CONTENTS

CONTENTS (Cont.)

CONTENTS (Cont.)

Page

CONTENTS (Cont.)

Page

CONTENTS (Cont.)

Page

PREFACE

0.1 MANUAL OBJECTIVES

This manual is both a tutorial and a reference document. The concepts of forms processing in a TRAX application are presented, and a sample terminal session is illustrated to help the reader understand how forms are used.

The manual presents the facilities of the Application Terminal Language (ATL), and uses a series of coordinated examples to show how to define and code forms using ATL, and how to incorporate coded form definitions into a transaction processor. An extensive description of the ATL language is included.

0.2 INTENDED AUDIENCE

This document is written for use by the TRAX application designer and application programmer. It assumes no prior knowledge of ATL, but requires an understanding of the way an application is designed and integrated into a TRAX transaction processor.

0.3 STRUCTURE OF THIS DOCUMENT

This manual is divided into 5 chapters and 2 appendices. The following list gives a general description of each section:

1. Chapter 1 – Introduces forms processing and defines and describes a number of the concepts required by a forms designer. The VT62 terminal is described, and a sample TRAX application is shown in an illustrated example of how to use a form.
2. Chapter 2 – Describes the syntax of ATL, gives a description of each statement and clause type, and describes various rules to be followed when you are coding forms.
3. Chapter 3 – Is a tutorial chapter which describes how to code each of the three major types of TRAX forms, transaction selection, report, and data entry and display forms.
4. Chapter 4 – Describes how to use the ATL utility program to incorporate form definition source files into a transaction processor.
5. Chapter 5 – Is a detailed description of the Application Terminal Language. It is organized in alphabetical order by statement name, and describes the effect of each statement, and the effect of clauses as they pertain to specific ATL statements.
6. Appendix A – Lists the Error Messages produced by the ATL utility when it compiles form definitions.
7. Appendix B – Is a Table of ATL Statements and Clauses designed for quick reference.

0.4 ASSOCIATED DOCUMENTS

Before reading this document, you must read the Introduction to TRAX (DEC AA-D327A-TC). If you are not familiar with the program development facilities supported under TRAX, you should read the TRAX Support Environment User's Guide (DEC AA-D331A-TC), and the DEC Editor Reference Manual (DEC AA-D345A-TC). If you are designing the application, you must read the TRAX Application Designer's Guide (DEC AA-D328A-TC) prior to using this document.

0.5 CONVENTIONS USED IN THIS DOCUMENT

Throughout this manual, ATL is used to refer to the Application Terminal Language. This following conventions are used in examples:

CTRL/x — The CTRL key and another key pressed simultaneously (e.g., CTRL/Z)

(RET) — The RETURN key (carriage-return/line feed).

(nn) — The number in the circle on an example output listing refers to the similarly numbered paragraph in the annotation text.

Red text — Where examples contain both user input and computer output, the characters you type are in red; the characters the computer prints are in black.

The following symbols are used to annotate the ATL Language Descriptions:

[ ] — Special brackets indicate optional information can be omitted from a statement or clause.

{ } — Braces indicate a choice of one or more parameters from the set enclosed by the braces. You can also specify a parameter more than once as part of the same clause.

Lower-case letters — Parameters in lower-case letters indicate data that you must supply such as a label, number, or text-string.

UPPER-CASE LETTERS — Parameters shown in upper-case letters are keywords. They must be specified and spelled, as shown in the parameter list.

Underlining — The default parameter values assumed for certain clauses (ATTRIBUTES, for example) are underlined.

● ● ● — An ellipsis indicates that repetition of a clause or a parameter specification may occur at this position.

CHAPTER 1
TRAX FORMS PROCESSING

1.1 INTRODUCTION

TRAX is a forms-oriented transaction processing system. All communication between the application terminal user and the transaction processing software is done with *forms.*

TRAX displays a predefined form on the user terminal and allows you to fill it out. Until the user completes the form, TRAX is not involved in the data entry process. Only when the form has been completed does TRAX begin to process the data.

The interactive forms used in TRAX applications are specified by a system designer or application programmer using a set of statements, clauses, and parameters known as the Application Terminal Language (ATL). The ATL statements and clauses that describe a form are called a *form definition.*

In a TRAX system, the application terminals are controlled by system software modules, called *terminal stations,* that are part of each transaction processor. Terminal stations control terminal operations with a set of instructions generated from the form definitions that you have defined.

These form definitions specify the form displays that the user sees on the screen, and the restrictions that govern data entry. These specifications are interpreted and executed by microprocessors inside the application terminal. In this way, terminals are controlled without having an active application program.

1.2 A TYPICAL TRANSACTION

A typical session between a terminal user and the system consists of the following sequence of interactions.

1. On the initial form display, the user selects the desired transaction, and presses a function key which tells the terminal station that a transaction selection has been made.
2. The terminal station retrieves and displays the form defined for the first exchange of the specified transaction.
3. The user enters data in the form's fields, or edits data in those fields. During this process, he conforms to the data entry restrictions specified in the form definition.
4. After the form is completed, pressing a user function key sends the data to the terminal station.
5. The terminal station accepts the data it receives from the terminal and formats it according to the specifications in the form definition. The resulting data structure is called an *exchange message.*
6. The terminal station then passes the exchange message to a *transaction processor,* where one or more application programs perform data processing upon the input data.

7. After the transaction processor has completed processing, it can take one of two actions:
   - It can send information back to the terminal station that modifies the current form according to the specifications in the form definition (via a *reply response message*). Steps 2 through 7 are repeated using the current form.
   - It can send information to the terminal station causing the display of another form (using a *proceed response message*). Steps 2 through 7 are performed using the new form display.
8. If the user wants to return to select a new transaction, (Step 1), the transaction must be designed so that the selection form is displayed after the user has pressed a system function key to close the current transaction.

The forms displayed and completed at an interactive TRAX terminal replace the paper forms and punched cards that have traditionally been used to process business transactions.

As an example, consider a traditional catalog-store business where customer orders are called in over the telephone to an order processing center. An order taker receiving a call for an order must complete a paper form which is then sent to the data entry section for conversion to machine-readable media such as cards, tape, or disk.

In the same business using TRAX, the order taker enters the order information directly onto a form display at an interactive terminal. When the data entry operation is completed, a function key is pressed that sends the data on the form to the transaction processor, where system and application software combine to process the data.

In a TRAX-based application, paper forms and a separate data entry operation are no longer needed, since the forms defined for use with the microprocessor-based application terminal performs several crucial operations when the user enters the data:

- The data is collected in a machine-readable format.
- The terminal validates the data according to the specifications in the form definition.
- Data is sent to the terminal station as a package, rather than having every character transmitted and echoed by the main system.

1.3 DEFINING FORMS

When the application designer has determined the layout of the forms used by a business, the ATL statements and clauses are used to code the set of source statements that specify a form definition.

The form definition specifies the appearance of the form and the placement of fields and captions. It also specifies:

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

Once they are coded, the ATL statements detailing the form definition are entered into a machine-readable file using the DEC Editor.

The file of ATL source statements is then submitted to the *ATL utility*. The ATL utility is a forms compiler. It converts the form definition source statements into a *form definition record* that contains screen formats and terminal commands that govern the appearance and operation of an application terminal.

1.4 TRANSACTION DEFINITION

A *transaction definition* sets the structure of each transaction that a transaction processor can perform. The transaction definition specifies parameters such as:

- The number of exchanges in the transaction
- The form definition for each exchange
- The processing route for each exchange message
- The execution sequence for exchanges

A transaction can have one or more exchanges. For example, the transaction in Figure 1-1 adds a new record to a file. It requires one exchange.

On the other hand, a transaction that modifies information in a data file might require two exchanges: one exchange to specify the record, and a second exchange to display the requested data, to allow user modifications, and then to process the changed data. A two-exchange transaction is described in Figure 1-2.

1.5 DATA FLOW IN A TRANSACTION

The single exchange transaction in Figure 1-1 (following page) illustrates the flow of data between forms and the transaction processor.

- The exchange begins when the transaction processor displays the form specified for the exchange, in this example, it is the exchange to add a record to the customer file.
- Then the user completes the input fields on the form display and presses a user function key.
- When the user function key is pressed, the data entered on the screen is sent from the terminal to the transaction processor, where it is formatted into a data structure called an exchange message.
- The transaction processor then passes the exchange message to the exchange routing list, which consists of two application programs called TSTs. The first TST validates the exchange message and passes control to the second TST.
- As the second TST completes processing, it passes the results back to the terminal station using a TRAX system call known as a reply response message.
- The terminal station uses the reply response message and the reply specifications in the form definition to write the customer number on the screen along with the message **TRANSACTION COMPLETE**.
- The exchange is completed, and the user is instructed to press the AFFIRM function key to continue adding customers to the file.

1.6 TRAX APPLICATION PROGRAMS – TSTS

A *Transaction Step Task (TST)* is a TRAX application program.

When the terminal station receives data from an application terminal, it formats the exchange message as specified in the form definition. Then, it invokes one or more TSTs to process the exchange message. To learn more about coding and using TSTs, refer to the *TRAX Application Programmer's Guide* (AA-D329A-TC).

