. This message contains the set of fields from the customer file.
2. *The exchange message generated by the form* (Figure 18-5) contains the same data after it has been updated by the user.
3. *The response message for the first reply* (Figure 18-6) is like that in the first form: two 80-character error message lines, or 160 total characters.
4. *The response message for the second reply* (Figure 18-7) contains no data.

18.2 DESIGNING THE FIRST FORM

The first form of the change customer transaction is simple. This section discusses some design points and then presents the finished form definition.

18.2.1 Design Points

A sketch of the form's layout is shown in Figure 18-8. Note the following design features:

- *The application name and transaction name appear on the form.* The user can easily determine what is going on at a terminal displaying the form.
- *Fields are provided for error messages.* When you design forms, you must decide if you have room for an error message field opposite each data entry field or if one error message field can serve the entire form.
- *The data entry field has a prompt.* Avoid abbreviations unless lack of space requires them. The abbreviations you *do* use should be standard so their meaning is clear. Most abbreviations are understandable without the period, which takes up space. But you may find apostrophes (') useful: the abbreviation "pay't," for instance, is clearer than "payt". If you use upper- and lower-case prompts (Figure 18-8), your forms will be easier to read.
- *The prompt is separated from the data entry field by one or more spaces.* Use two spaces if you can, because that keeps the prompt from appearing to run into the data entry field.
- *The data entry fields are displayed in reverse video* so that their position and size are clear. (Other methods of marking the position and size of the data entry fields can also be used (Section 17.2). It is usually preferable to highlight the data entry field in reverse video rather than the caption. The caption is read easily without highlighting.
- *Function keys and their effect are explained to the user.* In Figure 18-8, the function key message is somewhat lengthy. In other applications, brief messages may suffice. But interactive forms should have *some* function key explanation for the user.
- *The form has proportion and attractive design.*

Other design features are annotated in Figure 18-8:

- The sketch indicates that this is the *initial display* of the form. It is distinguished from other sketches of the form that show the effects of replies.
- The enabled function keys are checked.
- If the terminal bell is to sound when the form is displayed, the duration of the sound (in bell periods) is shown on the sheet.

RESPONSE 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]

Type of Message

[ ] – REPLY (Activates reply no. [ ][ ][ ])

[X] – PRCEED

[ ] – STPRPT

[ ] – CLSTRN

[ ] – ABORT (Activates reply no. [ ][ ][ ])

[ ] – TRNSFR (To exchange [ ][ ][ ][ ][ ][ ])

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	3	Telephone Area Code
8	135	3	Telephone Exchange
9	138	4	Telephone Extension
10	142	20	Attention – Of
11	162	12	Credit Limit (9,999,999.99)
12			
13			
14			
15			
16			
17			
18			
19			
20			
21			
22			
23			
24			
25			

Figure 18-4 PRCEED Message For Exchange 1

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 2

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	3	Telephone Area Code
8	135	3	Telephone Exchange
9	138	4	Telephone Extension
10	142	20	Attention – Of
11	162	12	Credit Limit (9,999,999.99)
12			
13			
14			
15			
16			
17			
18			
19			
20			
21			
22			
23			
24			
25			

Figure 18-5 Exchange Message for Exchange 2

RESPONSE 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][2]

Type of Message

[X] – REPLY (Activates reply no. [ ][ ][2])

[ ] – PRCEED

[ ] – STPRPT

[ ] – CLSTRN

[ ] – ABORT (Activates reply no. [ ][ ][ ])

[ ] – TRNSFR (To exchange [ ][ ][ ][ ][ ][ ])

Field No.	Starting Byte	Length (Bytes)	Contents
1	1	80	Error Message (Part One)
2	81	80	Error Message (Part Two)
3			
4			
5			
6			
7			
8			
9			
10			
11			
12			
13			
14			
15			
16			
17			
18			
19			
20			
21			
22			
23			
24			
25			

Figure 18-6 REPLY Message 1 for Exchange 2

RESPONSE 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][2]

Type of Message

[X] – REPLY (Activates reply no. [ ][ ][1])

[ ] – PRCEED

[ ] – STPRPT

[ ] – CLSTRN

[ ] – ABORT (Activates reply no. [ ][ ][ ])

[ ] – TRNSFR (To exchange [ ][ ][ ][ ][ ][ ])

Field No.	Starting Byte	Length (Bytes)	Contents
1			
2	—	N O N	E —
3			
4			
5			
6			
7			
8			
9			
10			
11			
12			
13			
14			
15			
16			
17			
18			
19			
20			
21			
22			
23			
24			
25			

Figure 18-7 REPLY Message 2 for Exchange 2

```
                    TRAX FORMS DEFINITION (V1.0  )

Trans. Proc. Name:  SAMPLE        Device: VT62                          Page L-1
Form name:          CHCUS1        Length: 24                  02:00 PM  10-Jul-78
.....................................................................................
!*************************************************************
!
!       Definition of form CHCUS1
!
!       The first form for the Change Customer transaction
!
!       In this form, the system asks the user for a customer ID
!       number so that the desired customer record can be read
!
!*************************************************************

!*************************************************************
!
!       Group A Statements - Define General Parameters
!
!*************************************************************

DEFAULT
        CLEAR = " "
        ENABLE = CLOSE

FORM
        SPLIT=8                         18 lines of display area

!*************************************************************
!
!       Group B Statements - Define Fields on Screen
!
!*************************************************************
!=============================================================
!
!       These are the application fields
!
!       The two error text fields are used to display error
!       messages contained in response messages
!
!=============================================================

DISPLAY = 3,12
        VALUE = "Customer Master File Subsystem - Change Customer Transaction"
        LENGTH = 60
        ATTRIBUTE = REVERSE,NOBLANK

DISPLAY = 5,1
        LABEL = REPLY.TEXT.A
        LENGTH = 80

DISPLAY = 6,1
        LABEL = REPLY.TEXT.B
        LENGTH = 80

PROMPT = 1,1
        VALUE = "Customer Number"

INPUT = .,20
```

```
                        TRAX FORMS DEFINITION (V1.0  )

Trans. Proc. Name:  SAMPLE        Device: VT62                          Page L-2
Form name:          CHCUS1        Length: 24                  02:00 PM  10-Jul-78
.....................................................................................
        LABEL = CUST.NO
        LENGTH = 6
        CLEAR = "0"
        ATTRIBUTE = REVERSE,RIGHT

!=====================================================================
!
!       This is a prompt field that tells the user what
!       function keys may be used.  The content of this field
!       may be changed by reply definitions (see below) if those
!       reply definitions change the enabled function keys.
!
!=====================================================================

PROMPT = 15,1
        LABEL = KEY.PROMPT
        LENGTH = 80
        VALUE = "Function Keys:  ",
                "ENTER to fetch customer record, ",
                "CLOSE to quit function"
        ATTRIBUTE = REVERSE

!*********************************************************
!
!       Group D Statements - Define Exchange Message
!
!*********************************************************

MESSAGE = 1
        VALUE =
                CUST.NO

!*********************************************************
!
!       Group E Statements - Define Replies
!
!*********************************************************

!=====================================================
!
!       Reply 2 is the reply for validation error messages
!
!               Write 2 80-character error messages onto screen
!               Write reply number into screen header
!
!=====================================================

REPLY = 2
        WRITE = REPLY.TEXT.A, REQUEST(1,80)
        WRITE = REPLY.TEXT.B, REQUEST(81,80)

END
```

```
                    TRAX FORMS DEFINITION (V1.0  )

Trans. Proc. Name:  SAMPLE        Device: VT62                          Page F-3
Form name:          CHCUS1        Length: 24                 02:00 PM  10-Jul-78
```

Exchange Message Layout

Length: 6 bytes

FIELD #	DISPL.	LENGTH	DESCRIPTION
1	1	6	CUST.NO

```
                    TRAX FORMS DEFINITION (V1.0  )

Trans. Proc. Name:  SAMPLE        Device: VT62                          Page F-4
Form name:          CHCUS1        Length: 24                 02:00 PM  10-Jul-78
```

REPLY # 1 Message Layout (Length = 0)

REPLY # 2 Message Layout (Length = 160)

FIELD #	DISPL.	LENGTH	DESCRIPTION
1	1	80	REPLY.TEXT.A
2	81	80	REPLY.TEXT.B

```
                         TRAX FORMS DEFINITION (V1.0  )

Trans. Proc. Name:  SAMPLE        Device: VT62                      Page S-1
Form name:          CHCUS1        Length: 24               02:00 PM  10-Jul-78
..............................................................................

                          INITIAL SCREEN PRESENTATION

         111111111122222222223333333333444444444455555555556666666666777777777778
12345678901234567890123456789012345678901234567890123456789012345678901234567890

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

           Customer Master File Subsystem - Change Customer Transaction


Customer Number    000000







Function Keys:  ENTER to fetch customer record, CLOSE to quit function

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

12345678901234567890123456789012345678901234567890123456789012345678901234567890
         111111111122222222223333333333444444444455555555556666666666777777777778


Keys enabled:    ENTER ABORT CLOSE
```

```
                        TRAX FORMS DEFINITION (V1.0  )

Trans. Proc. Name:  SAMPLE        Device: VT62                          Page 3-2
Form name:          CHCUS1        Length: 24                  02:00 PM  10-Jul-78
.....................................................................................

                    INITIAL SCREEN AFTER APPLYING REPLY #  2


         1111111111222222222233333333334444444444555555555566666666667777777778
12345678901234567890123456789012345678901234567890123456789012345678901234567890

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

           Customer Master File Subsystem - Change Customer Transaction

++=-----------------------------------------------------------------------=++
++=-----------------------------------------------------------------------=++


Customer Number     000000




Function Keys:  ENTER to fetch customer record, CLOSE to quit function

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

12345678901234567890123456789012345678901234567890123456789012345678901234567890
         1111111111222222222233333333334444444444555555555566666666667777777778


Keys enabled:     ENTER ABORT CLOSE
```

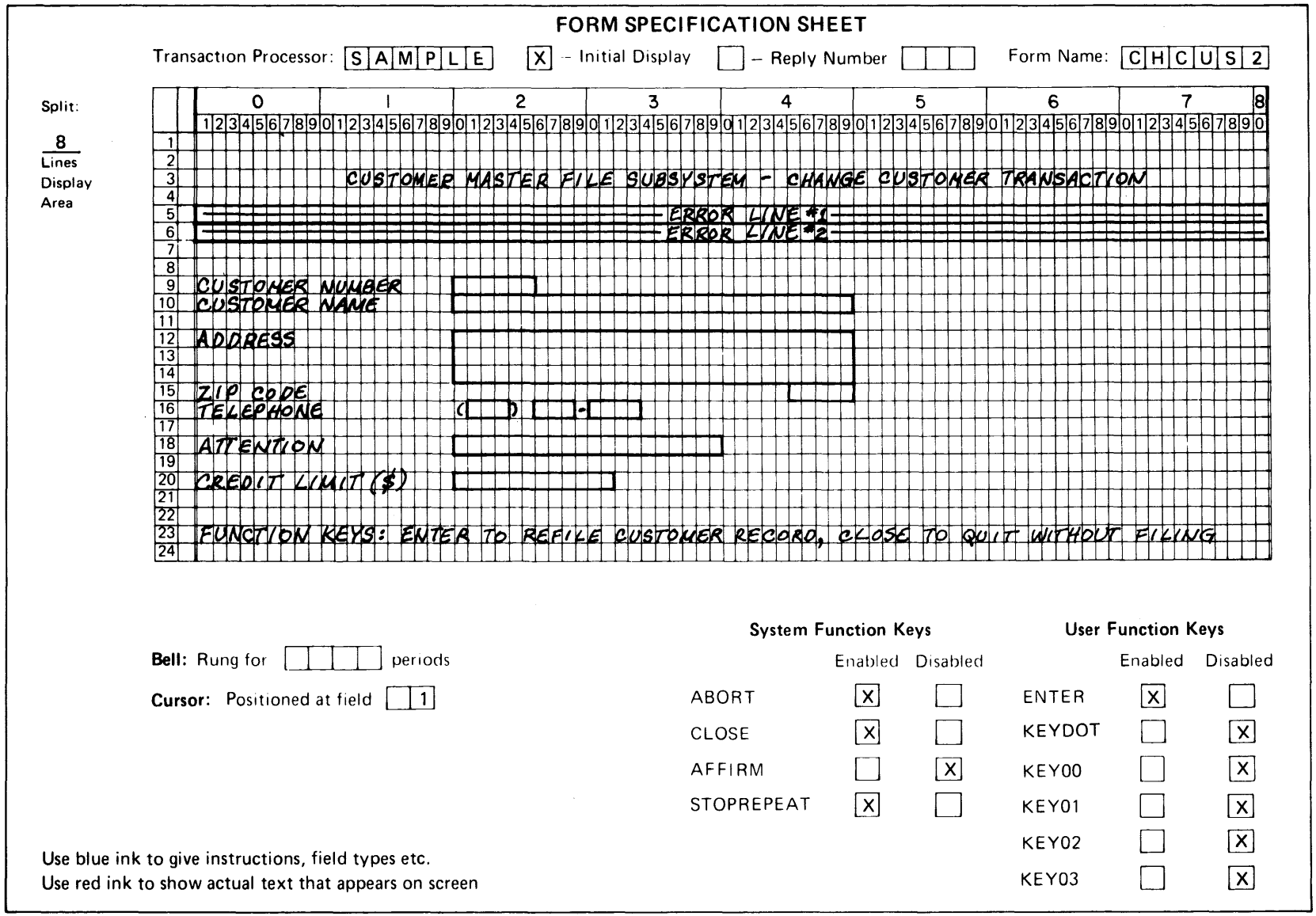


Figure 18-9 Sketch of Second Exchange Form

18.3.2 The Finished Form Definition

Here is the finished definition of the second form, as it is printed by the ATL utility program.

```
                        TMAX FORMS DEFINITION (v1.0  )

Trans. Proc. Name:  SAMPLE        Device: VT62                          Page L-1
Form name:          CHCUS2        Length: 24                  02:24 PM  10-Jul-78
.....................................................................................
!*************************************************************
!
!       Definition of form CHCUS2
!
!       The second form for the Change Customer transaction
!
!       This form displays the selected customer master file record
!       and allows the user to change the data contained in it.
!       The changed data is then sent back to the system in an
!       exchange message.
!
!*************************************************************

!*************************************************************
!
!       Group A Statements - Define General Parameters
!
!*************************************************************

DEFAULT
        ENABLE = AFFIRM
        ENABLE = CLOSE
        CLEAR = " "

FORM
        SPLIT=8                         !8 lines of display area

!*************************************************************
!
!       Group B Statements - Define Fields on Screen
!
!*************************************************************
!=============================================================
!
!       These are the application fields
!
!       The two error text fields are used to display error
!       messages contained in response messages
!
!=============================================================

DISPLAY = 3,12
        VALUE = "Customer Master File Subsystem - Change Customer Transaction"
        LENGTH = 60
        ATTRIBUTE = REVERSE,NOBLANK

DISPLAY = 5,1
        LABEL = REPLY.TEXT.A
        LENGTH = 80

DISPLAY = 6,1
        LABEL = REPLY.TEXT.B
        LENGTH = 80

PROMPT = 1,1
```

```
                    TRAX FORMS DEFINITION (V1.0  )

Trans. Proc. Name:  SAMPLE        Device: VT62                          Page L-2
Form name:          CHCUS2        Length: 24                  02:24 PM  10-Jul-78
.....................................................................................
        VALUE = "Customer Number"

INPUT = .,20
        LABEL = CUST.NO
        VALUE = REQUEST(.,6)
        ATTRIBUTE = REVERSE,NOMODIFY

PROMPT = .+1,1
        VALUE = "Customer Name"

INPUT = .,20
        LABEL = CUST.NAME
        LENGTH = 30
        VALUE = REQUEST(.,30)
        ATTRIBUTE = REQUIRED,REVERSE

PROMPT = .+2,1
        VALUE = "Address"

INPUT = .,20
        LABEL = ADDRESS.A
        LENGTH = 30
        VALUE = REQUEST(.,30)
        ATTRIBUTE = REVERSE

INPUT = .+1,20
        LABEL = ADDRESS.B
        LENGTH = 30
        VALUE = REQUEST(.,30)
        ATTRIBUTE = REVERSE

INPUT = .+1,20
        LABEL = ADDRESS.C
        LENGTH = 30
        VALUE = REQUEST(.,30)
        ATTRIBUTE = REVERSE

PROMPT = .+1,1
        VALUE = "ZIP Code"

INPUT = .,45
        LABEL = ZIP.CODE
        LENGTH = 5
        VALUE = REQUEST(.,5)
        ATTRIBUTE = REVERSE,NUMERIC,FULL

PROMPT = .+1,1
        VALUE = "Telephone"

PROMPT = .,20
        VALUE = "("

INPUT = .,.
        LABEL = TEL.AREA.CODE
        LENGTH = 3
```

```
                        TRAX FORMS DEFINITION (V1.0  )

Trans. Proc. Name:  SAMPLE        Device: VT62                          Page L-3
Form name:          CHCUS2        Length: 24                 02:24 PM  10-Jul-78
.....................................................................................
        VALUE = REQUEST(.,3)
        ATTRIBUTE = TAB,REVERSE,NUMERIC,FULL

PROMPT = .,.
        VALUE = ") "

INPUT = .,.
        LABEL = TEL.EXCHANGE
        LENGTH = 3
        VALUE = REQUEST(.,3)
        ATTRIBUTE = TAB,REVERSE,NUMERIC,FULL

PROMPT = .,.
        VALUE = "-"

INPUT = .,.
        LABEL = TEL.EXTN.NO
        LENGTH = 4
        VALUE = REQUEST(.,4)
        ATTRIBUTE = REVERSE,NUMERIC,FULL

PROMPT = .+2,1
        VALUE = "Attention"

INPUT = .,20
        LABEL = ATTENTION
        LENGTH = 20
        VALUE = REQUEST(.,20)
        ATTRIBUTE = REVERSE

PROMPT = .+2,1
        VALUE = "Credit Limit ($)"

INPUT = .,20
        LABEL = CREDIT.LIMIT
        LENGTH = 12
        VALUE = REQUEST(.,12)
        CLEAR = "0"
        ATTRIBUTE = RIGHT,REVERSE,SIGNED

!=====================================================================
!
!       This is a prompt field that tells the user what
!       function keys may be used.  The content of this field
!       may be changed by reply definitions (see below) if those
!       reply definitions change the enabled function keys.
!
!=====================================================================

PROMPT = 15,1
        LABEL = KEY.PROMPT
        LENGTH = 80
        VALUE = "Function Keys:  ",
                "ENTER to refile customer record, ",
                "CLOSE to quit without filing"
```

```
                        TRAX FORMS DEFINITION (V1.0  )

Trans. Proc. Name:  SAMPLE        Device: VT62                          Page L-4
Form name:          CMCUS2        Length: 24                  02:24 PM  10-Jul-78
.....................................................................................
        ATTRIBUTE = REVERSE

!*********************************************************
!
!       Group D Statements - Define Exchange Message
!
!*********************************************************

MESSAGE = 1
        VALUE =
                CUST.NO,
                CUST.NAME,
                ADDRESS.A,
                ADDRESS.B,
                ADDRESS.C,
                ZIP.CODE,
                TEL.AREA.CODE,TEL.EXCHANGE,TEL.EXTN.NO,
                ATTENTION,
                CREDIT.LIMIT

!*********************************************************
!
!       Group E Statements - Define Replies
!
!*********************************************************

!=========================================================
!
!       Reply 1 is affirmative reply:
!
!               Enable AFFIRM Key
!               Disable all other function keys
!               Write "TRANSACTION COMPLETE" on screen
!               Erase old function key message
!               Write new function key message
!               Write assigned Customer Number on screen
!               Write Reply Number into screen header
!
!=========================================================

REPLY = 1
        ENABLE = AFFIRM
        DISABLE = ENTER
        WRITE = REPLY.TEXT.A," *** TRANSACTION COMPLETE ***"
        WRITE = KEY.PROMPT,FILL(" ",80)
        WRITE = KEY.PROMPT,"Function Keys:  AFFIRM to proceed"


!=========================================================
!
!       Reply 2 is the reply for validation error messages
!
!               Write 2 80-character error messages onto screen
!               Write reply number into screen header
!
```

```
                    TRAX FORMS DEFINITION (V1.0  )

Trans. Proc. Name:  SAMPLE        Device: VT62                          Page L-5
Form name:          CHCUS2        Length: 24                  02:24 PM  10-Jul-78
.....................................................................................
!=============================================================

REPLY = 2
        WRITE = REPLY.TEXT.A, REQUEST(1,80)
        WRITE = REPLY.TEXT.B, REQUEST(81,80)

END
```

```
                    TRAX FORMS DEFINITION (V1.0  )

Trans. Proc. Name:  SAMPLE        Device: VT62                          Page F-1
Form name:          CHCUS2        Length: 24                  02:24 PM  10-Jul-78
.....................................................................................

General Form Parameters

Transaction processor:  SAMPLE
Form name:              CHCUS2

Length of Display Area:     8 lines
Length of Forms   Area:    15 lines
Number of error lines:      1 line @ line 24
Display width:             80 columns

   11 INPUT   Fields Declared
   11 PROMPT  Fields Declared
    3 DISPLAY Fields Declared
    0 MENU    Fields Declared

Maximum length of MENU fields:          0
Length of Exchange message:           173

Highest Defined REPLY #:   2

No Transaction SELECTion made with this form

Function KEYS enabled on:
     Initial Screen: ENTER ABORT AFFRM CLOSE
     REPLY #  1:     ABORT AFFRM CLOSE
     REPLY #  2:     ENTER ABORT AFFRM CLOSE

No KEYCAPs defined for this form
```

```
                    TRAX FORMS DEFINITION (V1.0  )

Trans. Proc. Name: SAMPLE        Device: VT62                          Page F-2
Form name:         CHCUS2        Length: 24                 02:24 PM  10-Jul-78
```