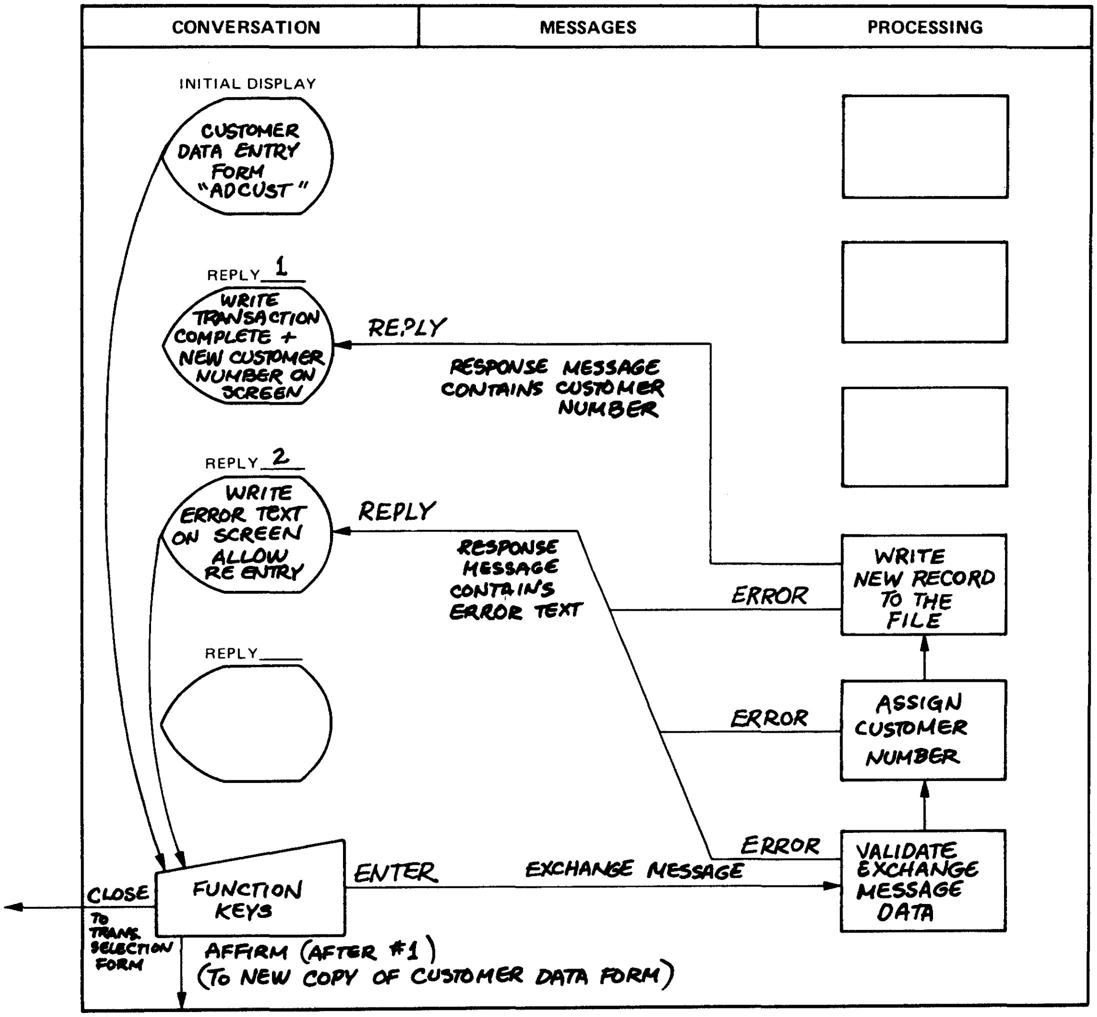
TRANSACTION PROCESSOR SAMPLE

TRANSACTION NAME ADDCUS

EXCHANGE NAME ADDEX1

FORM NAME ADCUST

PAGE 1 OF 1


CONVERSATION
MESSAGES
PROCESSING
INITIAL DISPLAY
CUSTOMER DATA ENTRY FORM "ADCUST"
REPLY 1
WRITE TRANSACTION COMPLETE + NEW CUSTOMER NUMBER ON SCREEN
REPLY
RESPONSE MESSAGE CONTAINS CUSTOMER NUMBER
REPLY 2
WRITE ERROR TEXT ON SCREEN ALLOW RE ENTRY
REPLY
RESPONSE MESSAGE CONTAINS ERROR TEXT
REPLY
WRITE NEW RECORD TO THE FILE
ERROR
ASSIGN CUSTOMER NUMBER
ERROR
ERROR
VALIDATE EXCHANGE MESSAGE DATA
FUNCTION KEYS
ENTER
EXCHANGE MESSAGE
CLOSE
TO TRANS. SELECTION FORM
AFFIRM (AFTER #1)
(TO NEW COPY OF CUSTOMER DATA FORM)

AT END:

□ – REPEAT

☒ – NOREPEAT

□ – NEXT

☒ – FIRST

□ – INITIAL

☒ – WAIT

□ – NOWAIT

Figure 1-1 A Transaction Structure Diagram

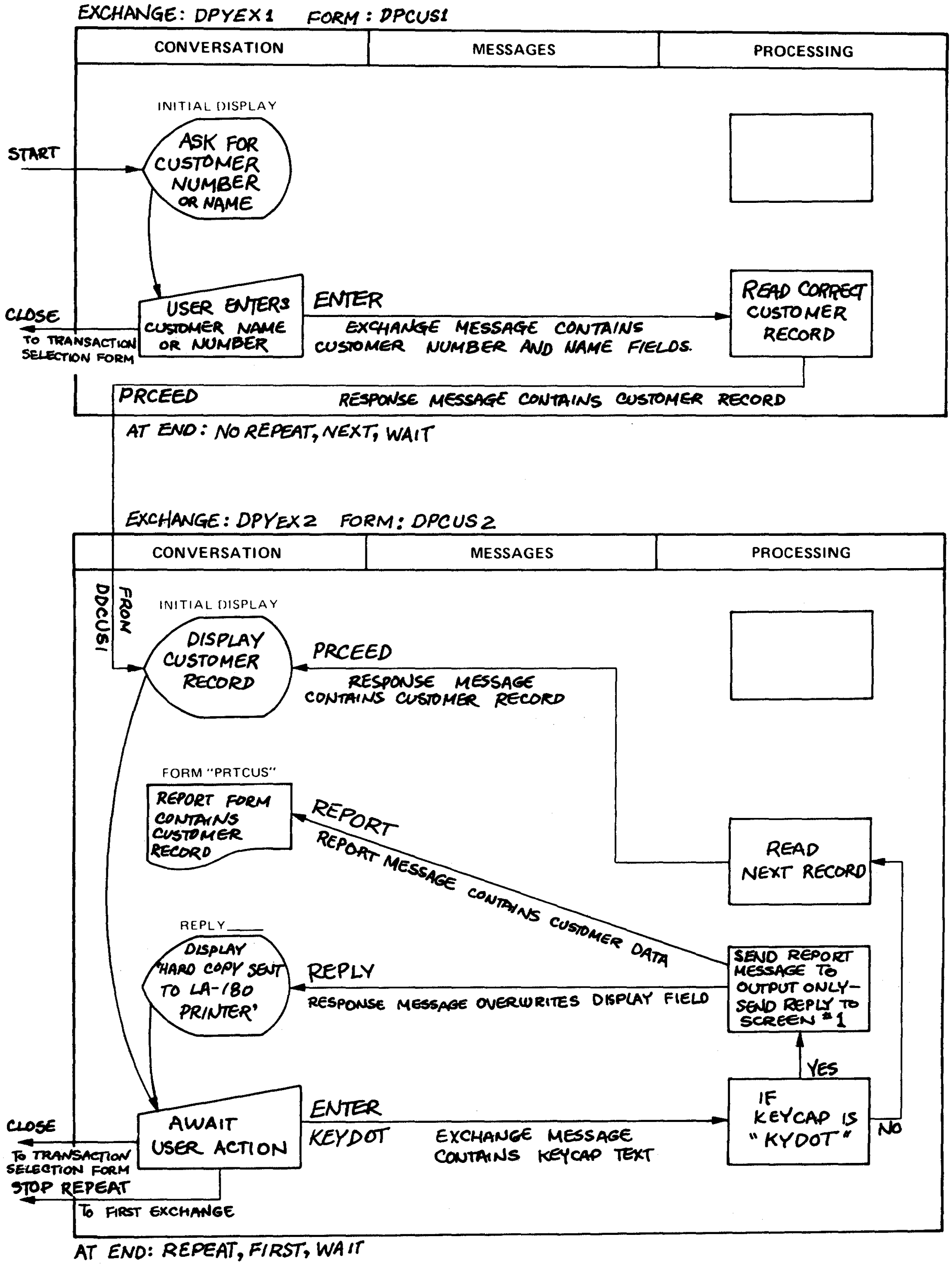

EXCHANGE: DPYEX1 FORM: DPCUS1
CONVERSATION
MESSAGES
PROCESSING
INITIAL DISPLAY
START
ASK FOR CUSTOMER NUMBER OR NAME
CLOSE
TO TRANSACTION SELECTION FORM
USER ENTERS CUSTOMER NAME OR NUMBER
ENTER
EXCHANGE MESSAGE CONTAINS CUSTOMER NUMBER AND NAME FIELDS.
READ CORRECT CUSTOMER RECORD
PRCEED
RESPONSE MESSAGE CONTAINS CUSTOMER RECORD
AT END: NO REPEAT, NEXT, WAIT
EXCHANGE: DPYEX2 FORM: DPCUS2
CONVERSATION
MESSAGES
PROCESSING
FROM DDCUS1
INITIAL DISPLAY
DISPLAY CUSTOMER RECORD
PRCEED
RESPONSE MESSAGE CONTAINS CUSTOMER RECORD
FORM "PRTCUS"
REPORT FORM CONTAINS CUSTOMER RECORD
REPORT
REPORT MESSAGE CONTAINS CUSTOMER DATA
READ NEXT RECORD
REPLY
DISPLAY 'HARD COPY SENT TO LA-180 PRINTER'
REPLY
RESPONSE MESSAGE OVERWRITES DISPLAY FIELD
SEND REPORT MESSAGE TO OUTPUT ONLY-SEND REPLY TO SCREEN #1
YES
CLOSE
TO TRANSACTION SELECTION FORM
AWAIT USER ACTION
ENTER
KEYDOT
EXCHANGE MESSAGE CONTAINS KEYCAP TEXT
IF KEYCAP IS "KYDOT"
NO
STOP REPEAT
TO FIRST EXCHANGE
AT END: REPEAT, FIRST, WAIT

Figure 1-2 A Two-exchange Transaction

1.7 FORMS AND FORM DEFINITIONS

A form is a terminal screen display that contains instructions, information, and space for entering and/or displaying data.

TRAX applications use three types of forms. Each type of form serves a specific purpose and must be coded using ATL language elements. Once the form definitions are processed by the ATL utility, the resulting object code is stored in the transaction processor's form definition file.

The three types of forms are:

1. *Transaction Selection Forms* are used to select a transaction and initiate it at the terminal. When a terminal station is defined using the STADEF utility (see the *TRAX Application Programmer's Guide* AA-D329A-TC.), the name of an initial form is specified for the station. This initial form is automatically displayed on the terminal when:
   - A transaction processor is startea.
   - A transaction instance terminates, and its subsequent action parameter specifies that the initial form is to be displayed.

   In a typical TRAX application, the user chooses the type of transaction to be processed and specifies the name of that transaction from a list of defined transaction types. The initial form defined for an application terminal is normally a transaction selected form.
2. *Entry . . . Forms,* the most commonly used forms in a TRAX application, allow the user to enter data onto a terminal screen directly, as well as to retrieve and display information from a data base. When you define an exchange, you usually specify the name of an entry form as part of the exchange definition.
3. *Report Forms* are used to format information for printing on an output-only terminal device. You specify a form name as part of the library call that sends a report message to an output-only station. These forms are never interactive; they are always invoked by a TST and sent to a hard-copy terminal.

Once the designer specifies a form, and a set of ATL statements is coded to implement the specification, the DEC Editor is used to enter the ATL language source statements into an ATL source statement file. The source statements are compiled, using the TRAX ATL utility program, into a set of screen formatting commands that collectively are called a form definition record. This form definition record is stored in the form definition file, along with all other form definitions for the transaction processor.

In a set of ATL form definition source statements, you specify:

1. The form layout as it appears on the application terminal. Forms are laid out in two major areas, *the Display area* and *the Form area.* The top of the screen is the Display area, the remainder is the Form area. Within each area, you can define certain types of fields.

   A field is a defined contiguous section of the screen that has attributes and characteristics specified by the form definition. There are five types of fields:

   A. *Display Fields* present user instructions and display messages from the transaction processor. Display fields appear in the Display area.

   B. *Menu Fields* list items that can be selected by the user and sent directly to the transaction processor. A Menu field appears in a form's Display area.

C. *Prompt Fields* provide instructions to the user filling the form. These prompts appear in the Form area.
D. *Input Fields* provide space in the form where the user can enter new data or edit existing information. Input fields appear in the Forms area.
E. *Print fields,* similar to prompt fields, are used to display instructions and data on forms specified for an output-only terminal. Print fields are specified on Report forms only.

2. The data entry restrictions that terminal users must observe. These include field content and justification attributes (for example, allowing only numeric characters in an Input field).
3. The field dimensions and their initial contents.
4. How the data in a response message is used to initialize field contents on a form display.
5. Format specifications to construct exchange messages from user input.
6. *Function keys* enabled for use at a particular time.
7. Sets of form modifications (*replies*) applied when reply messages are received from an application TST.

1.7.1 Replies

When the TST has processed the data from an exchange, the transaction processor sends a new visual display to the terminal. This may be a new form, a fresh copy of the current form, or a set of modifications, called a reply, to the current form.

In the case of a reply, the transaction processor acts upon the current form in one or more of the following ways:

1. It displays new instructions, text, or data field contents.
2. It enables a new set of user and system function keys.
3. It causes the terminal bell to sound, alerting the user.

Replies are specified by the REPLY statement in the ATL form definition. Each form can have up to 64 numbered REPLYs specified in the form definition.

1.8 THE TERMINAL STATION

A terminal station is a logical location in a transaction processor that controls the flow of data to and from an application terminal. You may think of it as the interface between forms and TSTs. The terminal station extracts information from data entered by the terminal user, and formats that information to create exchange messages. It also merges data from response messages into form definition records, creating screen displays that are returned to the application terminal.

1.9 MESSAGES

Messages are the structures used to transfer data between a TST and the terminal. Exchange messages carry information from the terminal to the application program. Response messages carry information from the application program to the terminal.

1.9.1 Exchange Messages

When a user function key is pressed, the data contained in a form's Input fields is sent from the terminal to the terminal station. At the terminal station, the transaction processor formats this data into an exchange message. The formatting is specified by the parameters contained in the MESSAGE statement in that form's ATL form definition.

The exchange message contains one or more of the following types of data:

1. Data entered by the user into a form's Input fields
2. System-supplied information, such as the current date and time
3. The contents of a menu field selected by the user
4. A text-string identifying the user function key that sent the data to the terminal station.

After the terminal station formats the exchange message, it is made available to each of the TSTs that do the processing for the exchange.

1.9.2 Response Messages

Response messages are the TSTs answer to exchange messages. Response messages can cause modifications to the current form, or they can display another form on the application terminal. Both classes of response messages can optionally include data for display on the form. A response message is sent to the terminal station where the exchange message originated.

One type of response message is a "reply response message".

An important distinction to note in TRAX applications is the difference between a reply and a "reply response message". A reply is the set of modifications, specified in a form definition, that the terminal station makes to a screen display when a "reply response message" is sent by a TST to the form.

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

- The text display in any field
- The set of enabled function keys
- The cursor position
- Sounding the terminal bell upon display of a reply

Other form aspects are "frozen" when a form is first displayed. These features cannot be changed by a reply:

- A field's size
- A field's data entry restrictions
- A field's display mode (such as, black on white)
- The format of the exchange message.

When a TST sends a reply response message to a terminal station, it specifies a reply number in the response message parameter list. The transaction processor matches that reply number against the numbered replies stored in the form definition record. The corresponding set of screen modifications is then made to the form on the application terminal.

Replies are commonly used to tell the terminal user of a completed action or an error discovered during the processing of a transaction. A reply allows the terminal user to continue in the same exchange without forcing the transaction processor to refresh the form.

For example, if a TST detects an error in the input data or encounters a system error during processing, it sends a reply response message, specifying a reply number in the parameter list. After the reply screen is received at the terminal, the user can study the modified form and then edit the contents of the form's Input fields. The process of entering a form, receiving a reply, and reentering data can be repeated any number of times before proceeding to the next exchange specified by the transaction definition.

Other types of response messages transfer the processing to another exchange. Depending on the type of response message, the new exchange (and the associated new form) may be the next exchange, the first exchange, or some other exchange defined for the current transaction type.

The various types of response messages are described in the *TRAX Application Designer's Guide* (AA-D328A-TC), and the way they are sent from TSTs is described in the *TRAX Application Programmer's Guide* (AA-D329A-TC).

For a transaction to display data on a terminal, two distinct programming efforts are involved:

1. A TST issuing a response message system call must include a parameter specifying the data to be included in the response message.
2. In the form definition, you must specify how the terminal station is to display the data included in the requesting response message. Using the ATL REQUEST function, you can specify the way that data is extracted from a response message and placed into Display, Menu, Print, Prompt, and Input fields.

1.10 THE APPLICATION TERMINAL

TRAX supports two types of application terminals: the *VT62,* a cathode ray tube (CRT) terminal which is used to process interactive transactions; and the *LA180,* an output-only hard-copy terminal used for printing reports.

1.10.1 Using the VT62 Application Terminal

This section explains how to use the interactive TRAX application terminal, the VT62. The VT62 is a "smart" video terminal that was designed expressly for use with TRAX. The VT62 keyboard is shown in Figure 1-3. The VT62 has a display screen which consists of 23 programmer-defined and user-accessible display lines, with a 24th line used to display system and error messages. The keyboard is a standard typewriter keyboard, with a number of control and function keys added to provide the user with increased data entry capability. The right-hand keypad provides a full set of numeric data entry keys. In addition, it contains four system function keys, and utilizes the SHIFT key to provide six user function keys.

The VT62 improves overall performance by checking input data for a number of attributes and conditions at the time the data is typed, rather than having to send the data to a verification program before processing can begin. If a user types an invalid character, the terminal locks the keyboard, sounds the bell and displays an error message on line 24 of the terminal screen. When a form is completed to the user's satisfaction, it can be transmitted to the terminal station by pressing an enabled user function key. When a transaction is completed, the user may determine the next terminal screen display by pressing an enabled system function key.

The VT62 also allows fields to be highlighted in reverse video. Normal video characters are seen as light characters on a dark background. Reverse video, as the name implies, displays characters as dark symbols on a light background.

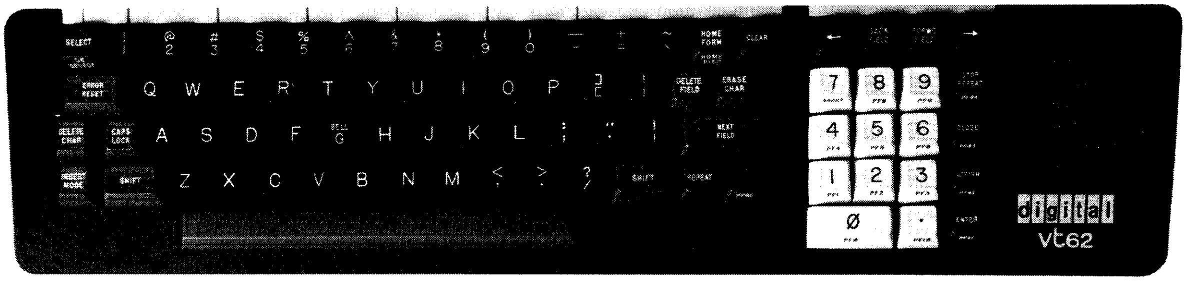


Figure 1-3 Terminal Keyboard

1.10.2 The CURSOR

The VT62 has one visible cursor. The cursor flashes its current position by turning that character position from regular to reverse video and back again. If you have defined any Menu fields on a form, you can use the NEXT FIELD, FORWD FIELD, BACK FIELD key to position the cursor in any Menu field in the display area. If you have defined any Input fields, you may use these same keys as well as the right arrow (→) and left arrow (←) keys to position the cursor at any location in an Input field. You may not position the cursor in an Input field with the NOMODIFY attribute.

1.10.3 The VT62 Terminal Keyboard

The application terminal keyboard has a number of specialized keys that assist the terminal user. The keys used to control keyboard operations are listed below, along with a description of the action that each key performs.

NEXT FIELD

When you press the NEXT FIELD key, the cursor moves from its current location to the beginning of the next defined Input or Menu field. If the current location is at the beginning or in the middle of an Input field, when the terminal fills the remaining characters in that field with the CLEAR character defined for that field. Consider the following fields:

Maynard, MA 03060

In this example, the cursor is positioned over the "y" in "Maynard". If you press the NEXT FIELD key, the letters to the right of the "y" are cleared (the CLEAR character specified for this field is a space), and the cursor is positioned at the start of the next field, the zip code.

May 03060

You use the NEXT FIELD key most often in initial data entry operations, where you want to blank the remainder of the field after entering the required data. In cases where the field initially contains a value, you can use the FORWD FIELD and BACK FIELD keys to move from one field to another without altering the field contents.

FORWD FIELD

When you press the FORWD FIELD key, the cursor moves to the first position of the next Input or Menu field without altering the contents of the current field. If you continue to press this key, it repeats the same function at the rate of ten times per second. This key is very useful for editing existing information on a form, since it allows you to skip over correct fields without altering the contents.

BACK FIELD
If you press the BACK FIELD key when the cursor is in the middle of an Input field, the cursor will return to the beginning of the current field. If the BACK FIELD key is pressed when the cursor is at the beginning of an Input or Menu field, the cursor will move to the beginning of the preceding field. The BACK FIELD key never alters the contents of a field, and is useful for skipping backward to incorrect fields. If you continue to press this key, it repeats the same function at the rate of ten times per second.

HOME FORM
When you press the HOME FORM key, the cursor moves to the start of the first Input field defined on the form.

HOME DISPLAY
If you press the HOME DISPLAY key (Shift + HOME FORM), the cursor moves to the start of the first Menu field defined on the form.

SELECT
The SELECT key is enabled only when a form has been defined with Menu fields. When you position the cursor in a Menu field and then press SELECT, the terminal displays the selected field in reverse video. If, after selecting a menu item, you press an enabled user function key (ENTER, KEYDOT, KEY0, KEY1, KEY2, KEY3), the terminal sends the contents of the selected Menu field to the terminal station.

DESELECT
If you find that a selected Menu field is incorrect before you transmit the selected field, you can deselect the menu item by pressing the DESELECT key. (To use the DESELECT key you must simultaneously press the SHIFT key and the SELECT key.) The terminal will return the field to normal video, and you may then begin selecting another menu item.

CLEAR
In the case of Menu fields, the CLEAR key functions as a global DESELECT key. Regardless of where the cursor is positioned, all selected fields are restored to normal video and deselected. The cursor returns to the first character of the first menu item.

If the form has Input fields, first pressing the CLEAR key erases all Input field contents, filling them with their defined CLEAR character(s). The terminal then returns the cursor to the start of the first Input field defined for that form.

ERASE CHAR
The ERASE CHAR key allows you to erase characters selectively from an Input field. If the field has been defined as left-justified (the default), pressing the ERASE CHAR key erases the character immediately to the left of the cursor. If you continue to press this key, it will repeat the same function at the rate of ten times per second.

The cursor then moves left one position, and a CLEAR character is inserted in the rightmost position in the field. If the character is in the leftmost position of a field, pressing

ERASE CHAR deletes that character and moves all other characters in the field one position to the left. Consider the following example:

Maaynard, MA

Initially, the cursor is on the "y".

Pressing, the ERASE CHAR key causes the following string to be displayed:

Maynard, MA

Notice that the second "a" has been erased, and the cursor is still on the "y".

The action of the ERASE CHAR key is similar for right-justified fields:

1,2005.00

Initially, the cursor is the "5".

Pressing the ERASE CHAR key causes the following string to be displayed:

1,205.00

In this case, the "0" to the left of the "5" was erased. The cursor is still on the "5".