INPUT Field Declarations

Standard attributes: ALL,LEFT,NOTAB,NOFULL,NOREQUIRED
MODIFY, NORMAL, ECHO

FLD #	ROW #	COL #	LNG	CLEAR CHAR	LABEL	(ATTRIBUTES)
1	1	20	6	" "	CUST.NO	(NOMODIFY,REVERSE)
2	2	20	30	" "	CUST.NAME	(REQUIRED,REVERSE)
3	4	20	30	" "	ADDRESS.A	(REVERSE)
4	5	20	30	" "	ADDRESS.B	(REVERSE)
5	6	20	30	" "	ADDRESS.C	(REVERSE)
6	7	45	5	" "	ZIP.CODE	(NUMERIC,FULL,REVERSE)
7	8	21	3	" "	TEL.AREA.CODE	(NUMERIC,TAB,FULL,REVERSE)
8	8	26	3	" "	TEL.EXCHANGE	(NUMERIC,TAB,FULL,REVERSE)
9	8	30	4	" "	TEL.EXTN.NO	(NUMERIC,FULL,REVERSE)
10	10	20	20	" "	ATTENTION	(REVERSE)
11	12	20	12	"0"	CREDIT.LIMIT	(SIGNED,RIGHT,REVERSE)

PROMPT Field Declarations

Standard Attributes: NORMAL

FLD #	ROW #	COL #	LNG	LABEL	(ATTRIBUTES)
1	1	1	15		
2	2	1	13		
3	4	1	7		
4	7	1	8		
5	8	1	9		
6	8	20	1		
7	8	24	2		
8	8	29	1		
9	10	1	9		
10	12	1	16		
11	15	1	80	KEY.PROMPT	(REVERSE)

DISPLAY Field Declarations

Standard attributes: NORMAL, BLANK

FLD #	ROW #	COL #	LNG	LABEL	(ATTRIBUTES)
1	3	12	60		(REVERSE,NOBLANK)
2	5	1	80	REPLY.TEXT.A	

```
                    TRAX FORMS DEFINITION (V1.0  )

Trans. Proc. Name:  SAMPLE        Device: VT62                          Page F-3
Form name:          CHCUS2        Length: 24                  02:24 PM  10-Jul-78
.....................................................................................
  3    6    1   80          REPLY.TEXT.B
```

```
                    TRAX FORMS DEFINITION (V1.0  )

Trans. Proc. Name:  SAMPLE        Device: VT62                          Page F-4
Form name:          CHCUS2        Length: 24                  02:24 PM  10-Jul-78
.....................................................................................

Exchange Message Layout

Length:   173 bytes

FIELD      DISPL.    LENGTH    DESCRIPTION
  #
-------------------------------------------------------------------------------
   1A          1         6      CUST.NO
   1B          7        30      CUST.NAME
   1C         37        30      ADDRESS.A
   1D         67        30      ADDRESS.B
   1E         97        30      ADDRESS.C
   1F        127         5      ZIP.CODE
   1G        132         3      TEL.AREA.CODE
   1H        135         3      TEL.EXCHANGE
   1I        138         4      TEL.EXTN.NO
   1J        142        20      ATTENTION
   1K        162        12      CREDIT.LIMIT
```

```
                        TRAX FORMS DEFINITION (V1.0  )

Trans. Proc. Name:  SAMPLE        Device: VT62                          Page F-5
Form name:          CHCUS2        Length: 24                  02:24 PM  10-Jul-78
```

Initial Screen Request Message Layout

Length: 173 bytes

DISPL.	LENGTH	DESCRIPTION
1	6	CUST.NO
7	30	CUST.NAME
37	30	ADDRESS.A
67	30	ADDRESS.B
97	30	ADDRESS.C
127	5	ZIP.CODE
132	3	TEL.AREA.CODE
135	3	TEL.EXCHANGE
138	4	TEL.EXTN.NO
142	20	ATTENTION
162	12	CREDIT.LIMIT

```
                        TRAX FORMS DEFINITION (V1.0  )

Trans. Proc. Name:  SAMPLE        Device: VT62                          Page F-6
Form name:          CHCUS2        Length: 24                  02:24 PM  10-Jul-78
```

REPLY # 1 Message Layout (Length = 0)

REPLY # 2 Message Layout (Length = 160)

FIELD #	DISPL.	LENGTH	DESCRIPTION
1	1	80	REPLY.TEXT.A
2	81	80	REPLY.TEXT.B

```
                        TRAX FORMS DEFINITION (V1.0  )

Trans. Proc. Name:  SAMPLE          Device: VT62                          Page 8-1
Form name:          CHCUS2          Length: 24                  02:24 PM  10-Jul-78
.....................................................................................

                          INITIAL SCREEN PRESENTATION


         1111111111222222222233333333334444444444555555555566666666667777777778
123456789012345678901234567890123456789012345678901234567890123456789012345678

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

          Customer Master File Subsystem - Change Customer Transaction



Customer Number     ++-++
Customer Name       ++----------------------------++

Address             ++----------------------------++
                    ++----------------------------++
                    ++----------------------------++
ZIP Code                                    ++-++
Telephone           (+++) +++-++++

Attention           ++------------------++

Credit Limit ($)    ++----------++

Function Keys:  ENTER to refile customer record, CLOSE to quit without filing

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

123456789012345678901234567890123456789012345678901234567890123456789012345678
         1111111111222222222233333333334444444444555555555566666666667777777778


Keys enabled:    ENTER ABORT AFFRM CLOSE
```

```
                        TRAX FORMS DEFINITION (V1.0  )

Trans. Proc. Name:  SAMPLE        Device: VT62                          Page S=2
Form name:          CHCUS2        Length: 24                  02:24 PM  10-Jul-78
................................................................................

                    INITIAL SCREEN AFTER APPLYING REPLY #  1


         1111111111222222222233333333334444444444555555555566666666667777777778
12345678901234567890123456789012345678901234567890123456789012345678901234567890

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


           Customer Master File Subsystem - Change Customer Transaction

 *** TRANSACTION COMPLETE ***


Customer Number     ++=++
Customer Name       ++--------------------------++

Address             ++--------------------------++
                    ++--------------------------++
                    ++--------------------------++
ZIP Code                                  ++=++
Telephone           (+++) +++=++++

Attention           ++------------------++

Credit Limit ($)    ++----------++


Function Keys:  AFFIRM to proceed

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

12345678901234567890123456789012345678901234567890123456789012345678901234567890
         1111111111222222222233333333334444444444555555555566666666667777777778


Keys enabled:     ABORT AFFRM CLOSE
```

```
                                TRAX FORMS DEFINITION (v1.0  )

Trans. Proc. Name:  SAMPLE        Device: VT62                          Page S-3
Form name:          CHCUS2        Length: 24                   02:24 PM  10-Jul-78
................................................................................

                           INITIAL SCREEN AFTER APPLYING REPLY #  2

         111111111122222222223333333333444444444455555555556666666666777777777778
12345678901234567890123456789012345678901234567890123456789012345678901234567890

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

           Customer Master File Subsystem - Change Customer Transaction

++----------------------------------------------------------------------------++
++----------------------------------------------------------------------------++


Customer Number    ++--++
Customer Name      ++----------------------------++

Address            ++----------------------------++
                   ++----------------------------++
                   ++----------------------------++
ZIP Code                                     ++-++
Telephone          (+++) +++-++++

Attention          ++-----------------++

Credit Limit ($)   ++----------++

Function Keys:  ENTER to refile customer record, CLOSE to quit without filing

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

12345678901234567890123456789012345678901234567890123456789012345678901234567890
         111111111122222222223333333333444444444455555555556666666666777777777778


Keys enabled:    ENTER ABORT AFFRM CLOSE
```

CHAPTER 19
DESIGNING AND SPECIFYING TSTS

During the transaction design process, you tentatively divided your application processing into TSTs. Now you must add further design details to each TST, so that an application programmer can implement it.

As you proceed with this detailed design, you may find that your tentative set of TSTs is inadequate. If this happens, decide on a better set of TSTs, and correct the design to reflect the changes. Then continue with the detailed TST design procedure described in this chapter.

This chapter discusses important considerations to remember during the detailed TST design:

- TST operation
- TST programming languages and their capabilities
- TST performance
- TST design documentation
- TST coding standards and development techniques

19.1 REVIEWING TST OPERATION

Before we continue with TST design concepts, let's review: what is a TST and how does it operate?

A TST is a program associated with a TST station. When an exchange message arrives at that station, the TST processes the exchange message.

The sources of data available to the TST during this processing are shown in Figure 19-1. When it is activated, the TST is given two data structures: the exchange message and the corresponding transaction workspace. These two parameters are passed to the TST as subroutine parameters, and the TST is programmed to receive them in that form.

During processing, the TST has access to these additional sources of data:

- The TST can read or write *application data files*.
- The TST can send *report messages* to output-only terminal stations, thus producing hard-copy reports.
- The TST can deposit *mailbox messages* in a mailbox station or can retrieve mailbox messages from a mailbox station.
- The TST can issue *programmed calls* to discover certain attributes of the transaction instance it is processing.

As you can see, these sources (or destinations) can cross transaction instance boundaries. For example, a message deposited in a mailbox can be picked up by another TST processing a different transaction instance's exchange message; data read from application data files can be placed there by some other transaction instance.

However, three data structures used by the TST are only accessible during the rest of the same transaction instance:

- If the TST issues a *response message*, it affects subsequent processing of the transaction instance.
- The *exchange message*, modified by the TST, passes to subsequent stations in the processing phase of the current exchange.
- The *transaction workspace*, also modified by the TST, is passed to subsequent TSTs in the current and following exchanges of the transaction instance.

When a TST begins processing an exchange message, it has no "memory" of prior processing other than the contents of the transaction workspace. It behaves like a newly loaded and initiated program. As a result, TST designs fall into a style where each TST has a small system "perspective"; that is, it has a well delineated purpose and uses a small, formal set of inputs and outputs. Further, most TSTs have only small processing path variations in them; they apply substantially the same processing to each exchange message they process.

Remember these TST characteristics as you design your own TSTs. Your finished application will perform better if you use TST characteristics to advantage.

19.2 CHOOSING A PROGRAMMING LANGUAGE

TRAX provides two application programming languages:

- COBOL
- BASIC-PLUS-2

Both of these languages can be used to program TSTs. You should choose the language that you and your application programmers are most familiar with.

Once you have selected a programming language, study its reference manual. If you are familiar with the language, you can design TSTs that are easier to program and that perform well.

If you or your application programmers are not familiar with either of the languages, refer to the Software Product Descriptions (SPDs) and language reference manuals for both COBOL and BASIC-PLUS-2. Choose the language that fits the background and experience of your programming staff.

Although TRAX allows you to program TSTs in different languages for the same application, you should select a single language for your TSTs. Multiple languages make system documentation difficult, and application programmers using different languages have difficulty communicating.

19.3 DESIGNING FOR OPTIMUM TST PERFORMANCE

The following aspects of a TST design are particularly important:

- Making the best use of a programming language
- Making the best use of file access methods
- Minimizing access conflicts in shared files
- Avoiding bottlenecks

19.3.1 Programming Language Considerations

There are two aspects of a language that you must consider when designing TSTs:

1. *Convenience* depends on how easily a language expresses a concept or procedure.
2. *Efficiency* depends on how quickly a language executes a set of statements, and the proportion of system resources the language requires to execute those statements.

Before you design TSTs, identify those aspects of your chosen language that are convenient to use and efficient. Try to design TSTs so that your application programmers can use convenient and efficient language features. Try to avoid situations where application programmers have to use awkward or inefficient aspects of the language.

If you are unsure of a language's convenience or efficiency after studying the *Language Reference Manual* and *User's Guide*, develop some test cases and analyze the results.

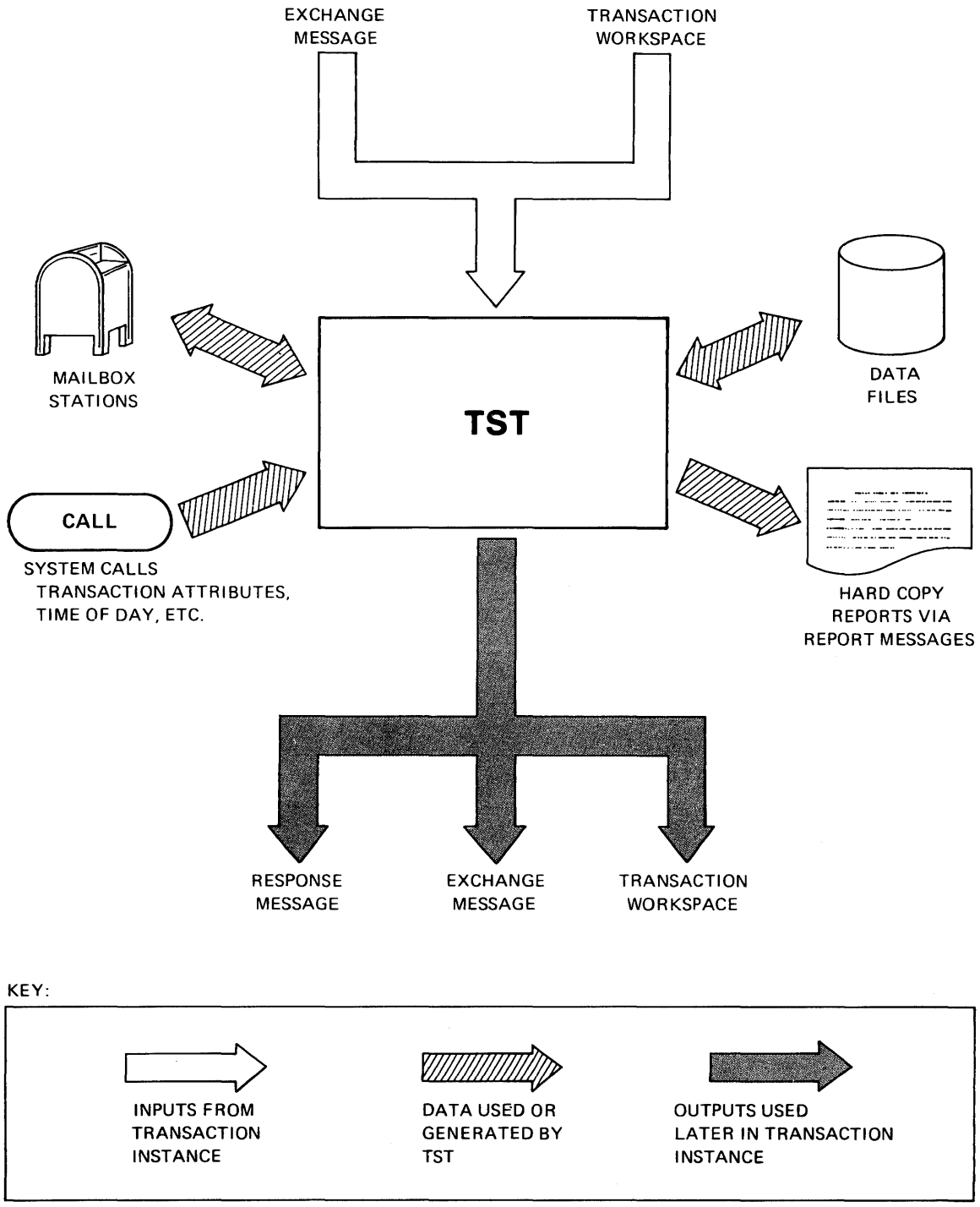


Figure 19-1 Data Available to a TST

19.3.2 File Access Considerations

The discussion in Section 19.3.1 also applies to file access techniques. Study RMS carefully. Determine which file access operations you will be using and their relative cost in system overhead.

The design of file access techniques is crucial to a commercial application. Poor performance in commercial applications is frequently caused by a file access bottleneck. Disk drives are a heavily used system resource, and you do not want to waste that resource with inefficient design techniques.

Avoid the temptation to use the simplest RMS file structures and then ask your application programmers to implement more complex file structures based upon them. You will often lose more in performance than you gain. If you need an indexed file structure, *use* an RMS indexed file structure.

An example helps to illustrate the importance of file access design. Assume your application calls for an indexed file, and a transaction must retrieve two records from this file. The records are neighbors in index sequence, except that there may be one record between them. How do you design this file access?

A straightforward technique is to use two RMS READ operations, each specifying a key for one of the two records. This works – but RMS must search twice through the index structure to find two records that are close neighbors.

A better technique is to locate the second record with one or two sequential read operations after the first record is found. This operation is more efficient than a second search through the file's index structure.

19.3.3 Minimizing Access Conflicts in Shared Files

To allow shared access to application data files, TRAX TSTs lock specific records in those files. Any TST that attempts to access a record locked by another transaction instance creates an access conflict. The TST is not given access until a TST processing the other transaction instance unlocks the record.

Avoiding repeated or lengthy access conflicts is important to application design. You must pay careful attention to the TSTs that *access* files as well as to the design of the files themselves.

19.3.3.1 The Duration of Record Locks – The duration of a record lock depends on other things besides the design of the TST. Files configured with staging or journaling prolong record locks until the end of each transaction instance, instead of releasing the records after they are unlocked or updated. This creates access conflict.

Records are locked on behalf of the transaction instance the TST is processing, rather than for the TST itself. This means that record locks can remain in force for three different periods:

1. *Within a TST.* Records can be locked by a TST and then unlocked by the same TST before it terminates execution. This record lock rarely impacts system performance, because there is minimal delay between the locking of the record and its release. Unless the TST becomes snared in an endless computational loop, nothing interrupts it and the record is unlocked in timely fashion.

   NOTE

   This does not apply if the file is staged or journaled; see the NOTE in item 3.

2. *Within an Exchange.* Records can be locked by a TST and unlocked by another TST in the same exchange. This record lock has greater impact on system performance, because a delay is always possible in the startup of the second TST. This TST could have other exchange messages queued at its station, and several seconds (or minutes) could elapse until the exchange message is processed. Meanwhile, the record remains locked.

NOTE

If the file is staged or journaled, the second TST may not unlock the record; it may remain locked until the end of the transaction instance. See the NOTE in item 3.

3. *Across Exchanges.* Records can be locked by a TST, and then unlocked by another TST in a subsequent exchange. This situation has an extreme impact on system performance, because the record remains locked during the intervening exchanges. The time required to complete an exchange depends on how fast the user completes the form and transmits it to the system; in some situations, this could take many minutes or even hours.

NOTE

Delay is often introduced where the file is defined with staging or journaling options. If a TST updates and then attempts to unlock a record in such a file, the record remains locked until the end of the transaction instance. If this involves intervening conversational phases of other exchanges, serious access conflicts can occur because of lengthy delays.

19.3.3.2 Avoiding Access Conflicts – If your design locks a record across several exchanges and this extended lock results in unacceptable access conflict, there are several things you might do.