DELETE FIELD
When you press the DELETE FIELD key, the terminal erases the contents of the current field and fills it with the defined CLEAR character for that field. The cursor is returned to the start of the field.

DELETE CHAR
Pressing the DELETE CHAR key deletes the character under the cursor. This key repeats its function at the rate of ten times per second. For left-justified fields, all characters to the right of the deleted character are shifted left one position. The following examples illustrate the action of this key:

Maaynard, MA

Initially, the cursor is on the first "a".

Pressing the DELETE CHAR key causes the following string to be displayed:

Maynard, MA

The first "a" has been deleted. The cursor is on the remaining "a".

The action of the DELETE CHAR key is similar for right-justified fields:

1,2005.00

Initially, the cursor is on the "5".

Pressing the DELETE CHAR key causes the following string to be displayed:

1,200.00

The "5" has been deleted, the cursor is now on the decimal point (.).

INSERT MODE
Pressing the INSERT MODE key allows you to insert characters into a previously filled Input field. If you type data into a left-justified field while in INSERT MODE, all characters to the right of the cursor move one position to the right. When typing into right-justified fields, all characters to the left of the cursor move one position to the left. Consider the examples below:

SMTH

Initially, the cursor is on the "T"
you press the INSERT MODE key, then type "I", the following string is displayed:

SMITH

The "I" has been inserted, and the cursor is still on the "T". To exit from insert mode, press the SHIFT key and the INSERT MODE key simultaneously.

Right-justified fields can be modified in the same way:

123.45

Initially, the cursor is on the decimal point (.).
you press the INSERT MODE key then type "0", the following string results:

1230.45

The "0" has been inserted to the left of the decimal point, and the cursor is still on the decimal point (.). Again, to exit from insert mode, press the SHIFT and INSERT MODE keys simultaneously.

You can determine if the keyboard is in insert mode by looking at the line of red lights on the extreme right of the keyboard. If the light labelled INSERT MODE is lit, you must press SHIFT and INSERT MODE to return the keyboard to its normal state.

Typing beyond an Input field boundary while in insert mode causes the keyboard to lock and the terminal bell to ring.

← (CURSOR LEFT)
Pressing the ← key moves the cursor one position to the left in an Input field. If you are at the left-most character position of a field, the cursor does not move. This key is a repeating key.

→ (CURSOR RIGHT)
Pressing the → key moves the cursor one position to the right in an Input field. If you are at the right-most character position of a field, the cursor does not move. This key is a repeating key.

ERROR RESET
When it detects a typing error or an attempt to enter improper data, the terminal writes a brief error message on the last line of the terminal screen, rings the bell, and locks the keyboard. When you acknowledge the error by pressing the ERROR RESET key, the keyboard is unlocked, and you can make corrections and resubmit the data.

1.10.4 System Function Keys

The VT62 has four system function keys that allow you to modify the flow of processing from the application terminal. Proper use of these function keys allows you to choose a number of different actions once an exchange is completed. TRAX supports the following system function keys on the terminal keypad:

STOPREPEAT
The STOP REPEAT key causes any exchange to leave that exchange and go to the exchange specified in the subsequent action section of the transaction definition. You must enable the STOP REPEAT key in the form definition before it can be used to control the actions of a transaction from an application terminal. This key is used to override the repeat exchange specification in the exchange definition.

CLOSE
Pressing the CLOSE key ends the transaction instance, and redisplays the initial form on the terminal screen. You must enable the CLOSE key in the form definition before it can be used to close a transaction from an application terminal.

AFFIRM
Pressing the AFFIRM key returns the application terminal to the form specified by the subsequent action section of the current transaction definition. You must enable the AFFIRM key in the form definition before it can be used to transfer control in a transaction instance. Pressing AFFIRM in a repeated exchange causes the form for that exchange to be redisplayed with cleared Input fields.

ABORT
To use the ABORT key, you must simultaneously press the SHIFT key and the 7 key (labeled as ABORT) on the function keypad. When you use the ABORT key, the transaction processor aborts the transaction instance, and returns the terminal to its initial screen display. You cannot disable the ABORT key.

1.10.5 User Function Keys

The VT62 has six user function keys on the terminal keypad. The effect of any of these keys is identical: the form's Input fields and any selected menu items are transmitted to the terminal station. The terminal station formats an exchange message for subsequent application processing. The advantage of having multiple user function keys is that transaction can be designed so that different processing paths are followed depending on which data entry function key is pressed.

For example, pressing the KEYDOT might cause a hard-copy report to be produced, while pressing ENTER might cause the next record to be displayed.

You differentiate processing modes by specifying the VALUE = KEY clause in the MESSAGE statement; this causes the appropriate keycap text-string value to be inserted in the exchange message and passed to the TST, where a simple test can be made to determine which key caused this message to be sent from the terminal to the TST.

Default keycap text-string values may be overridden by using the KEYCAP clause in the FORM statement.

The user function keys that can be enabled for use at an application terminal are:

ENTER
The ENTER key is the standard user function key. The ENTER key is assumed to be enabled unless you disabled it as part of a form definition. Pressing ENTER or any other user function key causes the contents of a form's Input fields and selected menu items (if any) to be transmitted. The key text value for the ENTER key is "ENTER".

KEYDOT
KEYDOT is the SHIFT key and the DOT key (.) on the function keypad. The default keycap text value is "KEYDT".

KEY0
KEY0 consists of the SHIFT key and the 0 key on the function keypad. The default keycap text value is "KEY00".

KEY1
KEY1 consists of the SHIFT key and the 1 key on the function keypad. The default keycap text value is "KEY01".

KEY2
KEY2 consists of the SHIFT key and the 2 key on the function keypad. The default keycap text value is "KEY02".

KEY3
KEY3 consists of the SHIFT key and the 3 key on the function keypad. The default keycap text value is "KEY03".

1.10.6 The VT62 LED Display Panel

The VT62 is equipped with a set of 8 Light Emitting Diodes (referred to in this section as LEDs or lights) which are located on the extreme right of the keyboard. These LEDs provide status information to the terminal user. The following list gives the LED legends, and explains the significance of the LED in both lighted and unlighted modes.

READY – When the READY light is on, it indicates that the terminal is enabled for user input. If the READY light is not on, the user cannot type on the terminal screen.

INSERT MODE – When the user invokes INSERT MODE, this light goes on. It remains on until INSERT MODE is terminated by the user, or until the NEXT FIELD, FORWD FIELD, or BACK FIELD keys are pressed.

KEYBOARD LOCKED – When the KEYBOARD LOCKED light is on, no input is possible. The KEYBOARD LOCKED light is turned on by a keying error, or when a user or system function key is pressed. The keyboard locked light is turned off by pressing ERROR RESET in the case of a keying error, or when a REPLY or a new form is received by the terminal from a transaction processor.

DISPLAY AREA – This light goes on when the cursor is positioned in a Menu field in the terminal's display area. In general, only the keys used to move through fields or from field to field are enabled when the DISPLAY AREA light is on. No data may be entered on the form when this light is on.

KEYING ERROR – When the user types an illegal character or attempts a prohibited operation from the terminal keyboard (such as skipping a required field), this light goes on and the keyboard is locked. The keying error light is turned off and the keyboard unlocked when the user presses ERROR RESET.

CLEAR TO SEND – If the TRAX transaction processor is polling the application terminal, this light will go on during the poll and turn off once the polling operation is completed. If no activity is seen in this light, the system manager should be contacted to determine why the poll is lost.

FUNCTION KEYPAD – The VT62 can be set to a mode where the keys on the numeric keypad are directly enabled as function keys, and the user does not have to press the SHIFT key concurrently with a user function key. If this mode has been enabled, the FUNCTION KEYPAD light is on.

CARRIER – When the carrier light is on, it indicates that a physical line connection exists between the terminal and the computer.

1.11 A SAMPLE TERMINAL SESSION – THE USER'S PERSPECTIVE

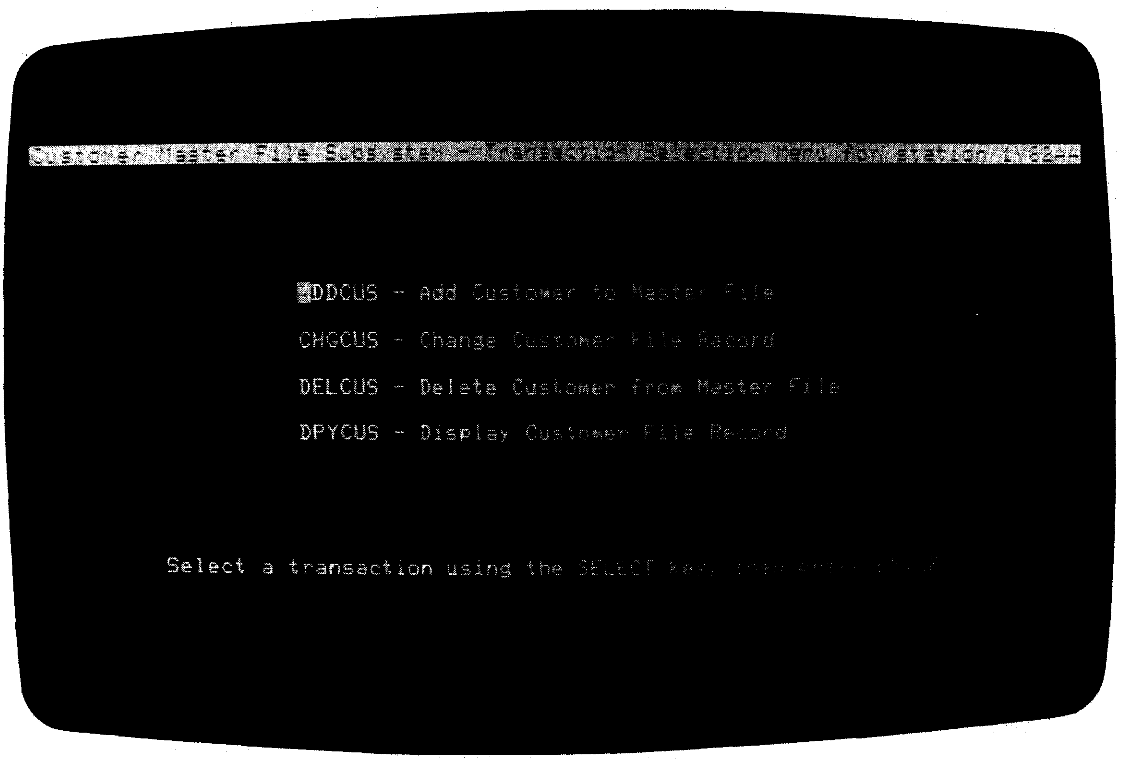
The examples in this chapter are based upon four transactions from the TRAX sample application. These transactions include:

1. Adding a customer record to a file
2. Changing customer records on a file
3. Deleting customer records from a file
4. Deleting customers from a file

1.11.1 Using a Transaction Selection Form

To invoke a transaction from an application terminal, you use a special type of form, known as a transaction selection form. This form lists the names of transactions defined in a transaction processor, and allows you to select or specify one of the named transactions. When a transaction processor is installed and running, the application terminals assigned to that transaction processor display an initial form any time that a transaction instance is not active on that terminal. In most transaction processing applications, a terminal will have a transaction selection form defined as its initial form.