1. Select a file definition option which allows other TSTs to *read* a locked record but not lock or update it. This option allows some transactions to display record data while another updates it. See Section 21.11.
2. Add a field to the record and use this field as an application-level record lock or record status indicator. This design uses the RMS record locking facility to protect the record only while it is read and the status field changed and rewritten. At all other times, RMS considers the record to be unlocked. Each application program must then check the record status flag before using data from the record to see if some other application program is using the record.

   Although this technique is useful, watch for these pitfalls:

   - This method relies on application programmers to inspect the record status field each time a record is read.
   - This method increases the number of file accesses, since the record status byte must be updated.
   - This method can result in records appearing to be permanently locked should an application program abort without restoring the status indicator.

3. Restructure the transaction or the file to avoid the difficulty.

   For instance, a common design technique uses an applicaton file with a control record in addition to its data records. In a customer file, this control record might contain the next available customer number. It would be consulted each time a new customer is added to the file and then incremented so that the next customer is assigned the next identification number.

   Although you may not have an access conflict problem with the customer records in this file, you would probably have a problem with the control record. This record is read, locked, and updated by each transaction instance that adds a new customer to the file. Each transaction instance must keep this record locked for the shortest possible time, because the time the record is locked has a significant effect on the throughput and response times of the application.

Consequently, your design should put the updating process (reading, locking, updating, unlocking) in one TST. This minimizes the time the control record is unavailable to other transaction instances. The updating of this control record must never be spread across two or more exchanges; and you should avoid spreading it across two TSTs in the same exchange whenever possible.

Be sure to consider the effects of staging and journaling. If the file in the previous example were staged or journaled, one of two situations would arise:

- If the control record update occurred in the last exchange, the situation would be marginally acceptable because the end of the transaction (the actual unlocking and updating) occurs soon after the reading and locking.
- If there were intervening exchanges between the point where the control record was locked and the end of the transaction instance, you would have to redesign the file or the transaction. For example, the control record might be removed from the file and placed in a separate file which is neither staged nor journaled.

19.3.4 Solutions to Possible Bottlenecks

Every application has potential bottlenecks. Bottlenecks are places in the application where much processing is done by a relatively small group of processing entities. Work backs up at bottlenecks when the application is pushed to its performance limit.

Bottlenecks have two detrimental effects on an application:

1. *Increased Response Time.* Bottlenecks may degrade response times at the user's terminal. That is, the user waits an excessively long time during the execution of a transaction.
2. *Reduced Throughput.* Bottlenecks may also reduce the total work the application can do in a given time period. That is, the application may not process work as fast as it is entered by users.

As you design around possible bottlenecks, remember which symptom you are trying to avoid – degraded response times or reduced throughput. Focusing on one of these two symptoms is important, because most techniques for avoiding bottlenecks trade one symptom for the other.

The rest of this section discusses several techniques that can be used to avoid bottlenecks:

- Allowing multiple copies of TSTs
- Adjusting TST priorities
- Designing transactions with overlapped processing
- Designing transactions with background processing

19.3.4.1 Allowing Multiple Copies of TSTs – TRAX allows you to specify the number of TST copies that can execute at the same time. You can set this number to *one*, so that arriving exchange messages process one by one. Or, you can set this parameter to a higher number and allow several exchange messages to process in parallel.

With multiple TST copies, you can solve bottlenecks stemming from TST execution times; that is, bottlenecks that arise because a TST cannot process exchange messages fast enough.

To use this technique effectively, you must be certain that:

- TST execution speed *is* the problem and not other program-delaying factors such as file access conflicts.
- Adequate resources are available to execute the copies of the TST without new conflicts such as contention for main memory or the central processor (CPU).

The number of TST copies you allow will vary by the severity of the bottleneck and the system resources available. For a severe bottleneck, you might set the parameter to a high number; but if many multiple copies cause problems in the application, you might limit the parameter to a relatively small number – say two or four.

19.3.4.2 Adjusting TST Priority – Another way of solving a TST bottleneck is to adjust the priority with which that TST is executed.

Like the multiple-copies method (Section 19.3.4.1), this method is best for correcting bottlenecks stemming from TST execution speed. Increasing the priority of a TST usually does not expedite its file accesses; the only effect of an increased priority is in contention for main memory and other TST startup resources.

Remember: when you raise the priority of one TST, you do so at the expense of another. You cannot raise the throughput of your application by adjusting TST priorities; you can only adjust the *relative* throughput of various transactions.

19.3.4.3 Designing Transactions with Overlapped Processing – Section 14.2 describes the technique of overlapping the processing phase of one exchange and the conversational phase of the text. This technique often improves the transaction response time seen by the user, but it cannot improve the throughput capability of the application as a whole.

As you design the TSTs in those transactions that use overlapped processing, remember where they appear in the transaction: are they executed before the exchange response message is sent or during the overlapped processing after the message is sent? TSTs executing during the overlapped portion of an exchange must conform to special restrictions, notably the restriction against issuing a response message.

Make sure that you understand the difference between sending a response message and terminating the TST. Once you understand the consequences of each, communicate this understanding to your application programmers.

By using the overlapped processing technique, you can design transactions where the user's conversation proceeds ahead of the processing of his last input. But this overlap is limited to one conversational cycle. If the original exchange's processing is not finished when the user's second set of input is ready, he cannot go to a third set of input. He must wait until the processing of the first input is complete and the processing of the second set begins.

19.3.4.4 Designing Transactions with Background Processing – If one of your transactions demands extended overlap or user entry of *all* data without waiting for processing to begin or terminate, you must use a background processing technique.

This technique divides a transaction into two parts:

1. An on-line transaction initiated by the user collects data and stores it temporarily.
2. A special transaction or a series of one or more support environment programs processes the stored data later.

One use of the background-processing technique spawns a separate transaction instance to process the data. To do this, you must design two transactions:

1. In the on-line transaction that converses with the user to collect data, one TST issues a system call to spawn the second transaction. The collected data can be passed in the spawned exchange message, in a mailbox message, or in a file.
2. The TSTs in the second transaction retrieve the data and then process it. These TSTs cannot converse with the user and are restricted to a "background" environment.

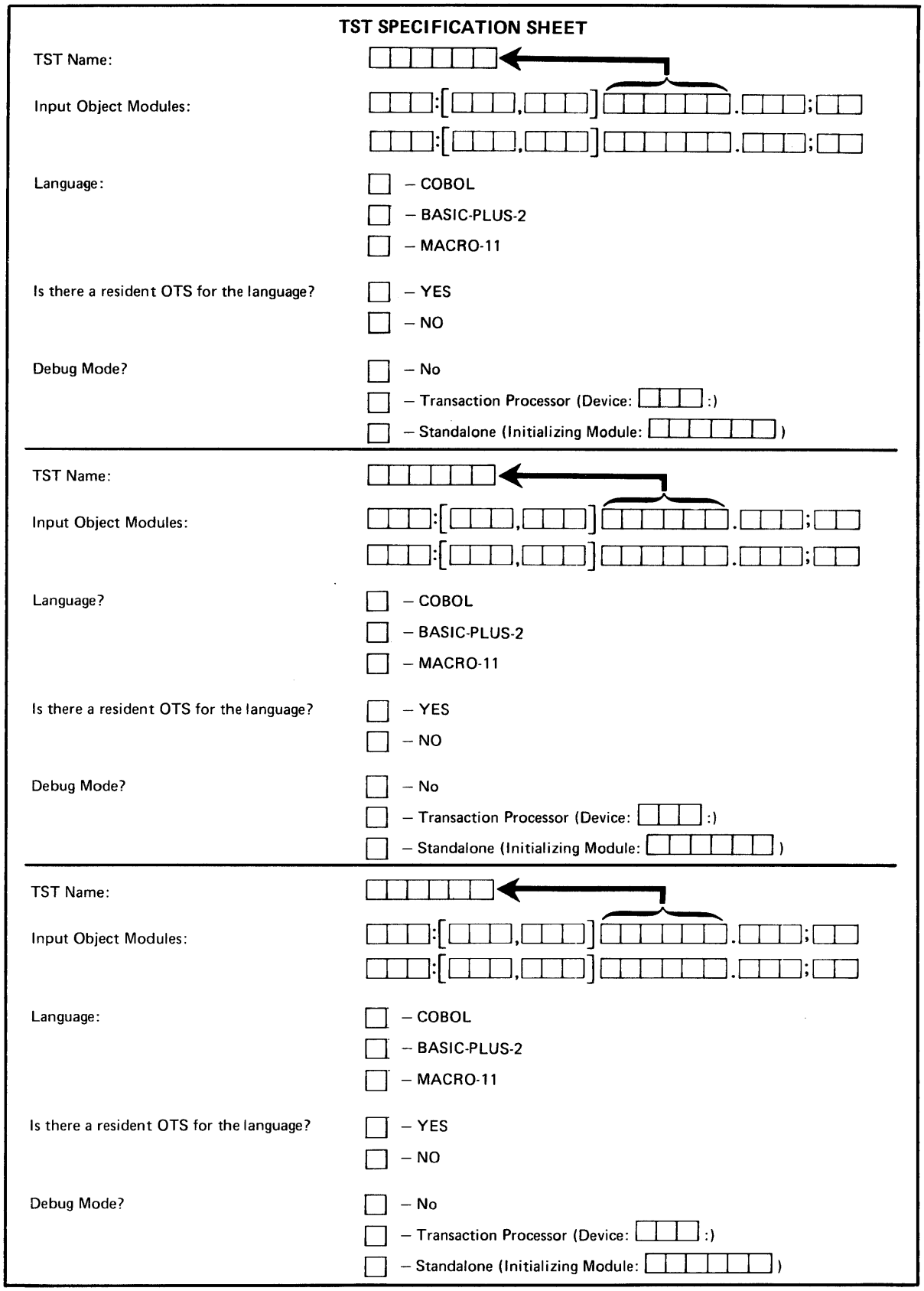


Figure 19-2 TST Specification Sheet

CHAPTER 20
TST DESIGN EXAMPLES

Three examples of TST design are presented in this chapter. Each example includes the documentation an application designer would supply to the application programmer as well as the source code that the programmer would generate. The source code is included so you can compare the design specifications with the finished TST.

The three TSTs in these examples are from the change customer transaction in the TRAX Sample Application. This transaction's forms were discussed in Chapter 18. The transaction structure diagram for this transaction is shown in Figure 18-1; it is normally included in the documentation for the programmer, although it is not reproduced in this chapter.

NOTE

The transaction structure diagram (Figure 18-1) calls for records to be locked across exchange boundaries. This is acceptable because in this application conflicts over customer records are unlikely. In a complex application, a more sophisticated approach to record sharing is necessary.

20.1 THE RDCUST TST

Together with the transaction structure diagram (Figure 18-1) the following figures comprise the documentation needed by an application programmer to write the RDCUST TST.

- RDCUST TST Specification Sheet — Figure 20-1
- Description of RDCUST TST Purpose and Processing — Figure 20-2
- Exchange Message Format — Figure 18-2
- REPLY Message Format — Figure 18-3
- PRCEED Message Format — Figure 18-4
- Customer Record Format — Figure 22-1

The finished TST is shown on pages 20-3 through 20-11.

TST SPECIFICATION SHEET

TST Name: RDCUST

Input Object Modules: SY□:[□□1,300]RDCUST.TSK;□□

□□□:[□□□,□□□]□□□□□□.□□□;□□

Language:
- [X] – COBOL
- [ ] – BASIC-PLUS-2
- [ ] – MACRO-11

Is there a resident OTS for the language?
- [X] – YES
- [ ] – NO

Debug Mode?
- [X] – No
- [ ] – Transaction Processor (Device: □□□:)
- [ ] – Standalone (Initializing Module: □□□□□□)

Figure 20-1 READ TST Specification Sheet

The RDCUST TST is the only TST in the first exchange of the change customer transaction. It receives the identification number of a customer via the exchange message and attempts to read the corresponding record from the customer file.

If the read operation is successful, the TST formats the data into punctuated display format (that is, ready for display on the terminal screen) and sends it back in a PRCEED response message.

If the read operation is not successful, the TST returns an error message in a REPLY response message. This message is sent as reply 1.

Figure 20-2 Description of RDCUST TST Purpose and Processing

```
                        TRAX FORMS DEFINITION (V1.0  )

Trans. Proc. Name:  SAMPLE       Device: VT62                          Page S=2
Form name:          CHCUS2       Length: 24                  02:24 PM  10-Jul-78
................................................................................

                    INITIAL SCREEN AFTER APPLYING REPLY #  1

         1111111111222222222233333333334444444444555555555566666666667777777778
12345678901234567890123456789012345678901234567890123456789012345678901234567890

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

           Customer Master File Subsystem - Change Customer Transaction

 *** TRANSACTION COMPLETE ***


Customer Number     ++=++
Customer Name       ++============================++

Address             ++============================++
                    ++============================++
                    ++============================++
ZIP Code                                    ++=++
Telephone           (+++) +++=++++

Attention           ++==================++

Credit Limit ($)    ++==========++


Function Keys:  AFFIRM to proceed

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

12345678901234567890123456789012345678901234567890123456789012345678901234567890
         1111111111222222222233333333334444444444555555555566666666667777777778


Keys enabled:     ABORT AFFRM CLOSE
```

```
                    TRAK FORMS DEFINITION (V1.0  )

Trans. Proc. Name:  SAMPLE        Device: VT62                          Page S-3
Form name:          CHCUS2        Length: 24                  02:24 PM  10-Jul-78
................................................................................

                  INITIAL SCREEN AFTER APPLYING REPLY #  2


         1111111111222222222233333333334444444444555555555566666666667777777778
12345678901234567890 1234567890 1234567890 1234567890 1234567890 1234567890 1234567890

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


            Customer Master File Subsystem - Change Customer Transaction

++------------------------------------------------------------------------------++
++------------------------------------------------------------------------------++


Customer Number     ++-=++
Customer Name       ++----------------------------++

Address             ++----------------------------++
                    ++----------------------------++
                    ++----------------------------++
ZIP Code                                    ++-=++
Telephone           (+++) +++-++++

Attention           ++------------------++

Credit Limit ($)    ++----------++

Function Keys:  ENTER to refile customer record, CLOSE to quit without filing

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

12345678901234567890 1234567890 1234567890 1234567890 1234567890 1234567890 1234567890
         1111111111222222222233333333334444444444555555555566666666667777777778


Keys enabled:     ENTER ABORT AFFRM CLOSE
```

CHAPTER 19
DESIGNING AND SPECIFYING TSTS

During the transaction design process, you tentatively divided your application processing into TSTs. Now you must add further design details to each TST, so that an application programmer can implement it.

As you proceed with this detailed design, you may find that your tentative set of TSTs is inadequate. If this happens, decide on a better set of TSTs, and correct the design to reflect the changes. Then continue with the detailed TST design procedure described in this chapter.

This chapter discusses important considerations to remember during the detailed TST design:

- TST operation
- TST programming languages and their capabilities
- TST performance
- TST design documentation
- TST coding standards and development techniques

19.1 REVIEWING TST OPERATION

Before we continue with TST design concepts, let's review: what is a TST and how does it operate?

A TST is a program associated with a TST station. When an exchange message arrives at that station, the TST processes the exchange message.

The sources of data available to the TST during this processing are shown in Figure 19-1. When it is activated, the TST is given two data structures: the exchange message and the corresponding transaction workspace. These two parameters are passed to the TST as subroutine parameters, and the TST is programmed to receive them in that form.

During processing, the TST has access to these additional sources of data:

- The TST can read or write *application data files*.
- The TST can send *report messages* to output-only terminal stations, thus producing hard-copy reports.
- The TST can deposit *mailbox messages* in a mailbox station or can retrieve mailbox messages from a mailbox station.
- The TST can issue *programmed calls* to discover certain attributes of the transaction instance it is processing.

As you can see, these sources (or destinations) can cross transaction instance boundaries. For example, a message deposited in a mailbox can be picked up by another TST processing a different transaction instance's exchange message; data read from application data files can be placed there by some other transaction instance.

However, three data structures used by the TST are only accessible during the rest of the same transaction instance:

- If the TST issues a *response message*, it affects subsequent processing of the transaction instance.
- The *exchange message*, modified by the TST, passes to subsequent stations in the processing phase of the current exchange.
- The *transaction workspace*, also modified by the TST, is passed to subsequent TSTs in the current and following exchanges of the transaction instance.

When a TST begins processing an exchange message, it has no "memory" of prior processing other than the contents of the transaction workspace. It behaves like a newly loaded and initiated program. As a result, TST designs fall into a style where each TST has a small system "perspective"; that is, it has a well delineated purpose and uses a small, formal set of inputs and outputs. Further, most TSTs have only small processing path variations in them; they apply substantially the same processing to each exchange message they process.

Remember these TST characteristics as you design your own TSTs. Your finished application will perform better if you use TST characteristics to advantage.

19.2 CHOOSING A PROGRAMMING LANGUAGE

TRAX provides two application programming languages:

- COBOL
- BASIC-PLUS-2

Both of these languages can be used to program TSTs. You should choose the language that you and your application programmers are most familiar with.

Once you have selected a programming language, study its reference manual. If you are familiar with the language, you can design TSTs that are easier to program and that perform well.

If you or your application programmers are not familiar with either of the languages, refer to the Software Product Descriptions (SPDs) and language reference manuals for both COBOL and BASIC-PLUS-2. Choose the language that fits the background and experience of your programming staff.

Although TRAX allows you to program TSTs in different languages for the same application, you should select a single language for your TSTs. Multiple languages make system documentation difficult, and application programmers using different languages have difficulty communicating.

19.3 DESIGNING FOR OPTIMUM TST PERFORMANCE

The following aspects of a TST design are particularly important:

- Making the best use of a programming language
- Making the best use of file access methods
- Minimizing access conflicts in shared files
- Avoiding bottlenecks

19.3.1 Programming Language Considerations

There are two aspects of a language that you must consider when designing TSTs:

1. *Convenience* depends on how easily a language expresses a concept or procedure.
2. *Efficiency* depends on how quickly a language executes a set of statements, and the proportion of system resources the language requires to execute those statements.

Before you design TSTs, identify those aspects of your chosen language that are convenient to use and efficient. Try to design TSTs so that your application programmers can use convenient and efficient language features. Try to avoid situations where application programmers have to use awkward or inefficient aspects of the language.

If you are unsure of a language's convenience or efficiency after studying the *Language Reference Manual* and *User's Guide*, develop some test cases and analyze the results.

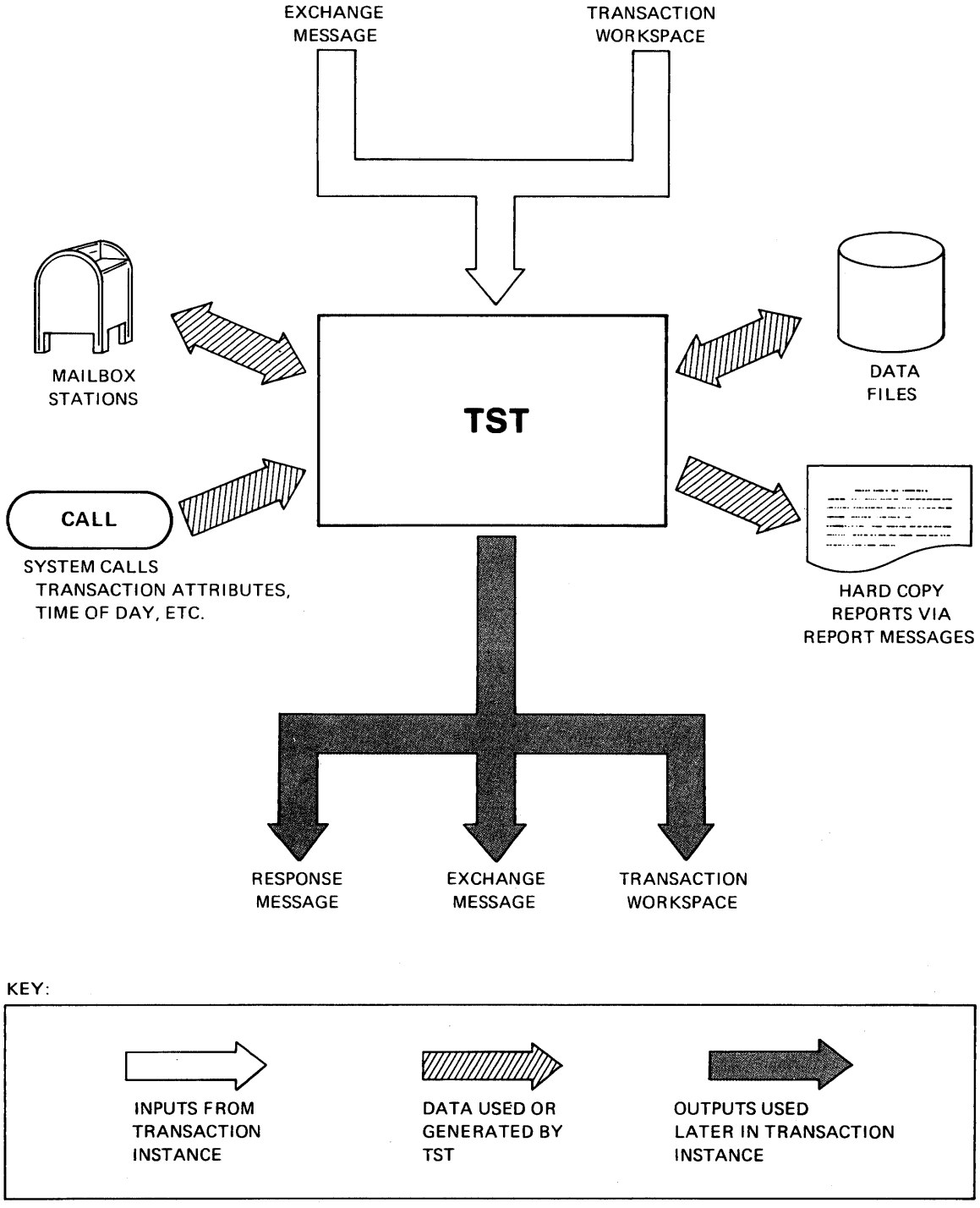


Figure 19-1 Data Available to a TST

19.3.2 File Access Considerations

The discussion in Section 19.3.1 also applies to file access techniques. Study RMS carefully. Determine which file access operations you will be using and their relative cost in system overhead.

The design of file access techniques is crucial to a commercial application. Poor performance in commercial applications is frequently caused by a file access bottleneck. Disk drives are a heavily used system resource, and you do not want to waste that resource with inefficient design techniques.

Avoid the temptation to use the simplest RMS file structures and then ask your application programmers to implement more complex file structures based upon them. You will often lose more in performance than you gain. If you need an indexed file structure, *use* an RMS indexed file structure.

An example helps to illustrate the importance of file access design. Assume your application calls for an indexed file, and a transaction must retrieve two records from this file. The records are neighbors in index sequence, except that there may be one record between them. How do you design this file access?

A straightforward technique is to use two RMS READ operations, each specifying a key for one of the two records. This works – but RMS must search twice through the index structure to find two records that are close neighbors.

A better technique is to locate the second record with one or two sequential read operations after the first record is found. This operation is more efficient than a second search through the file's index structure.

19.3.3 Minimizing Access Conflicts in Shared Files

To allow shared access to application data files, TRAX TSTs lock specific records in those files. Any TST that attempts to access a record locked by another transaction instance creates an access conflict. The TST is not given access until a TST processing the other transaction instance unlocks the record.

Avoiding repeated or lengthy access conflicts is important to application design. You must pay careful attention to the TSTs that *access* files as well as to the design of the files themselves.

19.3.3.1 The Duration of Record Locks – The duration of a record lock depends on other things besides the design of the TST. Files configured with staging or journaling prolong record locks until the end of each transaction instance, instead of releasing the records after they are unlocked or updated. This creates access conflict.

Records are locked on behalf of the transaction instance the TST is processing, rather than for the TST itself. This means that record locks can remain in force for three different periods:

1. *Within a TST.* Records can be locked by a TST and then unlocked by the same TST before it terminates execution. This record lock rarely impacts system performance, because there is minimal delay between the locking of the record and its release. Unless the TST becomes snared in an endless computational loop, nothing interrupts it and the record is unlocked in timely fashion.

   NOTE

   This does not apply if the file is staged or journaled; see the NOTE in item 3.

2. *Within an Exchange.* Records can be locked by a TST and unlocked by another TST in the same exchange. This record lock has greater impact on system performance, because a delay is always possible in the startup of the second TST. This TST could have other exchange messages queued at its station, and several seconds (or minutes) could elapse until the exchange message is processed. Meanwhile, the record remains locked.

NOTE

If the file is staged or journaled, the second TST may not unlock the record; it may remain locked until the end of the transaction instance. See the NOTE in item 3.

3. *Across Exchanges.* Records can be locked by a TST, and then unlocked by another TST in a subsequent exchange. This situation has an extreme impact on system performance, because the record remains locked during the intervening exchanges. The time required to complete an exchange depends on how fast the user completes the form and transmits it to the system; in some situations, this could take many minutes or even hours.

NOTE

Delay is often introduced where the file is defined with staging or journaling options. If a TST updates and then attempts to unlock a record in such a file, the record remains locked until the end of the transaction instance. If this involves intervening conversational phases of other exchanges, serious access conflicts can occur because of lengthy delays.

19.3.3.2 Avoiding Access Conflicts – If your design locks a record across several exchanges and this extended lock results in unacceptable access conflict, there are several things you might do.

1. Select a file definition option which allows other TSTs to *read* a locked record but not lock or update it. This option allows some transactions to display record data while another updates it. See Section 21.11.
2. Add a field to the record and use this field as an application-level record lock or record status indicator. This design uses the RMS record locking facility to protect the record only while it is read and the status field changed and rewritten. At all other times, RMS considers the record to be unlocked. Each application program must then check the record status flag before using data from the record to see if some other application program is using the record.

   Although this technique is useful, watch for these pitfalls:

   - This method relies on application programmers to inspect the record status field each time a record is read.
   - This method increases the number of file accesses, since the record status byte must be updated.
   - This method can result in records appearing to be permanently locked should an application program abort without restoring the status indicator.

3. Restructure the transaction or the file to avoid the difficulty.

   For instance, a common design technique uses an applicaton file with a control record in addition to its data records. In a customer file, this control record might contain the next available customer number. It would be consulted each time a new customer is added to the file and then incremented so that the next customer is assigned the next identification number.

   Although you may not have an access conflict problem with the customer records in this file, you would probably have a problem with the control record. This record is read, locked, and updated by each transaction instance that adds a new customer to the file. Each transaction instance must keep this record locked for the shortest possible time, because the time the record is locked has a significant effect on the throughput and response times of the application.

Consequently, your design should put the updating process (reading, locking, updating, unlocking) in one TST. This minimizes the time the control record is unavailable to other transaction instances. The updating of this control record must never be spread across two or more exchanges; and you should avoid spreading it across two TSTs in the same exchange whenever possible.

Be sure to consider the effects of staging and journaling. If the file in the previous example were staged or journaled, one of two situations would arise:

- If the control record update occurred in the last exchange, the situation would be marginally acceptable because the end of the transaction (the actual unlocking and updating) occurs soon after the reading and locking.
- If there were intervening exchanges between the point where the control record was locked and the end of the transaction instance, you would have to redesign the file or the transaction. For example, the control record might be removed from the file and placed in a separate file which is neither staged nor journaled.

19.3.4 Solutions to Possible Bottlenecks

Every application has potential bottlenecks. Bottlenecks are places in the application where much processing is done by a relatively small group of processing entities. Work backs up at bottlenecks when the application is pushed to its performance limit.

Bottlenecks have two detrimental effects on an application:

1. *Increased Response Time.* Bottlenecks may degrade response times at the user's terminal. That is, the user waits an excessively long time during the execution of a transaction.
2. *Reduced Throughput.* Bottlenecks may also reduce the total work the application can do in a given time period. That is, the application may not process work as fast as it is entered by users.

As you design around possible bottlenecks, remember which symptom you are trying to avoid -- degraded response times or reduced throughput. Focusing on one of these two symptoms is important, because most techniques for avoiding bottlenecks trade one symptom for the other.

The rest of this section discusses several techniques that can be used to avoid bottlenecks:

- Allowing multiple copies of TSTs
- Adjusting TST priorities
- Designing transactions with overlapped processing
- Designing transactions with background processing

19.3.4.1 Allowing Multiple Copies of TSTs – TRAX allows you to specify the number of TST copies that can execute at the same time. You can set this number to *one*, so that arriving exchange messages process one by one. Or, you can set this parameter to a higher number and allow several exchange messages to process in parallel.

With multiple TST copies, you can solve bottlenecks stemming from TST execution times; that is, bottlenecks that arise because a TST cannot process exchange messages fast enough.

To use this technique effectively, you must be certain that:

- TST execution speed *is* the problem and not other program-delaying factors such as file access conflicts.
- Adequate resources are available to execute the copies of the TST without new conflicts such as contention for main memory or the central processor (CPU).

The number of TST copies you allow will vary by the severity of the bottleneck and the system resources available. For a severe bottleneck, you might set the parameter to a high number; but if many multiple copies cause problems in the application, you might limit the parameter to a relatively small number – say two or four.

19.3.4.2 Adjusting TST Priority – Another way of solving a TST bottleneck is to adjust the priority with which that TST is executed.

Like the multiple-copies method (Section 19.3.4.1), this method is best for correcting bottlenecks stemming from TST execution speed. Increasing the priority of a TST usually does not expedite its file accesses; the only effect of an increased priority is in contention for main memory and other TST startup resources.

Remember: when you raise the priority of one TST, you do so at the expense of another. You cannot raise the throughput of your application by adjusting TST priorities; you can only adjust the *relative* throughput of various transactions.

19.3.4.3 Designing Transactions with Overlapped Processing – Section 14.2 describes the technique of overlapping the processing phase of one exchange and the conversational phase of the text. This technique often improves the transaction response time seen by the user, but it cannot improve the throughput capability of the application as a whole.

As you design the TSTs in those transactions that use overlapped processing, remember where they appear in the transaction: are they executed before the exchange response message is sent or during the overlapped processing after the message is sent? TSTs executing during the overlapped portion of an exchange must conform to special restrictions, notably the restriction against issuing a response message.

Make sure that you understand the difference between sending a response message and terminating the TST. Once you understand the consequences of each, communicate this understanding to your application programmers.

By using the overlapped processing technique, you can design transactions where the user's conversation proceeds ahead of the processing of his last input. But this overlap is limited to one conversational cycle. If the original exchange's processing is not finished when the user's second set of input is ready, he cannot go to a third set of input. He must wait until the processing of the first input is complete and the processing of the second set begins.

19.3.4.4 Designing Transactions with Background Processing – If one of your transactions demands extended overlap or user entry of *all* data without waiting for processing to begin or terminate, you must use a background processing technique.

This technique divides a transaction into two parts:

1. An on-line transaction initiated by the user collects data and stores it temporarily.
2. A special transaction or a series of one or more support environment programs processes the stored data later.

One use of the background-processing technique spawns a separate transaction instance to process the data. To do this, you must design two transactions:

1. In the on-line transaction that converses with the user to collect data, one TST issues a system call to spawn the second transaction. The collected data can be passed in the spawned exchange message, in a mailbox message, or in a file.
2. The TSTs in the second transaction retrieve the data and then process it. These TSTs cannot converse with the user and are restricted to a "background" environment.

This technique for background processing affects system performance like the simpler overlap technique (Section 19.3.4.3): the apparent response times are improved for the user, but the overall throughput of the application cannot be improved.

Remember, too, that a design that allows the user to enter data faster than it can be processed seriously affects application performance during busy periods. Be sure that this artificially high processing load does not exceed your application's processing capacity and create performance problems greater than those you are trying to avoid.

19.4 DOCUMENTING THE TST DESIGN

To document a TST design, write a specification for the processing flow. Because of the modular nature of TSTs and the corresponding "perspective" (Section 19.1), you can describe processing for many TSTs in one or two paragraphs. Such a specification, together with the relevant transaction structure diagram, is all that is normally needed.

Some detailed documentation of TST processing may be required. A customer invoicing transaction, for example, may require precise accounting procedures and methods for interest calculation. In these situations, you will naturally supply formulas or detailed flowcharts of the relevant program segments. Remember: application programmers are more comfortable with concepts described in *algorithms*, rather than in the formal language of mathematics.

For instance, if a TST must compute an average (or mean), you should not write documentation for the application programmer that looks like this:

$$M = \frac{\sum_{i=1}^{n} X_n}{n}$$

where, X_n is the nth list element and
M is the mean of the list $X_1, \ldots, X_n$

It is better to communicate concepts like these in algorithms:

> "Determine the mean (or average) of the list of account balances by computing the sum of the balances and then dividing the sum by the number of balances in the list."

In general, you should not draw more than one uncrowded page of flowchart to describe any TST. (There are exceptions, of course, for detailed computational procedures.) Any TST requiring more than one page of flowchart should normally be divided into two or more TSTs, and the design of each transaction using that TST should be modified to reflect the change.

Besides the processing flow documentation, you should give your application programmers documentation for relevant transaction data structures. This includes exchange messages, response messages, other messages, and the transaction workspace. Also include record layouts for each data file record accessed.

Be sure the application programmer knows what to do with these data structure formats. Be sure he knows when to use working storage within his program for the storage of temporary results and when to place those results in the transaction workspace instead. And be sure he knows which structures are initialized to known values and which have unspecified contents when his TST begins execution.

Be careful to document situations where the TST alters data in the exchange message or transaction workspace for processing at other stations. These data modifications will be important for proper operation of the transaction and must be implemented to agree with your design.

To minimize confusion, you should avoid changes to exchange messages except where necessary. Use the transaction workspace for communication between processing stations, not the exchange message. Then your application programmers will access the exchange message in a read-only manner, and some application errors might be avoided.

Finally, include a TST specification sheet as a cover sheet for each TST documentation package (Figure 19-2). This sheet shows the TST name, the associated TST station, and other parameters such as the maximum number of executing copies and the execution priority. The application programmer uses this information when the TST and its corresponding station are installed in the transaction processor.

19.5 CODING STANDARDS AND DEVELOPMENT TECHNIQUES

As the application designer, with responsibility for the successful implementation of an application, you should do more than generate functional designs for each of the application TSTs. Take additional steps to ensure that the TSTs can be implemented quickly, easily, and accurately. You can do this by suggesting coding standards and development techniques in your TST design documentation.

Coding standards are rules of style for the programmer to use when writing programs. There are nearly infinite ways to write a program. But only a few result in programs that are clear, easy to understand, and operationally correct.

Clear expression is of major importance in a commercial application program. Business requirements change frequently, often faster than a data processing system can be changed. An obscure program takes more time and effort to modify; and there is less chance that the modification will work properly.

Coding standards for your application programmers might include the following topics:

- Conventions for naming data items
- Guidelines for commenting programs
- Guidelines for segmenting programs into elements like sections, paragraphs, and subroutines
- Language features that should not be used
- Guidelines for formatting program source text, like indentation and use of tabs
- Guidelines for the treatment of complex conditional statements such as nested IF-THEN-ELSE statements
- Hints for obtaining the maximum execution efficiency in the selected language

Carefully devised coding standards go hand in hand with optimum development techniques. The on-line debugging features of BASIC-PLUS-2, for example, are most productive when your programmers have coded their programs in a clear and straightforward style.

Development techniques you may want to suggest include:

- Developing program segments or subroutines which can be included in the source code of several TSTs. (In COBOL, you can do this with the COPY statement or by using the text editor to include the selected source text. In BASIC-PLUS-2, you can use the APPEND command or the text editor.) This technique is particularly useful for transaction data structures (exchange message, response message, and transaction workspace formats).
- Creating a "skeleton" program containing program overhead items. This is often useful when there are several TSTs superficially similar but with somewhat different processing procedures. This includes major headings for program sections, source code normally the same in all programs, and perhaps the definitions of working storage variables and record formats for files. Each time a programmer begins work on a new TST, he can take a copy of this "skeleton" and then fill in the segments which are unique to his program.

Used appropriately, these techniques can significantly improve programmer productivity. They are most efficient, of course, when they are used with sensible coding standards.

TST SPECIFICATION SHEET

TST Name: ☐☐☐☐☐☐ ←

Input Object Modules: ☐☐☐:[☐☐☐,☐☐☐]☐☐☐☐☐☐.☐☐☐;☐☐

☐☐☐:[☐☐☐,☐☐☐]☐☐☐☐☐☐.☐☐☐;☐☐

Language:
- ☐ – COBOL
- ☐ – BASIC-PLUS-2
- ☐ – MACRO-11

Is there a resident OTS for the language?
- ☐ – YES
- ☐ – NO

Debug Mode?
- ☐ – No
- ☐ – Transaction Processor (Device: ☐☐☐ :)
- ☐ – Standalone (Initializing Module: ☐☐☐☐☐☐)

TST Name: ☐☐☐☐☐☐ ←

Input Object Modules: ☐☐☐:[☐☐☐,☐☐☐]☐☐☐☐☐☐.☐☐☐;☐☐

☐☐☐:[☐☐☐,☐☐☐]☐☐☐☐☐☐.☐☐☐;☐☐

Language?
- ☐ – COBOL
- ☐ – BASIC-PLUS-2
- ☐ – MACRO-11

Is there a resident OTS for the language?
- ☐ – YES
- ☐ – NO

Debug Mode?
- ☐ – No
- ☐ – Transaction Processor (Device: ☐☐☐ :)
- ☐ – Standalone (Initializing Module: ☐☐☐☐☐☐)

TST Name: ☐☐☐☐☐☐ ←

Input Object Modules: ☐☐☐:[☐☐☐,☐☐☐]☐☐☐☐☐☐.☐☐☐;☐☐

☐☐☐:[☐☐☐,☐☐☐]☐☐☐☐☐☐.☐☐☐;☐☐

Language:
- ☐ – COBOL
- ☐ – BASIC-PLUS-2
- ☐ – MACRO-11

Is there a resident OTS for the language?
- ☐ – YES
- ☐ – NO

Debug Mode?
- ☐ – No
- ☐ – Transaction Processor (Device: ☐☐☐ :)
- ☐ – Standalone (Initializing Module: ☐☐☐☐☐☐)

Figure 19-2 TST Specification Sheet

CHAPTER 20
TST DESIGN EXAMPLES

Three examples of TST design are presented in this chapter. Each example includes the documentation an application designer would supply to the application programmer as well as the source code that the programmer would generate. The source code is included so you can compare the design specifications with the finished TST.

The three TSTs in these examples are from the change customer transaction in the TRAX Sample Application. This transaction's forms were discussed in Chapter 18. The transaction structure diagram for this transaction is shown in Figure 18-1 ; it is normally included in the documentation for the programmer, although it is not reproduced in this chapter.

NOTE

The transaction structure diagram (Figure 18-1) calls for records to be locked across exchange boundaries. This is acceptable because in this application conflicts over customer records are unlikely. In a complex application, a more sophisticated approach to record sharing is necessary.

20.1 THE RDCUST TST

Together with the transaction structure diagram (Figure 18-1) the following figures comprise the documentation needed by an application programmer to write the RDCUST TST.

- RDCUST TST Specification Sheet — Figure 20-1
- Description of RDCUST TST Purpose and Processing — Figure 20-2
- Exchange Message Format — Figure 18-2
- REPLY Message Format — Figure 18-3
- PRCEED Message Format — Figure 18-4
- Customer Record Format — Figure 22-1

The finished TST is shown on pages 20-3 through 20-11.

TST SPECIFICATION SHEET

TST Name: RDCUST

Input Object Modules: SY:[1,300]RDCUST.TSK;

:[,]..;

Language:
- [X] – COBOL
- [ ] – BASIC-PLUS-2
- [ ] – MACRO-11

Is there a resident OTS for the language?
- [X] – YES
- [ ] – NO

Debug Mode?
- [X] – No
- [ ] – Transaction Processor (Device: :)
- [ ] – Standalone (Initializing Module:)

Figure 20-1 READ TST Specification Sheet

The RDCUST TST is the only TST in the first exchange of the change customer transaction. It receives the identification number of a customer via the exchange message and attempts to read the corresponding record from the customer file.

If the read operation is successful, the TST formats the data into punctuated display format (that is, ready for display on the terminal screen) and sends it back in a PRCEED response message.

If the read operation is not successful, the TST returns an error message in a REPLY response message. This message is sent as reply 1.