If the sample transactions were placed on a transaction selection form defined as the initial form for your terminal, you might see the following initial display on the terminal screen:



```
Customer Master File Subsystem - Transaction Selection Menu for station 1/62--

                    ADDCUS - Add Customer to Master File

                    CHGCUS - Change Customer File Record

                    DELCUS - Delete Customer from Master File

                    DPYCUS - Display Customer File Record



           Select a transaction using the SELECT key, then press ENTER
```

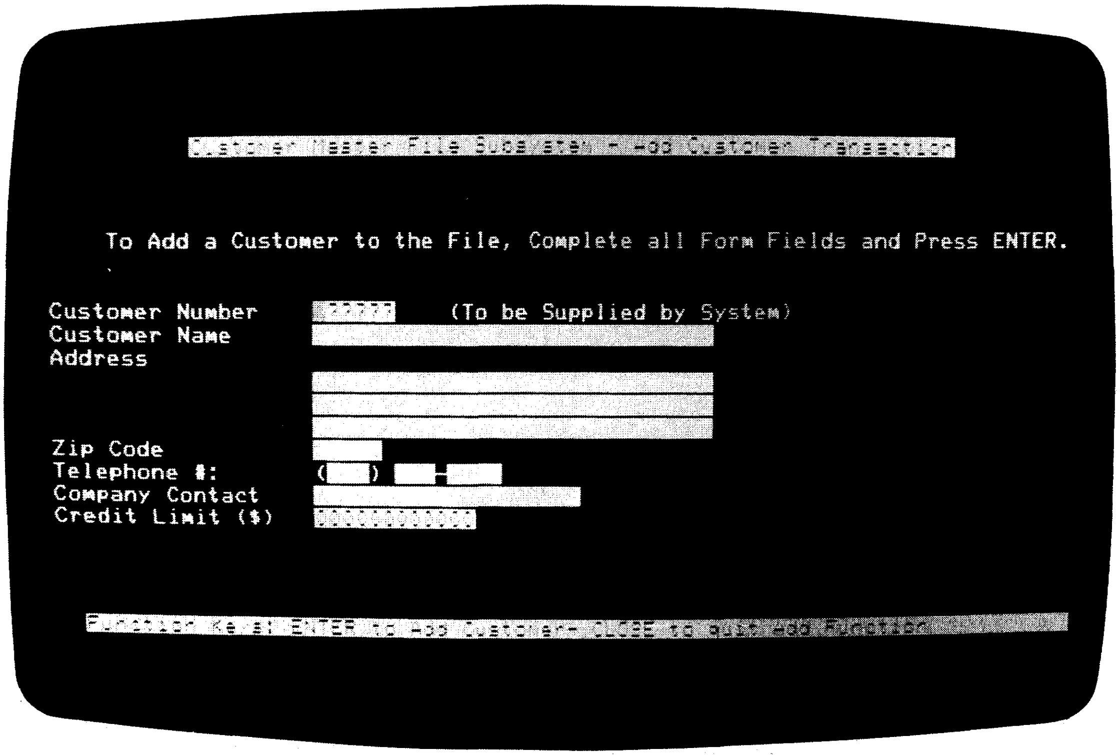
Figure 1-4 A Transaction Selection Form

The screen display in Figure 1-4 is a transaction selection form that allows you to choose a transaction from a menu. If you want to add a customer record to a file, you must first select the ADDCUS transaction from the menu, and then invoke it.

To select a transaction, move the cursor (using the NEXT FIELD key) to the desired transaction name, and press the SELECT key. In the case of the form SELECT, ADDCUS is the first menu item, and the cursor is already there, so all you have to do to select ADDCUS is press the SELECT key. To initiate the ADDCUS transaction, press the ENTER key after you have selected ADDCUS. Pressing ENTER causes the menu selection to be transmitted to the transaction processing executive, which invokes the ADDCUS transaction and initiates it at your application terminal. When ADDCUS is initiated, a copy of the first form defined for that transaction is displayed on your terminal screen. The first (and only) form of the ADDCUS transaction is shown in Figure 1-5. This form is the entry form used to gather the data required to add a customer record to the customer file.

1.11.2 Using Entry Forms

The entry form for the ADDCUS transaction contains a number of fields; some display instructions, while others are actual data entry fields (highlighted in reverse video). You begin typing data in the



```
       Customer Master File Subsystem - Add Customer Transaction

 To Add a Customer to the File, Complete all Form Fields and Press ENTER.

Customer Number     ?????     (To be Supplied by System)
Customer Name
Address



Zip Code
Telephone #:        (   )    -
Company Contact
Credit Limit ($)

Function Keys: ENTER to Add Customer- CLOSE to Quit Add Function
```

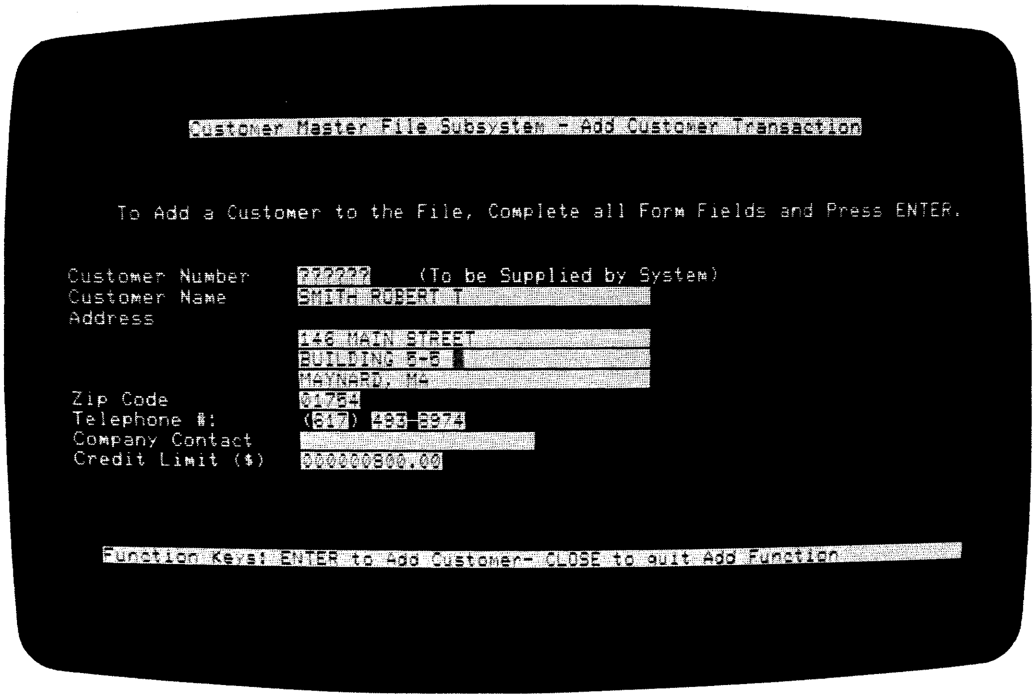
Figure 1-5 Entry Form – The Initial Screen Display

customer name field, and complete all Input fields as they appear on the form. Once you enter the data required to complete an Input field, press the NEXT FIELD key. NEXT FIELD fills the remaining character positions (if any) in the field with the clear character for that field, and moves the cursor to the first position of the next Input field. When you type the telephone number on the form, however, you may type all ten characters as if they were a single field. This is possible since the first two parts of the telephone number were defined with the ATL TAB attribute, causing the cursor to advance automatically when the field is full.

When you complete the data entry operation, the entry form will look like the screen display shown in Figure 1-6.

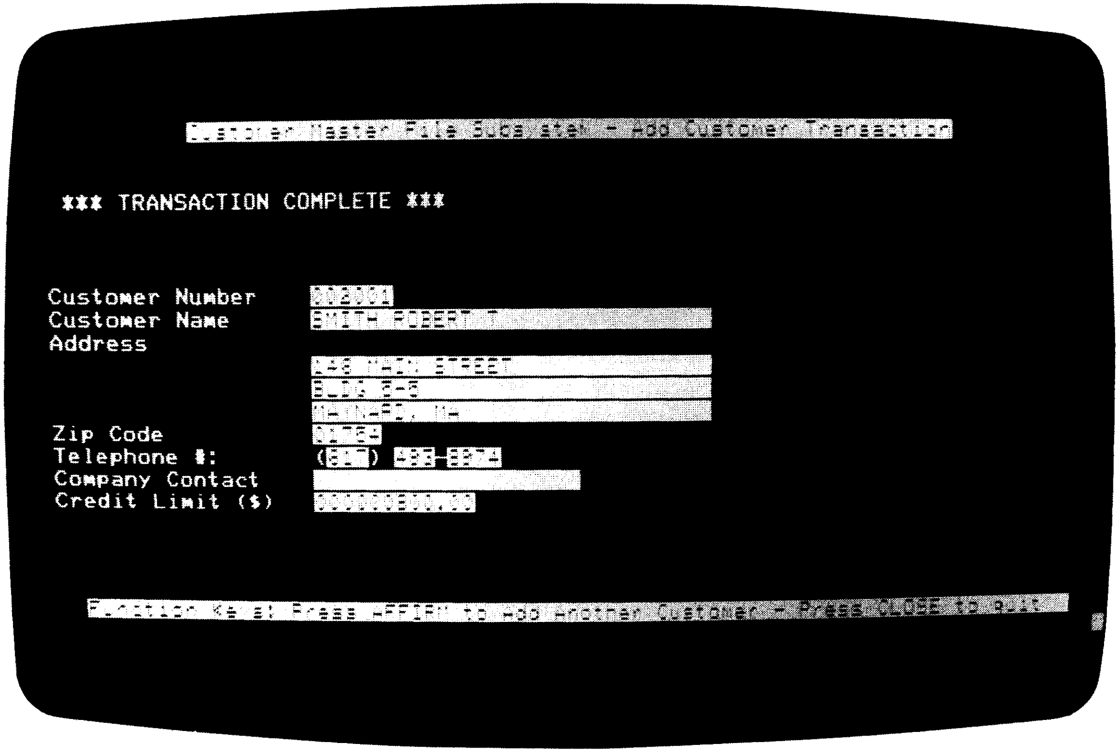
To transmit the completed form to the terminal station for processing, you follow the instructions shown on the last line of the screen display; that is, press the ENTER key. The data on the form is sent to the terminal station where it is formatted into an exchange message and passed to the exchange routing list. The TSTs associated with this transaction process the exchange message to create a new customer record. After writing the new record to the customer file, the last TST in the exchange sends a reply response message back to the entry form. The reply message contains a 6-character customer ID that is written to REPLY screen number 1. The completed REPLY screen number is shown in Figure 1-7.