Figure 20-2 Description of RDCUST TST Purpose and Processing

```
 COBOL  3.25   SRC:RDCUST.CBL;7          13-JUL-78  12:10:40 PAGE 001

CMD:RDCUST,RDCUST=RDCUST/TST
IDENT: 194121


  00001         IDENTIFICATION DIVISION.
  00002        ****************************************************************
  00003        *                                                              *
  00004        *                  T S T     D E S C R I P T I O N             *
  00005        *                                                              *
  00006        ****************************************************************
  00007        *
  00008        *  TST NAME:  RDCUST
  00009        *
  00010        *  TRANSACTION:  CHGCUS - CHANGE CUSTOMER RECORD
  00011        *
  00012        *  FUNCTION:     THIS TST ACCEPTS A CUSTOMER NUMBER SUPPLIED
  00013        *                BY THE TERMINAL OPERATOR AND USES IT TO FETCH
  00014        *                DESIRED RECORD FROM THE CUSTOMER MASTER FILE.
  00015        *                WHEN IT FINDS IT, THE DATA IS FORMATTED AND SENT
  00016        *                TO THE SECOND EXCHANGE BY USE OF A PROCEED
  00017        *                MESSAGE.
  00018        *
  00019        *  FILES:        CUSTOMER MASTER FILE "CUSTOM.DAT"
  00020        *
  00021        *  INPUT FORM:   CHCUS1 - SUPPLY CUSTOMER NUMBER TO FETCH RECORD
  00022        *
  00023        *  OUTPUT FORM:  CHCUS2 - DISPLAY AND EDIT CUSTOMER DATA.
  00024        *
  00025        *
  00026        *
  00027        ****************************************************************
  00028


 COBOL  3.05   SRC:RDCUST.CBL;7          13-JUL-78  12:10:40 PAGE 002

  00029        /***************************************************************
  00030        *                                                              *
  00031        *                          T H E                               *
  00032        *                                                              *
  00033        *   I D E N T I F I C A T I O N        D I V I S I O N         *
  00034        *                                                              *
  00035        *                                                              *
  00036        ****************************************************************
  00037
  00038         PROGRAM-ID. TSTEP.
  00039         DATE-COMPILED. TODAY.
  00040
```

```
COBOL  3,05   SRC:RDCUST.CBL;7          13-JUL-78  12:10:40 PAGE 003

 00041        /*****************************************************************
 00042        *                                                               *
 00043        *                              T H E                            *
 00044        *                                                               *
 00045        *     E N V I R O N M E N T        D I V I S I O N              *
 00046        *                                                               *
 00047        *                                                               *
 00048        *****************************************************************
 00049
 00050             13-JUL-78 .
 00051         ENVIRONMENT DIVISION.
 00052         CONFIGURATION SECTION.
 00053             SOURCE-COMPUTER.  PDP-11.
 00054             OBJECT-COMPUTER.  PDP-11.
 00055
 00056        *****************************************************************
 00057        *                                                               *
 00058        *           I N P U T - O U T P U T    S E C T I O N            *
 00059        *                                                               *
 00060        *****************************************************************
 00061
 00062         INPUT-OUTPUT SECTION.
 00063
 00064         FILE-CONTROL.
 00065
 00066             SELECT CUSTOM ASSIGN TO "CUSTOM.DAT"
 00067                    ORGANIZATION IS INDEXED
 00068                    ACCESS MODE IS DYNAMIC
 00069                    RECORD KEY IS CUSTOMER-NUMBER
 00070                    ALTERNATE RECORD KEY IS CUSTOMER-NAME WITH DUPLICATES
 00071                    FILE STATUS IS CUSTOMER-FILE-STATUS.
 00072
```

```
COBOL  3.V5   SRC:MDCUST.CBL;7          13-JUL-78  12:10:40 PAGE 004

00073       /*****************************************************************
00074       *                                                               *
00075       *                            T H E                              *
00076       *                                                               *
00077       *                D A T A      D I V I S I O N                   *
00078       *                                                               *
00079       *                                                               *
00080       *****************************************************************
00081
00082
00083        DATA DIVISION.
00084
00085       *****************************************************************
00086       *                                                             *
00087       *            F I L E        S E C T I O N                     *
00088       *                                                                 *
00089       *****************************************************************
00090
00091        FILE SECTION.
00092
00093        FD  CUSTOM
00094            LABEL RECORDS ARE STANDARD
00095            VALUE OF ID IS CUSTOM-CHANNEL-NUMBER
00096            DATA RECORD IS CUSTOMER-FILE-RECORD.
00097
00098        01  CUSTOMER-FILE-RECORD.
00099
00100
00101            03  CUSTOMER-NUMBER                     PIC X(6).
00102            03  CUSTOMER-NAME                       PIC X(30).
00103            03  ADDRESS-LINE-1                      PIC X(30).
00104            03  ADDRESS-LINE-2                      PIC X(30).
00105            03  ADDRESS-LINE-3                      PIC X(30).
00106            03  ADDRESS-ZIP-CODE                    PIC 9(5).
00107            03  TELEPHONE-NUMBER                    PIC 9(10).
00108            03  ATTENTION-LINE                      PIC X(20).
00109            03  CREDIT-LIMIT-AMOUNT                 PIC 9(10)V99.
00110            03  CURRENT-BALANCE                     PIC 9(10)V99.
00111            03  PURCHASES-YTD                       PIC 9(10)V99.
00112            03  NEXT-ORDER-SEQUENCE-NUMBER          PIC 9(4).
00113            03  NEXT-PAYMENT-SEQUENCE-NUMBER        PIC 9(4).
00114
00115
00116       *****************************************************************
00117       *                                                               *
00118       *   W O R K I N G - S T O R A G E      S E C T I O N            *
00119       *                                                               *
00120       *****************************************************************
00121
00122        WORKING-STORAGE SECTION.
00123
00124       *****************************************************************
00125       *              F I L E                C H A N N E L S           *
00126       *****************************************************************
00127
00128        01  CUSTOM-CHANNEL-NUMBER              PIC X(11)
00129                VALUE IS "CUSTOM/CH:3".
00130
```

```
COBOL  3.05   SRC:RDCUST.CBL;7          13-JUL-78  12:10:40 PAGE 005

 00131        ****************************************************************
 00132        *         F I L E    S T A T U S    N A M E S                  *
 00133        ****************************************************************
 00134         01  FILE-STATUS-WORD                          PIC XX.
 00135
 00136         01  CUSTOMER-FILE-STATUS                      PIC XX.
 00137
 00138        ****************************************************************
 00139        *       M E S S A G E                A R G U M E N T S       *
 00140        ****************************************************************
 00141         01  BUFFER-SIZE                               PIC 9999 COMP.
 00142
 00143         01  STATUS-WORDS.
 00144
 00145             03  STATUS-WORD-1                         PIC S9(4) COMP.
 00146             03  STATUS-WORD-2                         PIC S9(4) COMP.
 00147
 00148         01  REPLY-NUMBER                              PIC S9(4) COMP.
 00149
 00150         01  PROCEED-MESSAGE-BUFFER.
 00151
 00152             02  RM-CUSTOMER-FILE-RECORD.
 00153
 00154                     03  RM-CUSTOMER-NUMBER                    PIC X(6).
 00155                     03  RM-CUSTOMER-NAME                      PIC X(30).
 00156                     03  RM-ADDRESS-LINE-1                     PIC X(30).
 00157                     03  RM-ADDRESS-LINE-2                     PIC X(30).
 00158                     03  RM-ADDRESS-LINE-3                     PIC X(30).
 00159                     03  RM-ADDRESS-ZIP-CODE                   PIC 9(5).
 00160                     03  RM-TELEPHONE-NUMBER                   PIC 9(10).
 00161                     03  RM-ATTENTION-LINE                     PIC X(20).
 00162                     03  RM-CREDIT-LIMIT-AMOUNT          PIC Z,ZZZ,ZZZ.99.
 00163
 00164         01  REPLY-MESSAGE-BUFFER.
 00165             03  REPLY-MESSAGE-TEXT                    PIC X(80).
 00166             03  REPLY-FILLER                          PIC X(18)
 00167                     VALUE IS "File Status Word: ".
 00168             03  RMB-FSW                               PIC X(2).
 00169             03  RMB-FILE-NAME                         PIC X(60).
 00170
```

```
COBOL  3.05   SRC:MDCUST.CBL;7          13-JUL-78  12:10:40 PAGE 006

00171       /****************************************************************
00172       *                                                              *
00173       *                    L I N K A G E   S E C T I O N             *
00174       *                                                              *
00175       ****************************************************************
00176
00177        LINKAGE SECTION.
00178
00179       *****                      M E S S A G E                      *****
00180
00181        01  EXCHANGE-MESSAGE.
00182
00183            02  EM-INPUT-FORM-CHCUS1.
00184
00185                    03  EM-CUSTOMER-NUMBER          PIC X(6).
00186
00187
00188       *****                     W O R K S P A C E                  *****
00189
00190        01  TRANSACTION-WORKSPACE.
00191
00192
00193            02  WS-CUSTOMER-FILE-RECORD.
00194
00195                    03  WS-CUSTOMER-NUMBER                  PIC X(6).
00196                    03  WS-CUSTOMER-NAME                    PIC X(30).
00197                    03  WS-ADDRESS-LINE-1                   PIC X(30).
00198                    03  WS-ADDRESS-LINE-2                   PIC X(30).
00199                    03  WS-ADDRESS-LINE-3                   PIC X(30).
00200                    03  WS-ADDRESS-ZIP-CODE                 PIC 9(5).
00201                    03  WS-TELEPHONE-NUMBER                 PIC 9(10).
00202                    03  WS-ATTENTION-LINE                   PIC X(20).
00203                    03  WS-CREDIT-LIMIT-AMOUNT              PIC 9(10)V99.
00204                    03  WS-CURRENT-BALANCE                  PIC 9(10)V99.
00205                    03  WS-PURCHASES-YTD                    PIC 9(10)V99.
00206                    03  WS-NEXT-ORDER-SEQUENCE-NUM          PIC 9(4).
00207                    03  WS-NEXT-PAYMENT-SEQUENCE-NUM        PIC 9(4).
00208
00209       ****************************************************************
00210       *                                                              *
00211       *                          T H E                               *
00212       *                                                              *
00213       *         P R O C E D U R E   D I V I S I O N                  *
00214       *                                                              *
00215       ****************************************************************
00216
00217        PROCEDURE DIVISION USING EXCHANGE-MESSAGE, TRANSACTION-WORKSPACE.
00218
00219        DECLARATIVES.
00220       ****************************************************************
00221       *              I-O ERROR STATUS RETURN SECTION                 *
00222       *                                                              *
00223       *     THIS SECTION IS INVOKED AFTER THE OPERATING SYSTEM       *
00224       *     HAS DETECTED SOME FORM OF I-O ERROR.  THE FILE-STATUS    *
00225       *     WORD IS EXAMINED BY THE ROUTINE, AND AN APPROPRIATE      *
00226       *     ERROR MESSAGE IS FORMATTED AND PLACED INTO THE REPLY     *
00227       *     MESSAGE BUFFER.  THE GO TO SELECTS THE DESIRED ACTION    *
00228       *     THAT FOLLOWS.  TWO CASES CURRENTLY EXIST, THE REPLY      *
```

```
COBOL  3.05   SRC:RDCUST.CBL;7          13-JUL-78  12:10:40 PAGE 007

 00229        *      MESSAGE WHICH RESTARTS THE CURRENT EXCHANGE AND SHOWS *
 00230        *      THE ERRONEOUS DATA ON THE OPERATOR'S SCREEN, AND THE  *
 00231        *      ABORT REPLY MESSAGE WHICH CAUSES THE CURRENT TRANSACT-*
 00232        *      ION TO BE ABORTED WHEN A SEVERE ERROR IS ENCOUNTERED  *
 00233        *      AND RECOVERY IS IMPOSSIBLE.                           *
 00234        ***************************************************************
 00235
 00236         I-O-ERROR SECTION.
 00237             USE AFTER STANDARD ERROR PROCEDURE ON CUSTOM.
 00238         CHECK-FILE-STATUS-CODE.
 00239
 00240             MOVE CUSTOMER-FILE-STATUS TO FILE-STATUS-WORD,RMB-FSW.
 00241             MOVE "  Logical File Name: CUSTOM -CH3" TO
 00242             RMB-FILE-NAME.
 00243
 00244             IF FILE-STATUS-WORD IS EQUAL TO "10"
 00245                     MOVE "Reached End-of-File"
 00246                     TO REPLY-MESSAGE-TEXT
 00247                     GO TO SEND-REPLY-MESSAGE.
 00248
 00249             IF FILE-STATUS-WORD IS EQUAL TO "21"
 00250                     MOVE "Primary Key Sequence Error on WRITE"
 00251                     TO REPLY-MESSAGE-TEXT
 00252                     GO TO SEND-ABORT-MESSAGE.
 00253
 00254             IF FILE-STATUS-WORD IS EQUAL TO "22"
 00255                     MOVE "Duplicate Key Error"
 00256                     TO REPLY-MESSAGE-TEXT
 00257                     GO TO SEND-ABORT-MESSAGE.
 00258
 00259             IF FILE-STATUS-WORD IS EQUAL TO "23"
 00260                     MOVE "No Record Exists under that Key"
 00261                     TO REPLY-MESSAGE-TEXT
 00262                     GO TO SEND-REPLY-MESSAGE.
 00263
 00264             IF FILE-STATUS-WORD IS EQUAL TO "24"
 00265                     MOVE "Boundary Error on Write Statement"
 00266                     TO REPLY-MESSAGE-TEXT
 00267                     GO TO SEND-ABORT-MESSAGE.
 00268
 00269             IF FILE-STATUS-WORD IS EQUAL TO "30"
 00270                     MOVE "Unspecified I/O Error"
 00271                     TO REPLY-MESSAGE-TEXT
 00272                     GO TO SEND-ABORT-MESSAGE.
 00273
 00274             IF FILE-STATUS-WORD IS EQUAL TO "34"
 00275                     MOVE "Permanent Boundary Error on WRITE Statement"
 00276                     TO REPLY-MESSAGE-TEXT
 00277                     GO TO SEND-ABORT-MESSAGE.
 00278
 00279             IF FILE-STATUS-WORD IS EQUAL TO "91"
 00280                     MOVE "File locked by another task"
 00281                     TO REPLY-MESSAGE-TEXT
 00282                     GO TO SEND-REPLY-MESSAGE.
 00283
 00284             IF FILE-STATUS-WORD IS EQUAL TO "92"
 00285             MOVE "The Record you wanted is
 00286        -         " locked by another user.  You may press CLOSE to exit,
```

```
COBOL  3.05   SRC:RDCUST.CBL;7          13-JUL-78  12:10:40 PAGE 008

 00287       -          " or you may wait and press ENTER to try again."
 00288              TO REPLY-MESSAGE-BUFFER.
 00289                      GO TO SEND-REPLY-MESSAGE.
 00290
 00291              IF FILE-STATUS-WORD IS EQUAL TO "93"
 00292                      MOVE "REWRITE or DELETE attempted without prior
 00293       -              "READ being performed."
 00294                      TO REPLY-MESSAGE-TEXT
 00295                      GO TO SEND-REPLY-MESSAGE.
 00296
 00297              IF FILE-STATUS-WORD IS EQUAL TO "94"
 00298                      MOVE "Improper operation attempted"
 00299                      TO REPLY-MESSAGE-TEXT
 00300                      GO TO SEND-ABORT-MESSAGE.
 00301
 00302              IF FILE-STATUS-WORD IS EQUAL TO "95"
 00303                      MOVE "Allocation Failure - No space on device"
 00304                      TO REPLY-MESSAGE-TEXT
 00305                      GO TO SEND-ABORT-MESSAGE.
 00306
 00307              IF FILE-STATUS-WORD IS EQUAL TO "96"
 00308                      MOVE "No buffer space - SAME AREA already in use"
 00309                      TO REPLY-MESSAGE-TEXT
 00310                      GO TO SEND-ABORT-MESSAGE.
 00311
 00312              IF FILE-STATUS-WORD IS EQUAL TO "97"
 00313                      MOVE "Unable to find file named:"
 00314                      TO REPLY-MESSAGE-TEXT
 00315                      GO TO SEND-ABORT-MESSAGE.
 00316
 00317              IF FILE-STATUS-WORD IS EQUAL TO "98"
 00318                      MOVE "Error while attempting to CLOSE file."
 00319                      TO REPLY-MESSAGE-TEXT
 00320                      GO TO SEND-ABORT-MESSAGE.
 00321
 00322              MOVE "UNKNOWN I-O ERROR" TO REPLY-MESSAGE-TEXT
 00323              MOVE FILE-STATUS-WORD TO RMB-FSW.
 00324              GO TO SEND-ABORT-MESSAGE.
 00325
 00326         SEND-REPLY-MESSAGE.
 00327
 00328              MOVE 160 TO BUFFER-SIZE
 00329              MOVE 2 TO REPLY-NUMBER
 00330              CALL "REPLY" USING
 00331                              REPLY-MESSAGE-BUFFER,
 00332                              BUFFER-SIZE,
 00333                              REPLY-NUMBER,
 00334                              STATUS-WORDS,
 00335              GO TO END-ERROR-SECTION.
 00336
 00337         SEND-ABORT-MESSAGE.
 00338
 00339              MOVE 160 TO BUFFER-SIZE
 00340              MOVE 2 TO REPLY-NUMBER
 00341              CALL "ABORT" USING
 00342                              REPLY-MESSAGE-BUFFER
 00343                              BUFFER-SIZE
 00344                              REPLY-NUMBER
COBOL  3.05   SRC:RDCUST.CBL;7          13-JUL-78  12:10:40 PAGE 009

 00345                              STATUS-WORDS.
 00346         END-ERROR-SECTION.
 00347         END DECLARATIVES.
 00348
```

```
COBOL  3.05   SRC:RDCUST.CBL;7          13-JUL-78  12:10:40 PAGE 010

 00349        /*****************************************************************
 00350        *                                                               *
 00351        *                   MAIN PROCESSING ROUTINE                     *
 00352        *                                                               *
 00353        *****************************************************************
 00354         MAIN-TST-ROUTINE SECTION.
 00355
 00356         READ-CUSTOMER-RECORD.
 00357
 00358             MOVE "00" TO FILE-STATUS-WORD.
 00359
 00360             OPEN INPUT CUSTOM.
 00361             IF CUSTOMER-FILE-STATUS IS GREATER THAN "09"
 00362             GO TO END-PROGRAM.
 00363
 00364             IF EM-CUSTOMER-NUMBER IS > "000000"
 00365             GO TO KEY-OK.
 00366                     MOVE 160 TO BUFFER-SIZE,
 00367                     MOVE "You Specified an Invalid Customer ID #"
 00368                     TO REPLY-MESSAGE-BUFFER,
 00369                     MOVE 2 TO REPLY-NUMBER,
 00370                     CALL "REPLY" USING REPLY-MESSAGE-BUFFER,
 00371                                        BUFFER-SIZE,
 00372                                         REPLY-NUMBER,
 00373                                          STATUS-WORDS,
 00374                     GO TO END-PROGRAM.
 00375
 00376         KEY-OK.
 00377             MOVE EM-CUSTOMER-NUMBER TO CUSTOMER-NUMBER.
 00378
 00379             READ WITH LOCK CUSTOM RECORD.
 00380             IF CUSTOMER-FILE-STATUS IS EQUAL TO "92" AND
 00381             FILE-STATUS-WORD IS EQUAL TO "00" GO TO LOCKED-RECORD.
 00382             IF CUSTOMER-FILE-STATUS IS GREATER THAN "09"
 00383             GO TO END-PROGRAM.
 00384
 00385             MOVE CUSTOMER-FILE-RECORD TO WS-CUSTOMER-FILE-RECORD.
 00386
 00387             MOVE CUSTOMER-NUMBER TO RM-CUSTOMER-NUMBER.
 00388             MOVE CUSTOMER-NAME TO RM-CUSTOMER-NAME.
 00389             MOVE ADDRESS-LINE-1 TO RM-ADDRESS-LINE-1.
 00390             MOVE ADDRESS-LINE-2 TO RM-ADDRESS-LINE-2.
 00391             MOVE ADDRESS-LINE-3 TO RM-ADDRESS-LINE-3.
 00392             MOVE ADDRESS-ZIP-CODE TO RM-ADDRESS-ZIP-CODE.
 00393             MOVE TELEPHONE-NUMBER TO RM-TELEPHONE-NUMBER.
 00394             MOVE ATTENTION-LINE TO RM-ATTENTION-LINE.
 00395             MOVE CREDIT-LIMIT-AMOUNT TO RM-CREDIT-LIMIT-AMOUNT.
 00396
 00397             MOVE 173 TO BUFFER-SIZE.
 00398             CALL "PRCEED" USING PROCEED-MESSAGE-BUFFER,
 00399                                 BUFFER-SIZE,
 00400                                  STATUS-WORDS.
 00401
 00402             GO TO END-PROGRAM.
 00403
 00404         LOCKED-RECORD.
 00405
 00406             MOVE "The Record you wanted is
```

```
COBOL  3.05   SRC:RDCUST.CBL;7          13-JUL-78  12:10:40 PAGE 011

 00407       -          " locked by another user.  You may press CLOSE to exit,
 00408       -          " or you may wait and press ENTER to try again,"
 00409            TO REPLY-MESSAGE-BUFFER.
 00410
 00411            MOVE 160 TO BUFFER-SIZE.
 00412            MOVE 2 TO REPLY-NUMBER.
 00413
 00414            CALL "REPLY" USING
 00415                              REPLY-MESSAGE-BUFFER,
 00416                              BUFFER-SIZE,
 00417                              REPLY-NUMBER,
 00418                              STATUS-WORDS.
 00419

COBOL  3.05   SRC:RDCUST.CBL;7          13-JUL-78  12:10:40 PAGE 012

 00420       /*****************************************************************
 00421       *                                                               *
 00422       *          E N D   P R O G R A M   S E C T I O N                *
 00423       *                                                               *
 00424       *****************************************************************
 00425
 00426        END-PROGRAM-SECTION SECTION.
 00427
 00428        END-PROGRAM.
 00429
 00430            EXIT PROGRAM.
```

20.2 THE VALIDC TST

Together with the transaction structure diagram (Figure 18-1), the following figures comprise the documentation needed by an application programmer to write the VALIDC TST.

- VALIDC TST Specification Sheet — Figure 20-3
- Description of VALIDC TST Purpose and Processing — Figure 20-4
- Exchange Message Format — Figure 18-5
- REPLY Message Format — Figure 18-6
- Transaction Workspace Format — Figure 20-5

The finished TST is shown on pages 20-14 through 20-19.

TST SPECIFICATION SHEET

TST Name: VALIDC

Input Object Modules: SY:[1,300]VALIDC.TSK;

:[,]. ;

Language:
- [X] – COBOL
- [ ] – BASIC-PLUS-2
- [ ] – MACRO-11

Is there a resident OTS for the language?
- [X] – YES
- [ ] – NO

Debug Mode?
- [X] – No
- [ ] – Transaction Processor (Device: :)
- [ ] – Standalone (Initializing Module:)

Figure 20-3 VALIDC TST Specification Sheet

The VALIDC TST is the first of two TSTs in the second exchange of the change customer transaction. It receives the updated customer data via the exchange message and checks the data entered by the user for consistency and proper format. During this process, the TST moves the data from its punctuated form in the exchange message to its non-punctuated form in the transaction workspace. From there, the second TST writes it back into the file.

If there is a format error in the user data, the VALIDC TST issues a REPLY response message containing an error message. This message is sent as reply 1. The VALIDC TST then deletes the second TST from the exchange routing list (assuring that incorrect data is not written into the file) and terminates.

Figure 20-4 Description of VALIDC TST Purpose and Processing

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 20-5 Transaction Workspace Format for Change Customer Transaction

```
 COBOL  3.05   SRC:VALIDC.CBL;5          13-JUL-78  12:14:26 PAGE 001

CMD:VALIDC,VALIDC=VALIDC/TST
IDENT: 194122

  00001          IDENTIFICATION DIVISION.
  00002         ************************************************************
  00003         *                                                          *
  00004         *                  T S T     D E S C R I P T I O N         *
  00005         *                                                          *
  00006         ************************************************************
  00007         *
  00008         *  TST NAME:     VALIDC
  00009         *
  00010         *  TRANSACTION:  CHANGE CUSTOMER RECORD (CHGCUS)
  00011         *
  00012         *  FUNCTION:     ACCEPT INPUT DATA REFLECTING TERMINAL OPER-
  00013         *                ATOR'S EDIT OF OLD CUSTOMER RECORD DATA,
  00014         *                PERFORMS EDITING ON DECIMAL FIELDS, AND
  00015         *                PERFORMS ERROR CHECKING ON KNOWN FIELD
  00016         *                STRUCTURES. VALIDATED AND EDITED DATA IS
  00017         *                PLACED IN THE WORKSPACE FOR LATER PROCESSING
  00018         *                BY THE "REWRIT" TST WHICH UPDATES THE
  00019         *                CUSTOMER MASTER FILE.
  00020         *
  00021         *  INPUT FORM:   CHCUS2 - CHANGE CUSTOMER DATA #2 - EDIT RECORD
  00022         *
  00023         *  OUTPUT FORM:  NONE - EXCEPT REPLIES IN CASE OF ERROR.
  00024         *
  00025         ************************************************************
  00026
  00027
```

```
COBOL  3.05   SRC:VALIDC.CBL;5          13-JUL-78  12:14:26 PAGE 002

  000028       /****************************************************************
  000029       *                                                              *
  000030       *                              T H E                           *
  000031       *                                                              *
  000032       *   I D E N T I F I C A T I O N        D I V I S I O N         *
  000033       *                                                              *
  000034       *                                                              *
  000035       ****************************************************************
  000036
  000037        PROGRAM-ID. TSTEP.
  000038        DATE-COMPILED. TODAY.
  000039

COBOL  3.05   SRC:VALIDC.CBL;5          13-JUL-78  12:14:26 PAGE 003

  000040       /****************************************************************
  000041       *                                                              *
  000042       *                              T H E                           *
  000043       *                                                              *
  000044       *     E N V I R O N M E N T        D I V I S I O N             *
  000045       *                                                              *
  000046       *                                                              *
  000047       ****************************************************************
  000048
  000049            13-JUL-78 .
  000050        ENVIRONMENT DIVISION.
  000051        CONFIGURATION SECTION.
  000052            SOURCE-COMPUTER.  PDP-11.
  000053            OBJECT-COMPUTER.  PDP-11.
  000054
  000055
```

```
COBOL  3.05   SRC:VALIDC.CBL;5          13-JUL-78  12:14:26 PAGE 004

 00056        /*****************************************************************
 00057        *                                                               *
 00058        *                            T H E                              *
 00059        *                                                               *
 00060        *                  C A T A      D I V I S I O N                 *
 00061        *                                                               *
 00062        *                                                               *
 00063        *****************************************************************
 00064
 00065
 00066         DATA DIVISION.
 00067
 00068        *****************************************************************
 00069        *                                                               *
 00070        *   W O R K I N G - S T O R A G E      S E C T I O N            *
 00071        *                                                               *
 00072        *****************************************************************
 00073
 00074         WORKING-STORAGE SECTION.
 00075
 00076
 00077        *****************************************************************
 00078        *       M E S S A G E                A R G U M E N T S        *
 00079        *****************************************************************
 00080         01  BUFFER-SIZE                                  PIC 9999 COMP.
 00081
 00082         01  STATUS-WORDS.
 00083
 00084             03  STATUS-WORD-1                            PIC S9(4) COMP.
 00085             03  STATUS-WORD-2                            PIC S9(4) COMP.
 00086
 00087         01  REPLY-NUMBER                                 PIC S9(4) COMP.
 00088
 00089         01  REPLY-MESSAGE-BUFFER                         PIC X(160).
 00090
 00091         01  FORMAT-BUFFER                                PIC S9(12)V9(6)
 00092                 SIGN LEADING SEPARATE CHARACTER.
 00093
 00094         01  FORMAT-BUFFER-LENGTH                         PIC 9(3) COMP
 00095                             VALUE IS 12.
 00096
 00097         01  FORMAT-STATUS                                PIC 9(3) COMP.
 00098
```

```
COBOL  3.05   SRC:VALIDC.CBL;5          13-JUL-78  12:14:26 PAGE 005

00099       /*****************************************************************
00100       *                                                               *
00101       *                  L I N K A G E   S E C T I O N                *
00102       *                                                               *
00103       *****************************************************************
00104
00105        LINKAGE SECTION.
00106
00107       *****                      M E S S A G E                     *****
00108
00109        01  EXCHANGE-MESSAGE.
00110
00111            02  EM-INPUT-FORM-CHCUS2.
00112
00113                    03  EM-CUSTOMER-NUMBER                  PIC X(6).
00114                    03  EM-CUSTOMER-NAME                    PIC X(30).
00115                    03  EM-ADDRESS-LINE-1                   PIC X(30).
00116                    03  EM-ADDRESS-LINE-2                   PIC X(30).
00117                    03  EM-ADDRESS-LINE-3                   PIC X(30).
00118                    03  EM-ADDRESS-ZIP-CODE                 PIC X(5).
00119                    03  EM-TELEPHONE-NUMBER                 PIC X(10).
00120                    03  EM-ATTENTION-LINE                   PIC X(20).
00121                    03  EM-CREDIT-LIMIT-AMOUNT              PIC X(12).
00122
00123
00124
00125
00126       *****                    W O R K S P A C E                   *****
00127
00128        01  TRANSACTION-WORKSPACE.
00129
00130
00131            02  WS-CUSTOMER-FILE-RECORD.
00132
00133                    03  WS-CUSTOMER-NUMBER                  PIC X(6).
00134                    03  WS-CUSTOMER-NAME                    PIC X(30).
00135                    03  WS-ADDRESS-LINE-1                   PIC X(30).
00136                    03  WS-ADDRESS-LINE-2                   PIC X(30).
00137                    03  WS-ADDRESS-LINE-3                   PIC X(30).
00138                    03  WS-ADDRESS-ZIP-CODE                 PIC 9(5).
00139                    03  WS-TELEPHONE-NUMBER                 PIC 9(10).
00140                    03  WS-ATTENTION-LINE                   PIC X(20).
00141                    03  WS-CREDIT-LIMIT-AMOUNT              PIC 9(10)V99.
00142                    03  WS-CURRENT-BALANCE                  PIC 9(10)V99.
00143                    03  WS-PURCHASES-YTD                    PIC 9(10)V99.
00144                    03  WS-NEXT-ORDER-SEQUENCE-NUM          PIC 9(4).
00145                    03  WS-NEXT-PAYMENT-SEQUENCE-NUM        PIC 9(4).
00146
```

```
COBOL  3.05   SRC:VALIDC.CBL;5          13-JUL-78  12:14:26 PAGE 006

 00147        /*****************************************************************
 00148        *                                                               *
 00149        *                            T H E                              *
 00150        *                                                               *
 00151        *          P R O C E D U R E    D I V I S I O N                 *
 00152        *                                                               *
 00153        *****************************************************************
 00154
 00155         PROCEDURE DIVISION USING EXCHANGE-MESSAGE, TRANSACTION-WORKSPACE.
 00156
 00157         MAIN-TST-ROUTINE SECTION.
 00158         EXAMINE-FIELDS.
 00159
 00160             MOVE SPACES TO REPLY-MESSAGE-BUFFER.
 00161             MOVE ZERO TO BUFFER-SIZE.
 00162
 00163             IF EM-CUSTOMER-NUMBER IS EQUAL TO
 00164             WS-CUSTOMER-NUMBER GO TO KEY-OK.
 00165         BAD-KEY.
 00166             MOVE 160 TO BUFFER-SIZE.
 00167             MOVE "Incorrect Value in Customer Number Field"
 00168             TO REPLY-MESSAGE-BUFFER.
 00169             GO TO BAD-DATA.
 00170
 00171         KEY-OK.
 00172             MOVE EM-CUSTOMER-NUMBER TO WS-CUSTOMER-NUMBER.
 00173
 00174             MOVE EM-CUSTOMER-NAME TO WS-CUSTOMER-NAME.
 00175
 00176             MOVE EM-ADDRESS-LINE-1 TO WS-ADDRESS-LINE-1.
 00177             MOVE EM-ADDRESS-LINE-2 TO WS-ADDRESS-LINE-2.
 00178             MOVE EM-ADDRESS-LINE-3 TO WS-ADDRESS-LINE-3.
 00179             MOVE EM-ADDRESS-ZIP-CODE TO WS-ADDRESS-ZIP-CODE.
 00180
 00181             IF EM-TELEPHONE-NUMBER IS > 9999999999
 00182             OR EM-TELEPHONE-NUMBER IS < 0,
 00183             MOVE "TELEPHONE NUMBER INCORRECT" TO
 00184             REPLY-MESSAGE-BUFFER,
 00185             MOVE 160 TO BUFFER-SIZE,
 00186             GO TO BAD-DATA.
 00187
 00188             MOVE EM-TELEPHONE-NUMBER TO WS-TELEPHONE-NUMBER.
 00189
 00190             MOVE EM-ATTENTION-LINE TO WS-ATTENTION-LINE.
 00191
 00192        *****************************************************************
 00193        *                                                               *
 00194        *       STRNUM IS A SUBPROGRAM THAT STRIPS EDITING              *
 00195        *       CHARACTERS FROM A DOLLAR AMOUNT FIELD.                  *
 00196        *       THE RESULTING NUMBER IS FOUND IN THE FIELD              *
 00197        *       NAMED IN THE SECOND ARGUMENT, IN THIS CASE,             *
 00198        *       FORMAT-BUFFER                                           *
 00199        *                                                               *
 00200        *****************************************************************
 00201
 00202             CALL "STRNUM" USING EM-CREDIT-LIMIT-AMOUNT,
 00203                                 FORMAT-BUFFER,
 00204                                  FORMAT-BUFFER-LENGTH,
```

```
COBOL  3.05   SRC:VALIDC.CBL;5          13-JUL-78  12:14:26 PAGE 007

  0@205                                        FORMAT-STATUS.
  0@206
  0@207            MOVE FORMAT-BUFFER TO WS-CREDIT-LIMIT-AMOUNT.

I 0@207 0371  POSSIBLE HIGH ORDER RECEIVING FIELD TRUNCATION.


I 0@207 0372  POSSIBLE LOW ORDER RECEIVING FIELD TRUNCATION.

  0@208            GO TO END-PROGRAM.
  0@209
  0@210        BAD-DATA.
  0@211
  0@212            CALL "CALLRT" USING STATUS-WORDS.
  0@213            MOVE 2 TO REPLY-NUMBER.
  0@214            CALL "REPLY" USING REPLY-MESSAGE-BUFFER,
  0@215                               BUFFER-SIZE,
  0@216                               REPLY-NUMBER,
  0@217                                STATUS-WORDS.
  0@218


COBOL  3.05   SRC:VALIDC.CBL;5          13-JUL-78  12:14:26 PAGE 008

  0@219       /*****************************************************************
  0@220       *                                                               *
  0@221       *          E N D   P R O G R A M   S E C T I O N                *
  0@222       *                                                               *
  0@223       *****************************************************************
  0@224
  0@225        END-PROGRAM-SECTION SECTION.
  0@226
  0@227        END-PROGRAM.
  0@228
  0@229            EXIT PROGRAM.
```

20.3 THE REWRIT TST

Together with the transaction structure diagram (Figure 18-1), the following figures comprise the documentation needed by an application programmer to write the REWRIT TST.

- REWRIT TST Specification Sheet — Figure 20-6
- Description of REWRIT TST Purpose and Processing — Figure 20-7
- Transaction Workspace Format — Figure 20-5
- Customer Record Format — Figure 22-1
- REPLY Message Format — Figure 18-7

The finished TST is shown on pages 20-21 through 20-29.

TST SPECIFICATION SHEET

TST Name: [R][E][W][R][I][T]

Input Object Modules: [S][Y][ ]:[ ][ ][1],[3][0][0] [R][E][W][R][I][T].[T][S][K];[ ][ ]

[ ][ ][ ]:[ ][ ][ ][ ],[ ][ ][ ][ ] [ ][ ][ ][ ][ ][ ].[ ][ ][ ];[ ][ ]

Language:
- [X] – COBOL
- [ ] – BASIC-PLUS-2
- [ ] – MACRO-11

Is there a resident OTS for the language?
- [X] – YES
- [ ] – NO

Debug Mode?
- [X] – No
- [ ] – Transaction Processor (Device: [ ][ ][ ] :)
- [ ] – Standalone (Initializing Module: [ ][ ][ ][ ][ ][ ])

Figure 20-6 WRITE TST Specification Sheet

The REWRIT TST is the second of two TSTs in the second exchange of the change customer transaction. It receives the verified customer data via the transaction workspace and writes this data in the customer file.

After writing the data in the customer file, the REWRIT TST sends a confirmation message as a REPLY response message containing no text which invokes reply 2.

The TST aborts the transaction if it encounters an error condition during the file writing operation.

Figure 20-7 Description of REWRIT TST Purpose and Processing

```
 COBOL  3.05   SRC:REWRIT.CBL;4          13-JUL-78  12:16:24 PAGE 001

CMD:REWRIT,REWRIT=REWRIT/TST
IDENT: 194122


  00001         IDENTIFICATION DIVISION.
  00002        ************************************************************
  00003        *                                                          *
  00004        *                 T S T     D E S C R I P T I O N          *
  00005        *                                                          *
  00006        ************************************************************
  00007        *
  00008        *  TST NAME:      REWRIT
  00009        *
  00010        *  TRANSACTION:   CHGCUS - CHANGE CUSTOMER RECORD.
  00011        *
  00012        *  FUNCTION:      THIS TST TAKES THE INFORMATION FROM FORM CHCUS2
  00013        *                 THAT HAS BEEN VALIDATED BY THE TST "VALIDC"
  00014        *                 AND UPDATES THE CUSTOMER MASTER FILE WITH
  00015        *                 THE EDITED CUSTOMER RECORD.
  00016        *
  00017        *  FILES:         CUSTOMER MASTER FILE "CUSTOM.DAT"
  00018        *
  00019        *  INPUT FORM:    CHCUS2 - AFTER IT HAS BEEN PROCESSED BY VALIDC.
  00020        *
  00021        *  OUTPUT FORM:   CHCUS2 - REPLY #1 - TXN COMPLETE,
  00022        *                          REPLY #2 - ERROR ON I-O OPERATION.
  00023        *
  00024        ************************************************************
  00025
  00026


 COBOL  3.05   SRC:REWRIT.CBL;4          13-JUL-78  12:16:24 PAGE 002

  00027        /***********************************************************
  00028        *                                                          *
  00029        *                          T H E                           *
  00030        *                                                          *
  00031        *   I D E N T I F I C A T I O N       D I V I S I O N      *
  00032        *                                                          *
  00033        *                                                          *
  00034        ************************************************************
  00035
  00036         PROGRAM-ID. TSTEP.
  00037         DATE-COMPILED. TODAY.
  00038
```

```
LOGOL  3.05   SRC:REWRIT.CBL;4          13-JUL-78  12:16:24 PAGE 003

00039        /*****************************************************************
00040        *                                                               *
00041        *                              T H E                            *
00042        *                                                               *
00043        *     E N V I R O N M E N T        D I V I S I O N              *
00044        *                                                               *
00045        *                                                               *
00046        *****************************************************************
00047
00048             13-JUL-78 .
00049         ENVIRONMENT DIVISION.
00050         CONFIGURATION SECTION.
00051             SOURCE-COMPUTER.  PDP-11.
00052             OBJECT-COMPUTER.  PDP-11.
00053
00054        *****************************************************************
00055        *                                                               *
00056        *          I N P U T - O U T P U T    S E C T I O N             *
00057        *                                                               *
00058        *****************************************************************
00059
00060         INPUT-OUTPUT SECTION.
00061
00062         FILE-CONTROL.
00063
00064             SELECT CUSTOM ASSIGN TO "CUSTOM.DAT"
00065                    ORGANIZATION IS INDEXED
00066                    ACCESS MODE IS DYNAMIC
00067                    RECORD KEY IS CUSTOMER-NUMBER
00068                    ALTERNATE RECORD KEY IS CUSTOMER-NAME WITH DUPLICATES
00069                    FILE STATUS IS CUSTOMER-FILE-STATUS.
00070
```

```
COBOL  3.25   SRC:PENRIT.CBL;4             13-JUL-79  12:16:24 PAGE 004

      /************************************************************
      *                                                           *
      *                          T H E                            *
      *                                                           *
      *              D A T A      D I V I S I O N                 *
      *                                                           *
      *                                                           *
      *************************************************************


       DATA DIVISION.

      *************************************************************
      *                                                         *
      *           F I L E        S E C T I O N                  *
      *                                                           *
      *************************************************************

       FILE SECTION.

       FD  CUSTOM
           LABEL RECORDS ARE STANDARD
           VALUE OF ID IS CUSTOM-CHANNEL-NUMBER
           DATA RECORD IS CUSTOM-FILE-RECORD.

       01  CUSTOM-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  CURRENT-BALANCE                        PIC 9(10)V99.
           03  PURCHASES-YTD                          PIC 9(10)V99.
           03  NEXT-ORDER-SEQUENCE-NUMBER             PIC 9(4).
           03  NEXT-PAYMENT-SEQUENCE-NUMBER           PIC 9(4).


      *************************************************************
      *                                                           *
      *  W O R K I N G - S T O R A G E     S E C T I O N          *
      *                                                           *
      *************************************************************

       WORKING-STORAGE SECTION.

      *************************************************************
      *             F I L E              C H A N N E L S          *
      *************************************************************

       01  CUSTOM-CHANNEL-NUMBER              PIC X(11)
                 VALUE IS "CUSTOM/CH:3".