After the reply screen has been written, you may press one of two enabled system function keys, the AFFIRM key or the CLOSE key. The AFFIRM key ends that transaction instance, and redisplays the empty data entry screen, while the CLOSE key ends the transaction instance, and causes the transaction selection form to appear on the terminal screen.



```
          Customer Master File Subsystem - Add Customer Transaction


     To Add a Customer to the File, Complete all Form Fields and Press ENTER.


   Customer Number    ??????     (To be Supplied by System)
   Customer Name      SMITH ROBERT T
   Address
                      146 MAIN STREET
                      BUILDING 5-5
                      MAYNARD, MA
   Zip Code           01754
   Telephone #:       (617) 493-8874
   Company Contact
   Credit Limit ($)   000000800.00



     Function Keys: ENTER to Add Customer- CLOSE to quit Add Function
```

Figure 1-6 Completed Entry Form



```
        Customer Master File Subsystem - Add Customer Transaction


*** TRANSACTION COMPLETE ***


Customer Number    002001
Customer Name      SMITH ROBERT T
Address
                   146 MAIN STREET
                   BLDG 5-5
                   MAYNARD, MA
Zip Code           01754
Telephone #:       (617) 493-8874
Company Contact
Credit Limit ($)   000000800.00



   Function Keys: Press AFFIRM to Add Another Customer - Press CLOSE to quit
```

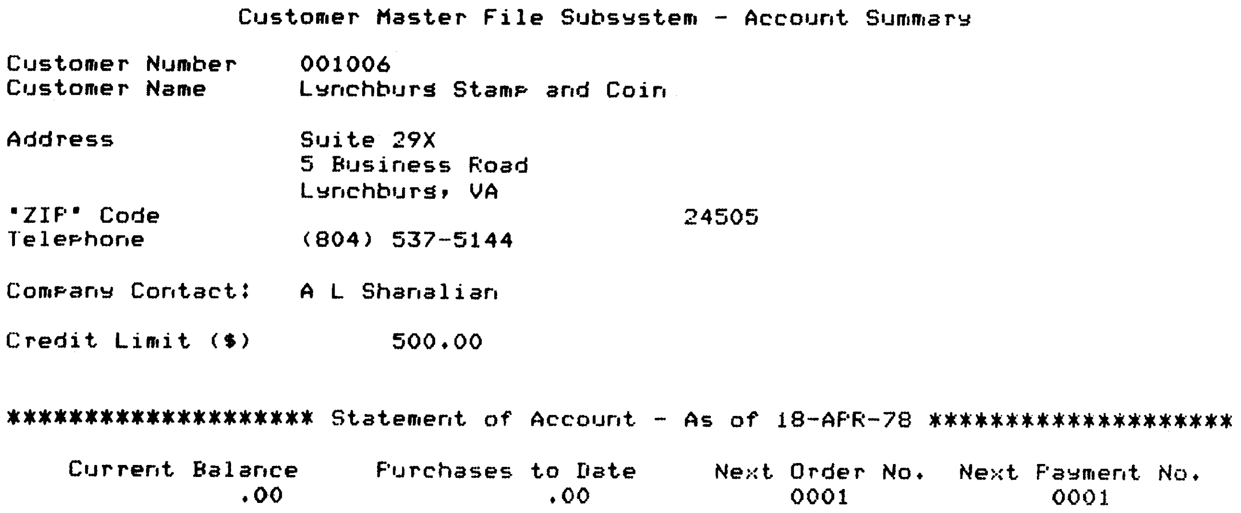
Figure 1-7 Entry Form after Reply #1

1.11.3 Using a Report Form

A report is a form used to display information on an output-only device such as the LA180 DECprinter. Using a report requires you to code an output-only form definition using ATL PRINT statements. In addition, a report-type response message must be sent to the output-only terminal from an application TST. You must also define the output-only terminal with a station name using the STADEF utility. (See the *TRAX Application Programmers Guide* AA-D329A-TC.)

Report forms can be used to produce a wide variety of hard-copy reports, for example, picking lists, shipping invoices, account statements, or order forms.

In the transaction processor SAMPLE (used for examples in this manual), the display customer transaction is set up with a report capability. After retrieving a record from the customer file, you can press the KEYDOT user function key from the second exchange of the display customer transaction DPYCUS. The TST associated with this exchange checks your input, and if KEYDOT is detected, generates a Report using the data from the customer record currently displayed on the application terminal. Figure 1-8 illustrates the report form sent to the output-only station.



```
               Customer Master File Subsystem - Account Summary

Customer Number    001006
Customer Name      Lynchburg Stamp and Coin

Address            Suite 29X
                   5 Business Road
                   Lynchburg, VA
'ZIP' Code                                  24505
Telephone          (804) 537-5144

Company Contact:   A L Shanalian

Credit Limit ($)         500.00

******************** Statement of Account - As of 18-APR-78 ********************

    Current Balance      Purchases to Date      Next Order No.  Next Payment No.
               .00                  .00              0001              0001
```

Figure 1-8 Report Form Output to LA180 Terminal

CHAPTER 2

THE APPLICATION TERMINAL LANGUAGE— AN OVERVIEW

2.1 HOW TO USE THE ATL LANGUAGE

ATL consists of a set of *statements, clauses,* and *parameters.*

You can use ATL to create form definitions for transaction processors. Using ATL, you can specify the layout and appearance of each form, the attributes of each field on those forms, the format of the exchange message created from user input, and the set of replies used by the transaction processor to respond to user input.

A form is designed and included in a transaction definition by the following steps:

1. The application designer examines the needs of the business, and determines the input and display operations required from an application terminal.
2. Once the transaction is divided into exchanges, the designer specifies the forms required for each exchange.
3. The designer or an application programmer codes the form definition using ATL.
4. The coded form definition is entered into a source file using the DEC Editor.
5. The source file is submitted to the ATL utility for compilation and debugging.
6. Once a form has compiled without errors, the ATL utility is run to add the form to the forms definition file of the transaction processor.
7. The transaction processor is installed and started, and the form is tested to insure it functions properly.

2.1.1 Statements

The statement is the fundamental element of ATL. There are 12 distinct types of statements. These fall into two major classes: form definition statements and ATL complier directive statements.

The form definition statements are used to define a form and its associated fields and messages. These statements require that at least one clause be specified in addition to the statement keyword and the statement parameters.

Each statement must begin on a new line of the source file. A statement keyword is often followed by one or more statement parameters. These parameters provide locational and structural information about the field or message being defined by the statement. The equal sign (=) is used to separate a statement keyword from its associated parameters.

The most common statement parameters are the row and column position of the field, and a number parameter whose use varies according to the statement types. Most statements also have one or more clauses associated with them.

The following section describes the 12 ATL statement types:

2.1.1.1 Descriptions of ATL Form Definition Statements

FORM

The FORM statement is used to specify general form parameters. You can also enable or disable function keys, cause the bell to ring, and divide the form into Display and Form areas by using the SPLIT clause.

The following five statements define fields on a form. Input and Prompt fields are in the Form area of the screen. Menu and Display fields are in the display area of the screen. Print fields are used only by output-only forms.

In all five field definition statements, you must specify location parameters as part of the statement, in the form:

```
statement-keyword = row-number, column-number
```

For example:

```
INPUT = 1,1
```

The row parameter specifies the field location relative to the beginning of the *area* in which the field is specified. Thus DISPLAY = 1,1 positions a display field on the first line of the display area, while INPUT = 1,1 positions a display field on the first line of the form area. In a PRINT statement, the row parameter specifies the field location relative to the top of the form.

INPUT

The INPUT statement is used to specify Input fields used in data entry operations. As part of an INPUT statement, you can specify the data entry and character representation attributes of the field, a label for the field, the length of the field, the initial value of the field, and the clear character used by the transaction processor to fill empty field positions.

PROMPT

The PROMPT statement is used to specify Prompt fields that provide instructions in the Form area of a screen. As part of a PROMPT statement, you can specify the initial value of the field, the length of the field, a label for the field, and either normal or reverse video display characteristics.

DISPLAY

The DISPLAY statement is used to specify Display fields. Display fields provide instructions and reply message data in the Display area of a screen. As part of a DISPLAY statement, you can specify the initial content of the field, the length of the field, a label for the field, normal or reverse video, and whether or not the field is blanked on the initial and subsequent form displays.

MENU

The MENU statement is used to specify Menu fields used to list items for user selection. The MENU statement requires that you specify an initial value for the field.

PRINT

The PRINT statement is used to specify the fields on a Report form sent to an output-only terminal. You must specify an initial value as part of the PRINT statement.

The MESSAGE and REPLY statements are used to define the interface between the form and the TSTs that use the form. The MESSAGE statement defines the format of the exchange message. The REPLY statement defines the modifications that occur when a TST sends a reply-type response message back to the form.

MESSAGE
The MESSAGE statement is used to specify the format of the exchange message constructed from the Input and Menu fields of the form. The MESSAGE statement allows you to specify a beginning character position for the message, and a number of different types of data that the terminal station places directly into the exchange message.

REPLY
The REPLY statement is used to specify the set of modifications made to a form when a reply-type response message is received by the form. You use the REPLY statement to specify how various labelled fields are modified, to enable or disable function keys, ring the terminal bell, or to move the cursor to a specific Input of Menu field. You must supply a number parameter to the REPLY statement. This number corresponds to the number parameter sent with the reply message.

2.1.1.2 **ATL Compiler Directive Statement Descriptions** – The compiler directive statements are used to control the way in which the ATL utility compiles source statements. The four compiler directive statements are:

DEFAULT
The DEFAULT statement is used to change the defaults assumed by the ATL utility. Using this statement, you can specify a new set of default field attributes, enabled function keys, and a clear character.

REPEAT
The REPEAT statement is used to delimit the beginning of a block of ATL statements that are to be repeatedly compiled by the ATL utility. The number parameter specifies the repeat count. You must specify the WITH clause as part of the REPEAT statement. This allows you to set up dummy parameters in the repeated statements, which are changed with each execution of the repeat block.

REND
The REND statement delimits the end of a repeat block.

END
The END statement delimits the end of a form definition source file.

2.1.2 Clauses

Each clause modifies the form, field, or message specified by the statement. Clauses generally have parameters associated with them, either keywords, values, or literal text strings. A clause keyword is separated from its associated parameters by the equal sign (=). Clause parameters are separated from each other by commas (,).

2.1.2.1 **Summary of ATL Clause Types**–The following clauses are permitted in ATL statements. This listing is only a summary. Complete clause specification information is given in Chapter 5.

ATTRIBUTES

The ATTRIBUTES clause can be specified as part of the DEFAULT, DISPLAY, INPUT, and PROMPT statements. This clause lets you specify the way data must by entered and/or displayed in a field. A complete list of attribute keywords is given in Section 2.3.2. The attribute keywords are explained briefly here, and in Chapter 5 as part of the individual statement descriptions. The most commonly specified attribute keywords are:

ANY An Input field can contain any displayable ASCII character in the range OCTAL 040 to 176.

LETTERS An Input field can contain only the letters A through Z, a through z, and space.

NUMERIC An Input field can contain only the numbers 0 through 9. Spaces are not permitted.

ALPHANUMERIC An Input field can contain any character that is a number, a letter, or a space.

SIGNED An Input field can contain signed numeric data: The numbers 0 through 9, the sign characters + and -, and field punctuation characters (, and.). Spaces are not permitted.

LEFT Data entered into an INPUT field appears in a left-justified format.

RIGHT Data entered into an INPUT field appears in a right-justified format.

NOTAB To advance to another Input field, the terminal user must press the NEXT FIELD or FORWD FIELD key after typing data into an Input field.

TAB After the user has completely filled the current field, the cursor moves automatically to the next Input field.

FULL The user must type enough characters to completely fill the Input field.
The user may skip past a field specified with the FULL attribute by pressing the FORWARD FIELD key.