```

```
COBOL  3.05   SRC:REWRIT.CBL;4          13-JUL-78  12:16:24 PAGE 005

 00129       ****************************************************************
 00130       *         F I L E    S T A T U S    N A M E S                  *
 00131       ****************************************************************
 00132
 00133        01  FILE-STATUS-WORD                          PIC XX.
 00134        01  CUSTOMER-FILE-STATUS                      PIC XX.
 00135
 00136
 00137       ****************************************************************
 00138       *       M E S S A G E                 A R G U M E N T S      *
 00139       ****************************************************************
 00140        01  BUFFER-SIZE                               PIC 9999 COMP.
 00141
 00142
 00143        01  STATUS-WORDS.
 00144
 00145            03  STATUS-WORD-1                         PIC S9(4) COMP.
 00146            03  STATUS-WORD-2                         PIC S9(4) COMP.
 00147
 00148        01  REPLY-NUMBER                              PIC S9(4) COMP.
 00149
 00150        01  REPLY-MESSAGE-BUFFER.
 00151            03  REPLY-MESSAGE-TEXT                    PIC X(80).
 00152            03  REPLY-FILLER                          PIC X(18)
 00153                    VALUE IS "File Status Word: ".
 00154            03  RMB-FSW                               PIC X(2).
 00155            03  RMB-FILE-NAME                         PIC X(60).
 00156
```

```
COBOL  3.05   SRC:REWRIT.CBL;4          13-JUL-78  12:16:24 PAGE 006

00157       /*****************************************************************
00158       *                                                               *
00159       *                    L I N K A G E   S E C T I O N              *
00160       *                                                               *
00161       *****************************************************************
00162
00163        LINKAGE SECTION.
00164
00165       *****                       M E S S A G E                       *****
00166
00167        01  EXCHANGE-MESSAGE.
00168
00169            02  EM-INPUT-FORM-CHCUS2.
00170
00171                    03  EM-CUSTOMER-NUMBER                  PIC X(6).
00172                    03  EM-CUSTOMER-NAME                    PIC X(30)
00173                    03  EM-ADDRESS-LINE-1                   PIC X(30).
00174                    03  EM-ADDRESS-LINE-2                   PIC X(30).
00175                    03  EM-ADDRESS-LINE-3                   PIC X(30).
00176                    03  EM-ADDRESS-ZIP-CODE                 PIC 9(5).
00177                    03  EM-TELEPHONE-NUMBER                 PIC 9(10).
00178                    03  EM-ATTENTION-LINE                   PIC X(20).
00179                    03  EM-CREDIT-LIMIT-AMOUNT              PIC X(12).
00180
00181
00182
00183
00184       *****                     W O R K S P A C E                     *****
00185
00186        01  TRANSACTION-WORKSPACE.
00187
00188
00189            02  WS-CUSTOM-FILE-RECORD.
00190
00191                    03  WS-CUSTOMER-NUMBER                  PIC X(6).
00192                    03  WS-CUSTOMER-NAME                    PIC X(30).
00193                    03  WS-ADDRESS-LINE-1                   PIC X(30).
00194                    03  WS-ADDRESS-LINE-2                   PIC X(30).
00195                    03  WS-ADDRESS-LINE-3                   PIC X(30).
00196                    03  WS-ADDRESS-ZIP-CODE                 PIC 9(5).
00197                    03  WS-TELEPHONE-NUMBER                 PIC 9(10).
00198                    03  WS-ATTENTION-LINE                   PIC X(20).
00199                    03  WS-CREDIT-LIMIT-AMOUNT              PIC 9(10)V99.
00200                    03  WS-CURRENT-BALANCE                  PIC 9(10)V99.
00201                    03  WS-PURCHASES-YTD                    PIC 9(10)V99.
00202                    03  WS-NEXT-ORDER-SEQUENCE-NUM          PIC 9(4).
00203                    03  WS-NEXT-PAYMENT-SEQUENCE-NUM        PIC 9(4).
00204
00205
00206
00207
```

```
COBOL  3.05   SRC:REWRIT.CBL;4          13-JUL-78  12:16:24 PAGE 007

00208       /*****************************************************************
00209       *                                                               *
00210       *                            T H E                              *
00211       *                                                               *
00212       *         P R O C E D U R E    D I V I S I O N                  *
00213       *                                                               *
00214       *****************************************************************
00215
00216        PROCEDURE DIVISION USING EXCHANGE-MESSAGE, TRANSACTION-WORKSPACE.
00217        DECLARATIVES.
00218       ******************************************************************
00219       *               I-O ERROR STATUS RETURN SECTION                  *
00220       *                                                                *
00221       *     THIS SECTION IS INVOKED AFTER THE OPERATING SYSTEM         *
00222       *     HAS DETECTED SOME FORM OF I-O ERROR.  THE FILE-STATUS      *
00223       *     WORD IS EXAMINED BY THE ROUTINE, AND AN APPROPRIATE        *
00224       *     ERROR MESSAGE IS FORMATTED AND PLACED INTO THE REPLY       *
00225       *     MESSAGE BUFFER.  THE GO TO SELECTS THE DESIRED ACTION      *
00226       *     THAT FOLLOWS.  TWO CASES CURRENTLY EXIST, THE REPLY        *
00227       *     MESSAGE WHICH RESTARTS THE CURRENT EXCHANGE AND SHOWS      *
00228       *     THE ERRONEOUS DATA ON THE OPERATOR'S SCREEN, AND THE       *
00229       *     ABORT REPLY MESSAGE WHICH CAUSES THE CURRENT TRANSACT-     *
00230       *     ION TO BE ABORTED WHEN A SEVERE ERROR IS ENCOUNTERED       *
00231       *     AND RECOVERY IS IMPOSSIBLE.                                *
00232       ******************************************************************
00233
00234        I-O-ERROR SECTION.
00235            USE AFTER STANDARD ERROR PROCEDURE ON CUSTOM.
00236        CHECK-FILE-STATUS-CODE.
00237
00238            IF CUSTOMER-FILE-STATUS IS GREATER THAN "01"
00239            MOVE CUSTOMER-FILE-STATUS TO FILE-STATUS-WORD,RMB-FSW.
00240
00241            MOVE "  Logical File Name: CUSTOM -CH3" TO
00242            RMB-FILE-NAME.
00243
00244            IF FILE-STATUS-WORD IS EQUAL TO "10"
00245                    MOVE "Reached End-of-File"
00246                    TO REPLY-MESSAGE-TEXT
00247                    GO TO SEND-REPLY-MESSAGE.
00248
00249            IF FILE-STATUS-WORD IS EQUAL TO "21"
00250                    MOVE "Primary Key Sequence Error on WRITE"
00251                    TO REPLY-MESSAGE-TEXT
00252                    GO TO SEND-ABORT-MESSAGE.
00253
00254            IF FILE-STATUS-WORD IS EQUAL TO "22"
00255                    MOVE "Duplicate Key Error"
00256                    TO REPLY-MESSAGE-TEXT
00257                    GO TO SEND-ABORT-MESSAGE.
00258
00259            IF FILE-STATUS-WORD IS EQUAL TO "23"
00260                    MOVE "No Record Exists under that Key"
00261                    TO REPLY-MESSAGE-TEXT
00262                    GO TO SEND-REPLY-MESSAGE.
00263
00264            IF FILE-STATUS-WORD IS EQUAL TO "24"
00265                    MOVE "Boundary Error on Write Statement"
```

```
COBOL  3.05   SRC:<RF.RIT.C-L;4          13-JUL-78  12:16:24 PAGE 008

00266                   TO REPLY-MESSAGE-TEXT
00267                   GO TO SEND-ABORT-MESSAGE.
00268
00269           IF FILE-STATUS-WORD IS EQUAL TO "30"
00270                   MOVE "Unspecified I/O Error"
00271                   TO REPLY-MESSAGE-TEXT
00272                   GO TO SEND-ABORT-MESSAGE.
00273
00274           IF FILE-STATUS-WORD IS EQUAL TO "34"
00275                   MOVE "Permanent Boundary Error on WRITE Statement"
00276                   TO REPLY-MESSAGE-TEXT
00277                   GO TO SEND-ABORT-MESSAGE.
00278
00279           IF FILE-STATUS-WORD IS EQUAL TO "91"
00280                   MOVE "File locked by another task"
00281                   TO REPLY-MESSAGE-TEXT
00282                   GO TO SEND-REPLY-MESSAGE.
00283
00284           IF FILE-STATUS-WORD IS EQUAL TO "92"
00285                   MOVE "Record locked by another task"
00286                   TO REPLY-MESSAGE-TEXT
00287                   GO TO SEND-REPLY-MESSAGE.
00288
00289           IF FILE-STATUS-WORD IS EQUAL TO "93"
00290                   MOVE "REWRITE or DELETE attempted without prior
00291                   "READ being performed."
00292                   TO REPLY-MESSAGE-TEXT
00293                   GO TO SEND-REPLY-MESSAGE.
00294
00295           IF FILE-STATUS-WORD IS EQUAL TO "94"
00296                   MOVE "Improper operation attempted"
00297                   TO REPLY-MESSAGE-TEXT
00298                   GO TO SEND-ABORT-MESSAGE.
00299
00300           IF FILE-STATUS-WORD IS EQUAL TO "95"
00301                   MOVE "Allocation Failure - No space on device"
00302                   TO REPLY-MESSAGE-TEXT
00303                   GO TO SEND-ABORT-MESSAGE.
00304
00305           IF FILE-STATUS-WORD IS EQUAL TO "96"
00306                   MOVE "No buffer space - SAME AREA already in use"
00307                   TO REPLY-MESSAGE-TEXT
00308                   GO TO SEND-ABORT-MESSAGE.
00309
00310           IF FILE-STATUS-WORD IS EQUAL TO "97"
00311                   MOVE "Unable to find file named:"
00312                   TO REPLY-MESSAGE-TEXT
00313                   GO TO SEND-ABORT-MESSAGE.
00314
00315           IF FILE-STATUS-WORD IS EQUAL TO "98"
00316                   MOVE "Error while attempting to CLOSE file."
00317                   TO REPLY-MESSAGE-TEXT
00318                   GO TO SEND-ABORT-MESSAGE.
00319
00320           MOVE "UNKNOWN I-O ERROR" TO REPLY-MESSAGE-TEXT
00321           MOVE FILE-STATUS-WORD TO RMB-FSW
00322           GO TO SEND-ABORT-MESSAGE.
00323
```

```
COBOL  3.05   SRC:REWRIT.CBL;4          13-JUL-78  12:16:24 PAGE 009

 00324        SEND-REPLY-MESSAGE.
 00325
 00326            MOVE 160 TO BUFFER-SIZE
 00327            MOVE 2 TO REPLY-NUMBER
 00328            CALL "REPLY" USING
 00329                             REPLY-MESSAGE-BUFFER,
 00330                             BUFFER-SIZE,
 00331                             REPLY-NUMBER,
 00332                             STATUS-WORDS.
 00333            GO TO END-ERROR-SECTION.
 00334
 00335        SEND-ABORT-MESSAGE.
 00336
 00337            MOVE 160 TO BUFFER-SIZE
 00338            MOVE 2 TO REPLY-NUMBER
 00339            CALL "ABORT" USING
 00340                             REPLY-MESSAGE-BUFFER
 00341                             BUFFER-SIZE
 00342                             REPLY-NUMBER
 00343                             STATUS-WORDS.
 00344        END-ERROR-SECTION.
 00345        END DECLARATIVES.
 00346
```

```
COBOL  3.05   SRC:REWRIT.CBL;4          13-JUL-78  12:16:24 PAGE 010

 00347        /****************************************************************
 00348        *                                                               *
 00349        *                   MAIN PROCESSING ROUTINE                     *
 00350        *                                                               *
 00351        ****************************************************************
 00352
 00353         MAIN-TST-ROUTINE SECTION.
 00354         REWRITE-CUSTOMER-RECORD.
 00355
 00356
 00357             OPEN I-O CUSTOM.
 00358             IF CUSTOMER-FILE-STATUS IS GREATER THAN "09"
 00359             GO TO END-PROGRAM.
 00360
 00361             MOVE WS-CUSTOMER-NUMBER TO CUSTOMER-NUMBER.
 00362
 00363             REWRITE WITH UNLOCK CUSTOM-FILE-RECORD
 00364             FROM WS-CUSTOM-FILE-RECORD.
 00365
 00366             IF CUSTOMER-FILE-STATUS IS GREATER THAN "09"
 00367             GO TO END-PROGRAM.
 00368
 00369             MOVE 1 TO REPLY-NUMBER
 00370             MOVE SPACES TO REPLY-MESSAGE-BUFFER
 00371             MOVE 0 TO BUFFER-SIZE.
 00372             CALL "REPLY" USING REPLY-MESSAGE-BUFFER,
 00373                                BUFFER-SIZE,
 00374                                 REPLY-NUMBER,
 00375                                  STATUS-WORDS.
 00376
 00377
 00378        ****************************************************************
 00379        *                                                               *
 00380        *         E N D   P R O G R A M   S E C T I O N                 *
 00381        *                                                               *
 00382        ****************************************************************
 00383
 00384         END-PROGRAM-SECTION SECTION.
 00385
 00386         END-PROGRAM.
 00387
 00388             EXIT PROGRAM.
```

CHAPTER 21
DESIGNING AND SPECIFYING FILES

Files are an important part of commercial data processing systems. Most commercial systems maintain a large volume of data, and many have a large number of inquiries and updates against that data. In many applications, file design is by far the most important determining factor in the system's performance. For these reasons, on-line commercial applications require the utmost care in file design.

File design is, of necessity, intertwined with other system design steps. For example, you must have some idea of the files needed in your application before you can design transactions. And conversely, you must have some idea of the transactions in your application before you can design files.

This manual cannot provide a complete course in the effective design of files for on-line commercial applications. If you are not experienced in this kind of file design, you might find it helpful to consult a text on the subject. But to assist you in designing and specifying TRAX files, this chapter discusses the following specific topics:

- File design prerequisites
- Data recording formats
- Codes
- Relationships between fields
- Computing field and record sizes
- Potential record usage problems
- Choosing a file organization
- Calculating file sizes
- File reliability and recovery
- Checking file performance
- Documenting file design

21.1 FILE DESIGN PREREQUISITES

Chapter 12 summarized the systems analysis steps you must take before attempting a detailed file design. These are:

1. *Business Data Requirements.* You must study the items of data that your business application requires you to store in files.
2. *Data Groupings.* You must understand the groups in which the data are stored and retrieved. For example, an outstanding balance might be associated with a customer, an order, or an invoice.
3. *Data Item Size.* You must understand the size and composition of each data item.
4. *Data Quantity.* You must understand how many of each data item must be kept in the application files.
5. *Access Patterns.* You must understand how data items are commonly retrieved. For example, a group of customer data items might commonly be retrieved either by name or by customer number.
6. *Data Security Issues.* You must know which system users should be given access to a data item and which should not. For instance, a customer's credit rating should be inaccessible from an order-taker's terminal but accessible from the credit manager's terminal.
7. *Performance Issues.* You must understand the business environment and the proposed application design so that you can predict where file access problems may occur and where these problems would be serious.

21.2 DATA RECORDING FORMAT

Study the data recording formats available in your programming language. COBOL, for instance, allows computational and display data items. BASIC-PLUS-2 allows string variables and binary numeric variables. In either language, a numeric quantity can be expressed in both forms; but each language limits what the programmer can expect with a given data format.

By using the display or string data formats, you usually sacrifice computational speed and space. But you gain data portability and easy file maintenance. These data formats are character-oriented; their meaning does not depend on variations of machine architecture, and they can be read by programs written in different languages and run on different machines.

By using computational or binary data formats, you usually gain computational speed and compactness; but you may lose data portability and easy file maintenance. These formats often limit your application to one particular language or machine.

Sometimes, a single overriding concern dictates the format you must use. For example, a system with a large data base may require a compact data recording format. On the other hand, a system that uses several programming languages or one that communicates data to other systems may require maximum portability of data. The first system should use computational or binary data formats; the second system, string or display formats.

In the absence of overriding application requirements, use the data recording format that is most convenient in your programming language.

If you decide on a substantial number of binary-format data fields, remember that your application programmers will have to construct file dump utility programs to inspect the files and debug their application programs. Alternatively, your programmers may find the DATATRIEVE language useful in inspecting and debugging data files. (See *TRAX DATATRIEVE User's Guide,* AA-D347A-TC.)

21.3 CODES

Where a data field can have only a few possible, known values you can use codes instead of a complete value. Using such codes saves significant file space.

For example, a customer file may require a field describing each customer's relationship with the business. Although the relationships might be expressed in words such as SHOPKEEPER, COLLECTOR, and MAILORDER-OPERATOR, it would be more sensible to list the relationships and develop a compact code to be placed in the file instead.

Do not adopt a coding scheme without careful thought. Code schemes can have significant problems that adversely affect the performance or reliability of the application.

- *Excessive Rigidity.* Code schemes must allow expansion and adjustment. For example, a code that starts with seven values may have fifteen within a few months. If you only allow space in your file for single-digit codes, you must soon reorganize the file for two digit codes. Plan ahead and you can avoid this problem.
- *Code Interpretation for Users.* Although you may find codes attractive because they save file space, they may be considerably less attractive to your application's users. Where possible, you should allow users to enter information in its uncoded form. Only within the application programs and data structures should you rely exclusively on the code values.
- *Vulnerability of Coded Data.* Coded data is useless if the relationship between the codes and their meanings is lost or altered. Further, changing the meaning of code instantaneously affects each record carrying that coded value.

For example, a customer record might carry a coded field specifying the discount that a customer is allowed. If a discount class is made more liberal and the same identifying code is kept, customers with that code immediately benefit from the revised discount policy.

This is often the intended effect. But what happens to records of old orders? Order records may carry references to discount codes. If so, the revised discount policy will be associated with old orders as well as with current and future orders. This could cause confusion if an old order were subsequently the subject of a customer inquiry. The company's staff would be hard put to justify the original discount amount which had, in fact, been correctly computed according to the discount policy then in effect.

21.4 RELATIONSHIPS BETWEEN FIELDS

The files you design for your application will probably have relationships between fields. This means that certain fields contain data that refers to other fields or records. For example, a record describing an order has a customer identification number. Such a number refers to a record in the customer file containing the full name and address of the customer.

Minimize the relationships between fields when designing your application files. Each new relationship complicates other aspects of file design, such as updates to shared files and reliability after system outages.

There are two important kinds of inter-field relationships:

1. *Logical relationships* are based on the names or values of related fields. A logical relationship indicates that a set of information is related *by name* to a second set of information.
2. *Physical relationships* are based on the place or location where related information is found. A physical relationship indicates that a set of information is related to a second set of information which is found *in a specified place.*

For example, an invoice record could be related to a customer record in two ways:

1. *A logical relationship* could be introduced by placing a customer number in the invoice record. This customer number represents a logical relationship because it specifies *who* the invoice belongs to, rather than *where* the owner's name can be found.
2. *A physical relationship* could be introduced by placing a record number in the invoice record. This record number would represent a physical relationship because it specifies *where* we can find the name of the customer who owns the invoice, rather than *who* the customer is.