NOFULL Input fields need not be filled completely.

REQUIRED The user must enter data into this field before going on to the next field.

NOREQUIRED The user may skip past this field without entering data.

MODIFY The terminal user may change the contents of an Input field specified with this attribute.

NOMODIFY

The terminal user cannot change the contents or enter data in an Input field specified with this attribute. Furthermore, the user cannot position the cursor in this field.

NORMAL

The Input, Prompt or Display field appears as white characters on a dark background.

REVERSE

The Input, Prompt or Display field appears as dark characters on a white background. Spaces appear as white squares.

BLANK

This attribute can be specified for Display fields only. A Display field with this attribute is not displayed on the initial screen representation of the form. The field appears only on the reply screen displays defined for a form. If a reply does not specify text for a Display field with the BLANK attribute, that field is blanked on the reply screen.

NOBLANK

A Display field defined as NOBLANK is displayed on the screen as part of the initial form. It is not automatically blanked for a REPLY but a REPLY may explicitly blank or overwrite it.

NOECHO

An Input field can be specified with the NOECHO attribute. When the user types data into such a field, it never appears on the screen. Only one field per form can have this attribute. NOECHO fields are limited to 40 characters in length.

BELL

The BELL clause can be specified as part of the FORM and REPLY statements. This clause lets you specify when and for how long the terminal bell sounds.

CLEAR

The CLEAR clause can be specified as part of the DEFAULT and INPUT statements. This clause lets you specify the character used to fill empty spaces in Input fields, both at the time they are first displayed, and when they are transmitted to the terminal station.

CURSOR

The CURSOR clause can be specified as part ot the REPLY statement. This clause lets you specify where the cursor is positioned when a reply screen has been displayed.

DISABLE

The DISABLE clause can be specified as part of the DEFAULT, FORM, and REPLY statements. This clause lets you specify which user and system function keys are to be disabled when a form or reply screen is displayed.

ENABLE

The ENABLE clause can be specified as part of the DEFAULT, FORM, and REPLY statements. This clause lets you specify which user and system function keys are to be enabled when a form or reply screen is displayed.

KEYCAP
The KEYCAP clause can be specified as part of the FORM statement. This clause allows you to specify new keycap text strings for user function keys. The value specified in this clause can be transmitted to the terminal station by specifying the VALUE=KEY clause in the MESSAGE statement for that form. See Section 3.1.1.5 for a detailed explanation of how this clause is used.

LABEL
The LABEL clause can be specified as part of the DISPLAY,INPUT, or PROMPT statement. This clause allows you to name a field for subsequent reference in a MESSAGE statement VALUE clause, or a REPLY statement WRITE clause.

LENGTH
The LENGTH clause can be specified as part of the DISPLAY, FORM, INPUT, and PROMPT statements. When used with the FORM statement, this clause lets you specify the number of lines on a form. When used with the DISPLAY, INPUT, or PROMPT statements, this clause specifies the number of characters in the field defined by the statement.

SELECT
The SELECT clause can be specified as part of the FORM statement. This clause identifies a form as a transaction selection form. This clause also specifies the reply screens to be displayed if a selected transaction is not known to the transaction processor, or is not authorized for display on the selecting terminal.

SPLIT
The SPLIT clause can be specified as part of the FORM statement. This clause is used to define the size of a form's Display area. On a VT62, the maximum value specified as the parameter is 23. If the SPLIT clause is omitted, the default value is SPLIT = 0 (i.e. the form has no display area).

VALUE
The VALUE clause can be specified as part of the DISPLAY, INPUT, MENU, MESSAGE, PRINT, and PROMPT statements. In addition, the VALUE clause parameters are also permissible as WRITE clause parameters in a REPLY statement. This clause lets you specify the contents of a field or a message. The VALUE clause parameters allow you considerable flexibility in initializing fields and constructing exchange messages. You can specify a number of parameters as part of a VALUE clause. These parameters are listed in Section 2.3.2, and described briefly here, and in detail as part of the statement descriptions in chapter 5.

MENU	The contents of the selected MENU items are placed in the exchange message. The length assumed for this parameter is equal to the length of the longest defined menu field. The null character (OCTAL 000) is used to pad any unfilled character positions in the exchange message.
label-name	The contents of the Input field described by the label parameter are placed in the exchange message.
KEY	When a user function key is pressed the text string associated with that function key is placed in the exchange message.

(See Section 3.1.1.5 for a discussion of this feature.) The length of the field to be defined is the length of the longest defined key text.

"string"

A string of characters enclosed by quotation marks. The enclosed string is displayed on the screen exactly as typed, without the quotation marks.

FILL ("character", count)

The character you specify in the first parameter is written into the field the number of times specified by the second parameter.

DATE

The current date is written into the field in the format: DD-MMM-YY

TIME

The current time of day is written into the field in the format: HH:MM:SS

TRANSACTION

The system-determined transaction instance number is written into the field as ten ASCII characters.

NAME

The 6-character name of the transaction that is currently being run from the application terminal is written into the field.

STATION

The 6-character terminal station name is written into the field.

REQUEST (postition, count)

The REQUEST function allows you to obtain information contained in the response message that requested this form.

When you specify a field with a VALUE clause containing the REQUEST function, the field is filled with text from the requesting response message, beginning at the character postition specified by the first parameter, and continuing for the number of characters specified by the second parameter.

The first character position in the requesting response message can be referenced by specifying 1 as the first parameter of the REQUEST function.

Characters are moved from the requesting message to the field in left-to-right order, regardless of the field justification attribute of the receiving field.

When a VALUE clause is used to specify the contents of an Input, Display, Menu, Print, or Prompt field, a length is implicitly specified as the number of characters supplied by the VALUE clause parameters. ATL permits an explicit length value through the LENGTH clause. When a field has both an implicit and explicit length specification, ATL assigns the greater value as the field length.

WIDTH
The WIDTH clause can be specified as part of the FORM statement. This clause lets you specify the logical page width of a form used on an output-only device.

WITH
The WITH clause can be specified as part of the REPEAT statement. The parameters of the WITH clause let you specify initial values and increments for dummy integer and character variables used as part of statements in a repeat block.

WRITE
The WRITE clause allows you to specify modifications to be made when a REPLY acts upon a previously defined field. The WRITE clause paramenters identify a field, and describe how it is filled when the reply screen is displayed.

When a BLANK display field is referenced by a WRITE clause, any portion of the field that is not explicitly specified in a VALUE parameter is erased automatically on the reply screen.

If you specify a WRITE clause with only the label-name parameter, and do not specify the field contents, the REPLY writes the specified field according to the VALUE clauses specified in the original field definition statement.

In this case, the original field definition must have at least oneVALUE clause and that VALUE clause must not contain the REQUEST () parameter.

The WRITE clause must contain a label-name parameter. A number of VALUE parameters may also be specified. A brief description of these parameters is listed here. Detailed explanations of these parameters are found in the description of the REPLY statement in Chapter 5.

label-name
The label-name parameter specifies the label-name of the field written into by the subsequent parameters in the WRITE clause. The field specified by this label can be an Input, Display or Prompt field but not a Menu field.

"string"
A string of characters enclosed by quotation marks.

FILL ("character", count)
The character specified in the first parameter is written into the field referenced by the label the number of times specified in the second parameter.

DATE
The current date is written to the specified field in the format: DD-MMM-YY

TIME
The current clock time is written to the specified field in the format: HH:MM:SS

TRANSACTION — The system-determined transaction instance number is written to the specified field as ten ASCII characters.

NAME — The 6-character name of the current transaction is written to the specified field.

STATION — The 6-character terminal station ID is written to the specified field.

REQUEST (position, count) — The REQUEST function allows you to obtain information contained in the response message that requested the current reply screen. A field written using the REQUEST function in a WRITE clause is filled with text from the requesting response message beginning at the character position specified by the first parameter, and continuing for the number of characters specified by the second parameter.

2.2 ATL SYNTAX RULES

2.2.1 General Statement Format

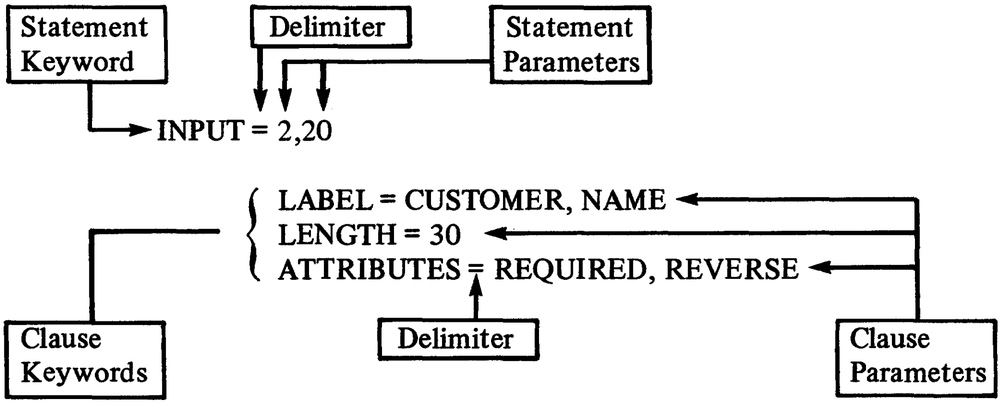
An ATL statement has the general format:

```
Statement-keyword   [=statement-parameters]

   ┌clause 1┐
   │clause 2│
   │  . . . │
   └clause n┘
```

An ATL clause has the general format:

```
Clause-keyword   [=parameter 1, parameter 2, . . . , parameter n]
```

The following diagram identifies the various parts of a typical ATL source statement:



ATL statements are generally free-form, allowing you to code source statements for easy reading. ATL assumes a few syntax conventions. They require that:

1. Only one statement may be coded on a source line.
2. A statement may be continued on more than one line.
3. A statement keyword must be separated from its statement parameters by an equal sign (=).
4. Clause keywords must be separated from the parameter list by an equal sign (=).
5. Statements must be separated from clauses, and clauses from each other by one or more spaces, tabs, and/or line terminators.
6. Additional spaces and tabs may be inserted anywhere in a source line, except within a keyword or parameter specification.
7. In ATTRIBUTE and VALUE clauses, and in the VALUE parameter of the WRITE clause, you can specify any number of parameters in any order you require.
8. In the VALUE and WRITE clauses, you can specify the same parameter keyword or parameter type more than once in the same clause.
9. When you specify more than one parameter to a clause, you must separate items with a comma.
10. You can specify only one ATTRIBUTE clause keyword from a keyword grouping. For example, you cannot specify SIGNED and NUMERIC attributes for the same field.
11. Fields may not overlap each other.

The following illustrates different forms of the same ATL statement:

```
INPUT = 2,20 ATTRIBUTES = TAB, REVERSE, NUMERIC
```

or

```
INPUT = 2,20
ATTRIBUTES = TAB
             REVERSE,
             NUMERIC
```

2.2.2 Statement Ordering

The order in which statements are used in an ATL form definition is generally unimportant. There are, however, a few instances where the statement sequence becomes important.