A logical relationship does not rely upon a file's internal structure or a record's position on a storage device. A file with a logical relationship can be moved from place to place, and the relationship is not destroyed. The order of records in the file can change (assuming that the file's logical structure remains intact) and the relationships are still valid. However, more time is required to access data through a logical relationship, because the logical relationship must be reduced to a physical relationship by consulting the file's index or other logical structure.

Physical relationships have faster access speeds, which is their principal attraction. Physical relationships can be disrupted by moving a file or by restructuring it to allow additional records. Two common file operations – file reorganization and file backup – can disrupt physical relationships. Trouble arises when a set of files is restored from backup media, and the files were not originally backed up on the same date: that is, they are of different generations. It takes special care to design physical relationships so that situations such as these do not destroy the proper relationships between fields.

Try to use logical relationships first, and then resort to physical relationships only where the need for better performance has been proven. Important application characteristics such as stability, reliability, maintainability, flexibility, and ease of operation are more easily attained with logical field relationships. Do not needlessly sacrifice these objectives for performance.

21.5 COMPUTING FIELD AND RECORD SIZES

To compute field and record sizes:

1. Group your application fields into records.
2. Determine each field size.
3. Compute each record size by summing its field sizes.

To determine the size of each data field, study the data recording techniques of your programming language. (Consult the applicable programming language reference manual.) When you compute the size of each record, include space for growth in the number of fields in the record.

Next, look at the record lengths. These situations are potential problems, and you should redesign your records to avoid them where possible:

- Large records
- Occasionally-used fields
- Variable record lengths in relative files

21.5.1 Large Records

Records with more than 150 bytes are large records. Although RMS handles records up to several thousand bytes long, you gain several advantages by dividing the data into smaller records. For instance, if different users require access to different subsets of data, you could form these subsets into separate records. Then one user could read (and lock, if necessary) one set of data while another has similar access to the other set. Besides reducing access conflicts, this arrangement allows each program to devote less space to record buffers.

If you need large records, devote extra attention to later phases of the file design. Large records make efficient blocking and deblocking difficult, and poor file design wastes large amounts of space as logical record boundaries conflict with physical device block boundaries. Large records also require more buffer space in each TST, as well as more space in the transaction's system workspace if they are staged.

21.5.2 Occasionally-Used Fields

If a record has many fields that are only occasionally filled with data, the record will contain a lot of wasted space. Divide such records into two or more segments, each a record itself. One record will contain the fields which are always filled; others contain those fields which are used occasionally. These latter records need not be placed in the file if the corresponding data is not present, and you may save considerable space.

Occasionally-used fields also require you to devise a data value which, when placed in the field, indicates that the field contains no data. Select a value that does not correspond with a valid data value that might appear in the field.

21.5.3 Variable Record Lengths in Relative Files

When a relative file is constructed, each record slot is allocated the same length; if you use records of different lengths, you must allocate the size of the largest record. If the record lengths differ significantly and, especially, if there are more short records in the file than long ones, considerable file space is wasted.

21.6 POTENTIAL RECORD USE PROBLEMS

Be careful when you group fields into records. You can create design or performance problems.

Consider the ways that the application's transactions access the records you defined. Watch for these potential problems:

- High activity on particular records
- A large working set of records
- Record locks of long duration

21.6.1 High Activity on Particular Records

When one or more transactions read a common record, the access volume to that record is high. This reduces system performance and degrades response times. Adapt your design to compensate for this increased volume.

21.6.2 Large Working Set of Records

The set of records needed by a transaction is called its *record working set.* A transaction that must access (and perhaps lock) a large number of records simultaneously may cause problems:

- Other transactions may not access the records
- The transaction instance requires a large record buffer space, either in TSTs or in the transaction workspace
- If any records are staged or journaled, additional buffer space is required in the transaction instance's system workspace.

A different allocation of data fields between records may reduce a transaction's record working set to manageable proportions.

21.6.3 Record Locks of Long Duration

Problems arise if a transaction locks records for a lengthy period. If the record is not accessed heavily, there may be no problem. But you may have a serious problem if the records locked for long periods are also those with high access volumes.

Study the time that each record is locked to see whether that period of time creates conflicts with other application users. Remember that updates to staged and journaled files do not unlock records until the end of each transaction instance.

Remember, too, that any lock duration that involves user interaction must be assumed to be lengthy.

If you discover potential record locking problems, restructure your files to eliminate them. For example, if journaling or staging are the cause of the problem, segregate data into two or more files: place data to be journaled or staged in one file and place less sensitive data in another.

A control record at the front of a master file is often used to indicate the next available serial number or the number of records in the file. If the data in such a file is staged or journaled, serious problems can develop. This control record is usually a high-access record, and staging or journaling extends the time any transaction instance locks it. Place the control record in a different file (unstaged, of course) to avoid this problem.

21.7 CHOOSING A FILE ORGANIZATION

After grouping fields into records, you must select a file organization to store and retrieve records efficiently.

Most files used in commercial applications are either sequential files or indexed files. Sequential files are useful for data that is not accessed randomly from on-line transactions; work files, history files, and log files fall in this category. Indexed files are indispensable for on-line random access situations. You may also find that relative-record files are useful in on-line random access situations where physical field relationships are appropriate. (See Section 21.4.)

The design of sequential files presents no special difficulty. However, the design of each indexed file deserves attention on two levels:

1. *Functional design.* This involves decisions such as the number of indexes and the composition of each index. Keep the number of indexes for each file to a minimum. Each additional index means overhead when a record is added or deleted.

You began this process during the business analysis described in Chapter 12. Now, you must polish the functional design and see that it is adequate for the additional design you have since done.

2. *Technical design.* For this, you need to study RMS carefully with your files in mind. RMS is a flexible data management system with many parameters that you can vary as you design each file. These parameters are important, because each affects the performance of the finished file.

21.8 CALCULATING FILE SIZES

Now you can calculate the size of your data files. This calculation must consider:

1. The size of each type of file record
2. The number of each type of file record
3. Wasted space for such things as blocking factors and inter-record gaps on magnetic tape
4. Space for indexes
5. Space for application growth

You should provide for application growth in two ways:

1. Leaving space in each record to add new data fields.
2. Increasing file size for future records.

With your detailed RMS file design, you can calculate the space wasted for blocking and deblocking disk-based files. There is no wasted space on magnetic tape, but the space taken by inter-record gaps must be considered.

When you have calculated the size of each file, check that each file physically fits on its designated disk device or tape reel. If it does not fit, you must partition it. This is done in two ways:

1. The record layout can be partitioned and a portion put in each of two or more files. This creates twice as many records as in the original file, but each record is proportionally shorter than before. This method requires extra overhead space to store duplicate indexes for the extra files.
2. The file can be partitioned so that half of the records are placed in each of two files. The record layout is not changed. This method requires no additional index overhead space. If the original file contained multiple indexes, though, you must choose one index to govern the file division. When accessing by that index, TSTs will know which file to consult. But when accessing by other indexes, they will have to consult both files since they will not know beforehand which file may contain the desired record; this may create a performance or buffer space problem.

21.9 FILE RELIABILITY AND RECOVERY

You laid the groundwork for file security in your design for application files by selecting a design that allows orderly shared access and by avoiding inter-field relationships.

Earlier in the previous chapter, you read some of the advantages that can be gained by segregating data into two or more files when they might fit in one. File reliability is often gained as well. If you segregate updated fields from those referenced in read-only fashion, you accomplish several things:

- Only the file containing updated data need be locked when accessed. This reduces access conflicts.
- Only that file need be staged or journaled. This reduces the time and buffer space needed for these functions.
- Only that file need be reconstructed after a system outage. This reduces the time needed to recover from a system outage.

Judicious use of the TRAX staging and journaling capabilities make recovery easier and quicker after a system outage. But remember the impact these options have on system performance; do not choose them without considering the tradeoffs involved.

It is time to think about the operational aspects of your application design, such as file backup schedules. Journaled files can be reconstructed if a backup for the files exists along with a complete set of journals for the period between the backup and the system outage. The generation and preservation of these backups and journals require carefully-devised operations schedules and policies.

The time needed to reconstruct a file from journals is proportional to the time between the last backup and the system outage. A recent backup has fewer journals to process and therefore requires less recovery time. Too many backups interfere with normal system operation; too few make recovery a lengthy process.

21.10 CHECKING FILE PERFORMANCE

Once you have a detailed design for your application files, consider whether the design provides the performance the application demands.

Approach this from two points of view:

1. *Search for files or records that may cause a bottleneck.* To do this, concentrate on each file in turn; imagine the pattern of access requests made on the file by the application transactions.
2. *Search for transactions that may not provide acceptable response times.* To do this, study each transaction and estimate the time required for the transaction file accesses. Allow time for delayed access to records with high access volumes.

If you have a borderline case, set up a benchmark test on a TRAX system. You need not program the entire application or even an entire transaction. But you can set up files, fill them with dummy data, and then run tests to see how long each set of accesses takes.

21.11 DOCUMENTING THE FILE DESIGN

When you have a satisfactory file design, document the design so that application programmers can implement it. This documentation should have at least four items for each file:

1. *File Description.* Prepare a one-page description summarizing the purpose of the file, the types of records it contains, what these records represent, how the records are indexed, and other such descriptive information.
2. *Record Layouts.* Prepare a record layout sheet (Figure 21-1) for each different record format used in the file.
3. *File Definition Sheet.* Prepare a file definition sheet (Figure 21-2) that specifies the way the transaction processor should access the file.

You may wish to code the record layouts yourself in the selected high-level programming language. The program fragments defining the record layouts can then be used by each application programmer as part of his programs. This saves programming time and promotes standard definitions for each record, as well as standard data names for each field in the record.

In addition to the documentation for each file, consider drawing a diagram or chart to show how the files are interrelated. Such a chart helps an application programmer, especially if access paths involve retrievals from a series of files. These linked access paths are hard to visualize.

RECORD LAYOUT SHEET

Transaction Processor: ☐☐☐☐☐☐

File Description: _____

Logical Filename: ☐☐☐☐☐☐

This is Record Format: ☐☐ of ☐☐

Logical Record Length: ☐☐☐☐

Physical Record Length: ☐☐☐☐ (Tape Only)

Field No.	Starting Byte	Length (Bytes)	Contents	Data Type
1				
2				
3				
4				
5				
6				
7				
8				
9				
10				
11				
12				
13				
14				
15				
16				
17				
18				
19				
20				
21				
22				
23				
24				
25				

Figure 21-1 Record Layout Sheet

FILE DEFINITION

Part One

Transaction Processor Name: ☐☐☐☐☐☐

Logical Filename: ☐☐☐☐☐☐

RMS File Specification: ☐☐☐:[☐☐☐.☐☐☐]☐☐☐☐☐☐.☐☐☐;☐☐

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

Is This an Indexed File?
- ☐ – Yes: No. of Keys ☐☐
  Maximum Key Length ☐☐☐
- ☐ – No: Sequential or Relative File

Maximum Concurrent File Accesses? ☐☐

Read-Only?
- ☐ – Yes
- ☐ – No

Fast Deletions?
- ☐ – Yes
- ☐ – No

Lock Interval ☐☐ 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: _____

Assigned I/O Channel Number ☐☐

Figure 21-2 File Definition Sheet

CHAPTER 22
A FILE DESIGN EXAMPLE

This chapter discusses the design of the customer file from the TRAX Sample Application and presents the set of file documents that would be given to the application programmer.

22.1 DESIGN CONSIDERATIONS

Functional requirements dictate that the customer file carry these data items:

Customer number
Customer name
Customer address (three 30-character lines)
Customer ZIP Code
Telephone number (including area code)
Attention line (20 characters for ATTN:)
Credit limit
Current balance
Purchases year-to-date
Next order sequence number
Next payment sequence number

The data is recorded as display or string variables, so that data can be exchanged between TSTs written in COBOL and BASIC-PLUS-2.

NOTE

Using different programming languages in the same application is normally avoided. Both languages are used in the Sample Application for demonstration purposes, hence the data format requirement.

No filler space is left in the record because no expansion of the number of fields is anticipated.

The file is indexed. The primary index is by customer number, and there is a secondary index by name.

There are no inter-field relationships from a record in this file to another file. However, other files in the Sample Application contain customer numbers creating logical relationships to records in this file.

None of the information in this file is amenable to coding. (The postal ZIP code is entered directly by the user and is not evaluated or interpreted by the system in any way.)

The file is shared by many users executing a variety of transactions. However, few users update customer records; most just read data from them. Significant access conflict is unlikely.

The first record in the file (customer number 000000) is a control record that contains the next available customer number, since customer numbers are assigned by the application. Although this record might become a bottleneck if large numbers of customers were added, a problem is not anticipated with the expected volume of customer additions.

Because of the low volume of changes to this file, it is not staged or journaled. In the case of a system failure, any lost work can be re-entered manually from existing documents.

22.2 DESIGN DOCUMENTATION

The following design documentation is the set of documents an application programmer needs to create and access the customer file in the TRAX Sample Application.

- File Description — Figure 22-1
- Record Layout — Figure 22-2
- File Definition Sheet — Figure 22-3

The customer file supplies data for individual customers. Each record in the file represents one customer. It is indexed by customer number (primary index) and customer name (secondary index).

There is only one record format used in the customer file. The record is 205 bytes long; no space is left for record expansion.

The first record in the customer file (customer number 000000) is a control record containing the next customer number. This number is contained in the Credit Limit Amount field of the record layout. When used in this manner, the Credit Limit field has no assumed decimal point.

The customer file is not staged or journaled.

Figure 22-1 Description of Customer File

RECORD LAYOUT SHEET

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

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

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

This is Record Format: [ ][1] of [ ][1]

Logical Record Length: [ ][2][0][5]

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 22-2 Record Layout Sheet

FILE DEFINITION

Part One

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

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

RMS File Specification: [S][Y][ ]:[[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][0]

[ ] – 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: One record for each customer

Assigned I/O Channel Number [ ][3]

Figure 22-3 File Definition Sheet for Customer File

CHAPTER 23

THE COMPLETE TRANSACTION PROCESSOR DOCUMENTATION

In Chapters 12 through 22, you designed a transaction processor.

This chapter summarizes the documentation needed by the application programmers who construct the transaction processor. This documentation is the basis for future system maintenance and enhancements.

23.1 THE TRANSACTION PROCESSOR DEFINITION SHEET

One final specification sheet, the transaction processor definition sheet, remains to be completed. It describes the overall characteristics of your application. Your application programmers will need this information to generate the transaction processor.

A blank transaction processor definition sheet is shown in Figure 23-1. This sheet asks for the following information:

1. *Transaction Processor Name.* Assign an abbreviated name (no more than six characters) to identify your transaction processor. This name must be unique from other transaction processor names on your system.
2. *Maximum Number of Transaction Types.* Supply the maximum number of different transactions to be defined and supported by this transaction processor. If you expect to add a number of transaction definitions in the future, include them in the number you specify.
3. *Maximum Number of Concurrent Transaction Instances.* Enter the number of transaction instances the transaction processor must handle simultaneously. Users attempting to begin transaction instances that exceed this number must wait until some other transaction instance finishes. Be sure to include transaction instances spawned by TSTs and initiated by batch or link stations, as well as those initiated by application terminals.
4. *Maximum Number of Application Terminals.* Enter the number of application terminals supported by the transaction processor. This number should agree with the number of terminal stations you have defined (or the number you expect in the future).
5. *Maximum Number of User TSTs.* Enter the number of different TSTs supported by this transaction processor. Again, if you expect new TSTs to be added soon, enter the projected number. Do not count any TST more than once, even if you will have multiple copies of a TST executing at once.
6. *Maximum Number of Master Link Stations.* Enter the number of master link stations that your transaction processor needs.
7. *Maximum Number of Slave Link Stations.* Enter the number of slave link stations your transaction processor needs to communicate with master link stations in other transaction processors.
8. *Maximum Size of Receive Link Message.* If you are including slave link stations, enter the largest link message that any slave link station can receive from a master link station in another transaction processor. Otherwise, leave this space blank.
9. *Maximum Number of Submit Batch Stations.* Enter "1" if your transaction processor submits work to the system's batch processors; enter "0" if it does not.
10. *Maximum Number of Slave Batch Stations.* Enter the number of slave batch stations the transaction processor needs to initiate transaction instances at the request of support environment programs.
11. *Maximum Number of Mailbox Stations.* Enter the number of different mailbox stations used by the transaction processor.
12. *Maximum Number of Application Data Files.* Enter the number of different application data files accessed by the transaction processor. This is the total number of different files, not the number accessed concurrently.

13. *Maximum Size of Transaction Slot.* This parameter is derived from three other parameters that you have calculated for each transaction:

    Exchange message size
    Transaction workspace size
    System workspace size

    To calculate the maximum size of the transaction slot, you must consider each transaction in turn. For each transaction, take the three parameters listed and divide each by 64. Round each result to the next higher integer. Then add the three rounded results. Do this for each transaction, and then pick the transaction with the greatest result. Enter this result on the transaction definition form.

14. *Automatic Crash Recovery.* Select whether or not your transaction processor includes automatic crash recovery.

Figure 23-2 shows a completed transaction processor definition sheet. This form represents a small transaction processor having only the four Customer file transactions discussed in this manual.

23.1.1 Transaction Documentation

For each transaction, you should have:

1. A transaction structure diagram
2. A transaction definition sheet

23.1.2 Form Documentation

You should have a form definition for each form used by the transaction processor. This includes entry forms, transaction selection forms, and report forms.

23.1.3 TST Documentation

For each TST, you should have:

1. A description of the TST purpose and a summary of its processing
2. A TST specification sheet
3. A layout sheet for the exchange message the TST will process
4. Layout sheets for response messages, report messages, and mailbox messages that the TST manipulates
5. A layout sheet for the transaction workspace that the TST uses
6. Layout sheets for the data file records that the TST uses
7. Perhaps, flowcharts or formulas that define complex processing

23.1.4 File Documentation

For each file, you should have:

1. A file description
2. Record layouts
3. A file definition sheet
4. RMS file parameters

23.1.5 Station Documentation

For each station, you should have completed a section of a station definition sheet.

23.1.6 Access Security Documentation

You should have completed sheets defining work classes, as well as sheets defining users and their access to work classes.

TRANSACTION PROCESSOR SPECIFICATION SHEET

Transaction Processor Definition: ☐☐☐☐☐☐

Transaction Processor Name: ☐☐☐☐☐☐

Maximum number of transaction types:	(0–64)	☐☐
Maximum number of concurrent transaction instances:	(0–64)	☐☐
Maximum number of application terminals:	(0–64)	☐☐☐
Maximum number of user TSTs	(0–256)	☐☐☐
Maximum number of master link stations:	(0–10)	☐☐
Maximum number of slave link stations:	(0–64)	☐☐
Maximum size of receive link message:	(0–4096)	☐☐☐☐
Maximum number of submit batch stations:	(0–1)	☐
Maximum number of slave batch stations:	(0–16)	☐☐☐
Maximum number of mailbox stations:	(0–10)	☐☐☐
Maximum number of application data files:	(0–64)	☐☐☐
Maximum transaction slot size:	(1–8192)	☐☐☐☐ blocks
Automatic crash recovery:	☐ – YES	☐ – NO

Figure 23-1 Blank Transaction Processor Specification Sheet

TRANSACTION PROCESSOR SPECIFICATION SHEET

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

Maximum number of transaction types:	(0–64)	[ ][4]
Maximum number of concurrent transaction instances:	(0–96)	[ ][4]
Maximum number of application terminals:	(0–256)	[ ][ ][4]
Maximum number of user TSTs	(0–256)	[ ][1][0]
Maximum number of master link stations:	(0–10)	[ ][0]
Maximum number of slave link stations:	(0–64)	[ ][0]
Maximum size of receive link message:	(0–4096)	[ ][ ][ ][0]
Maximum number of submit batch stations:	(0–1)	[0]
Maximum number of slave batch stations:	(0–16)	[ ][ ][0]
Maximum number of mailbox stations:	(0–10)	[ ][ ][0]
Maximum number of application data files:	(0–64)	[ ][ ][1]
Maximum transaction slot size:	(1–8192)	[ ][ ][ ][8] blocks
Automatic crash recovery:	[ ] – YES	[X] – NO

Figure 23-2 Completed Transaction Processor Specification Sheet

INDEX

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 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

Please cut along this line.

---------- 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 ML5-5/E39
Maynard, Massachusetts 01754