1. Each form definition must end with an END statement.
2. The set of defaults specified by a DEFAULT statement apply only to statements that follow the DEFAULT statement. They remain in effect until overridden by a subsequent default statement.
3. The visible cursor advances from Input field to Input field in the order they are defined in the form definition. Similarly, the cursor moves form Menu field to Menu field in the order in which they were defined.
4. REPEAT and REND statements affect only those statements between them.
5. The values taken by the row and column parameters of the statements that use the dot construct (See Section 2.2.6) depend on the location and length of the preceding field.

2.2.3 Abbreviating Keywords

All examples in this manual use the full spelling of keywords. For purposes of clarity and readability, you are strongly advised to use the full spelling of a keyword whenever possible.

However, if you wish, you may abbreviate statement keywords, clause keywords, and parameter keywords. The abbreviation must contain at least three characters and must uniquely identify the keyword. An example of the concept of uniqueness is illustrated in the following ATL statement:

```
INPUT = 1,10
          LABEL = CUSTOMER–NAME
          ATTRIBUTES = REQ, NUM, REV
          LENGTH = 6
```

If you were to attempt to compile this statement using the ATL utility, a syntax error would occur, since the keyword REQ is not unique. REQ is the first three letters of two keywords, REQUIRED, and REQUEST. In this case, you must specify REQUI in order to properly abbreviate REQUIRED. A complete listing of ATL keywords and unique abbreviations is contained in Table 2-1.

You can use both upper- and lower-case characters in source statements. All lower-case source statement text that is not part of a literal string is translated to upper case by the ATL utility.

2.2.4 Literal Text Delimiter

All literal text strings in a form definition must be delimited by either the double quotation mark (“) character or the single quotation mark (‘) character. If you wish to include a quoted string as part of some literal text, you must first quote the entire literal text string using one type of quotation mark, then enclose that quoted text string with a pair of the other type of quote marks. The following example shows how you can enter literal text as part of a source statement:

```
PROMPT = 2,10
             VALUE = “Customer Number:”
```

To insert quotes in a literal text string, use both delimiters:

```
PROMPT = 2,10
            VALUE = ‘ “EOQ” Quantity:’
```

The null text-string is specified by two adjacent quotation characters (‘ ’ or “ ”).

2.2.5 Using Comments in ATL Statements

A comment is delimited by an exclamation point (!). All text between an exclamation point and the line terminator (or another exclamation point) is considered a comment, printed on your statement listing, and otherwise ignored by the ATL utility. Exclamation points contained in a quoted text are considered part of the text, and are not comment delimiters.

Table 2-1
ATL Keywords & Abbreviations

KEYWORD	ABBR	KEYWORD	ABBR
AFFIRM	AFF	MESSAGE	MES
ALPHANUMERIC	ALP	MODIFY	MOD
ANY	ANY	NAME	NAM
ATTRIBUTES	ATT	NOECHO	NOEC
BELL	BEL	NOENTER	NOEN
CLEAR	CLE	NOFULL	NOF
CLOSE	CLO	NOMODIFY	NOM
CURSOR	CUR	NOREQUIRED	NORE
DATE	DAT	NORMAL	NORM
DEFAULT	DEF	NOTAB	NOT
DISABLE	DISA	PRINT	PRI
DISPLAY	DISP	PROMPT	PRO
ENABLE	ENA	REND	REN
END	END	REPEAT	REPE
ENTER	ENT	REPLY	REPL
FILL	FIL	REQUEST	REQUE
FORM	FOR	REQUIRED	REQUI
FULL	FUL	REVERSE	REV
INPUT	INP	RIGHT	RIG
KEY	KEY	SELECT	SEL
KEY0	KEY0	SIGNED	SIG
KEY1	KEY1	SPLIT	SPL
KEY2	KEY2	STATION	STA
KEY3	KEY3	STOPREPEAT	STO
KEYCAP	KEYC	TAB	TAB
KEYDOT	KEYD	TIME	TIM
LABEL	LAB	TRANSACTION	TRA
LEFT	LEF	VALUE	VAL
LENGTH	LEN	WIDTH	WID
LETTERS	LET	WIDTH	WIT
MENU	MEN	WRITE	WRI

A comment can consist of an entire source line. The example below shows how comments can be used to graphically enhance your ATL source statements:

```
!**************************************************************!
!                                                              !
!                     FORM NAME:  CHCUS1                       !
!                                                              !
!**************************************************************!
```

Comments can also appear as part of a statement or clause:

```
INPUT = 1,20
        LABEL = CUST. NO              !Assign label name to field
        LENGTH = 6                    !Define field length
        ATTRIBUTE = TAB, NUMERIC      !Field is right-justified
                    REV, RIGHT        !numeric, reverse video + auto tab.
```

2.2.6 The "DOT" Constructs

The field definition statements (DISPLAY, INPUT, MENU, PRINT, and PROMPT) require that you specify row and column parameters to define the initial character position of the field.

To simplify the coding of forms, you can specify row and column values as relative offsets from the previous field through the use of the ATL "dot" constructs. A "." in a row or column parameter of an ATL statement refers to the character position immediately following the previous field.

You may also specify spacing by supplying .+nn or .-nn values in the row and column parameters. For example, the following DISPLAY statement specifies that the field is to begin ten characters past the end of the preceding field.

```
DISPLAY = . , . + 10
```

An advantage of the dot construct is that it allows you to reposition whole blocks of ATL statements simply by changing the value of the row parameter for the first field. For example, if the first statement in the display area is:

```
DISPLAY = 1,1
```

and all subsequent statements use the dot construct, then changing the row parameter in the first statement to read:

```
DISPLAY = 2,1
```

moves all display area fields down one row.

2.2.7 Assumed ATL Utility Default Conditions

If a form is specified without a FORM statement, or if certain clauses are not specified as part of the FORM statement, the ATL utility assumes the following default conditions exist:

1. The form being defined contains a Form area only, i.e. there is no Display area. The assumed value of the SPLIT clause is SPLIT = 0.
2. The ABORT system function key, and the ENTER user function key are enabled. All other system and user function keys are disabled. (To change the set of enabled function keys, see the ENABLE and DISABLE clause descriptions in this Chapter and in Chapter 5.)
3. The form is not being used to select the next transaction. (A transaction selection form requires a SELECT clause in the FORM statement. See the SELECT clause description.)

4. The default keycap text values are associated with the user function keys. (You can change keycap text using the KEYCAP clause. See the KEYCAP clause in this chapter and in Chapter 5.)
5. The Bell will not ring when an initial form display occurs. (The Bell is not rung unless the BELL clause is specified.)

2.3 ATL LANGUAGE SUMMARY

This section presents the statements, clauses, parameters, and keywords of the ATL language in a ready-reference format. No attempt is made to define language element usage. A complete discussion of ATL language elements and how they are used is given in Chapter 5.

2.3.1 Conventions Used to Describe the Language

[ ] Special brackets indicating optional information that can be omitted from a statement or clause.

{ } Braces indicating that a choice of one or more parameters must be made from the set enclosed by the braces. You can also specify a parameter more than once as part of the same clause.

Lower-case letters Parameters described in lower-case letters indicate data that you must supply such as a label, number, or text-string.

UPPER-CASE LETTERS Parameters shown in upper-case are keywords. They must be specified and spelled as shown in the parameter list.

Bold Face Keywords The default parameter keyword values assumed for certain clauses (ATTRIBUTES, for example) are shown in bold face.

● ● ● Ellipsis indicate clauses and parameters can be repeatedly specified.

2.3.2 Summary of ATL Statements and Clauses

DEFAULT

```
┌                         ⎛ ⎧ ANY           ⎫ ⎞   ┌                      ┐┐
│                         ⎜ ⎪ ALPHANUMERIC  ⎪ ⎟   │                      ││
│   ATTRIBUTES =          ⎜ ⎨ LETTERS       ⎬ ⎟   │   ⎧             ⎫    ││
│                         ⎜ ⎪ NUMERIC       ⎪ ⎟   │   ⎪     ≡       ⎪    ││
│                         ⎜ ⎩ SIGNED        ⎭ ⎟   │   ⎪             ⎪    ││
│                         ⎜                   ⎟   │   ⎪             ⎪    ││
│                         ⎜ ⎧ LEFT  ⎫         ⎟   │   ⎪     ≡       ⎪    ││
│                         ⎜ ⎩ RIGHT ⎭         ⎟   │   ⎪             ⎪    ││
│                         ⎜                   ⎟   │   ⎪             ⎪    ││
│                         ⎜ ⎧ NOTAB ⎫         ⎟   │ , ⎨    • • •    ⎬    ││
│                         ⎜ ⎩ TAB   ⎭         ⎟   │   ⎪             ⎪    ││
│                         ⎜                   ⎟   │   ⎪     ≡       ⎪    ││
│                         ⎜ ⎧ NOFULL ⎫        ⎟   │   ⎪             ⎪    ││
│                         ⎜ ⎩ FULL   ⎭        ⎟   │   ⎪     ≡       ⎪    ││
│                         ⎜                   ⎟   │   ⎩             ⎭    ││
│                         ⎜ ⎧ NOREQUIRED ⎫    ⎟   │                      ││
│                         ⎜ ⎩ REQUIRED   ⎭    ⎟   │                      ││
│                         ⎜                   ⎟   │                      ││
│                         ⎜ ⎧ MODIFY   ⎫      ⎟   │                      ││
│                         ⎜ ⎩ NOMODIFY ⎭      ⎟   │                      ││
│                         ⎜                   ⎟   │                      ││
│                         ⎜ ⎧ NORMAL  ⎫       ⎟   │                      ││
│                         ⎜ ⎩ REVERSE ⎭       ⎟   │                      ││
│                         ⎜                   ⎟   │                      ││
│                         ⎜ ⎧ BLANK   ⎫       ⎟   │                      ││
└                         ⎝ ⎩ NOBLANK ⎭       ⎠   └                      ┘┘

[ ENABLE = keyname ]

[ DISABLE = keyname ]

[ CLEAR = "character"]
```

```
DISPLAY = row, column

 [ VALUE = { "string"                    } [ , { "string"                    } , . . . ] ]
 [         { FILL ("character", count)   } [   { FILL ("character", count)   }         ] ]
 [         { DATE                        } [   { DATE                        }         ] ]
 [         { TIME                        } [   { TIME                        }         ] ]
 [         { TRANSACTION                 } [   { TRANSACTION                 }         ] ]
 [         { NAME                        } [   { NAME                        }         ] ]
 [         { STATION                     } [   { STATION                     }         ] ]
 [         { REQUEST (position, count)   } [   { REQUEST (position, count)   }         ] ]

 [ LENGTH = count ]

 [ ATTRIBUTES =   { { NORMAL   } }  [ , { = } ] ]
 [                { { REVERSE  } }  [   {   } ] ]
 [                {              }  [   {   } ] ]
 [                { { BLANK    } }  [   { = } ] ]
 [                { { NOBLANK  } }  [   {   } ] ]

 [ LABEL = label-name ]

END

FORM

 { [ SPLIT = length ]                                                              }
 {                                                                                 }
 { [ ENABLE = keyname ]                                                            }
 {                                                                                 }
 { [ DISABLE = keyname ]                                                           }
 {                                                                                 }
 { [ KEYCAP = keyname, "text-string" ]                                             }
 {                                                                                 }
 { [ SELECT = { MENU              }        ,reply-1      { ,reply-2       } ]      }
 { [          { input-field-label }                      { ,NOAUTHORIZE   } ]      }
 {                                                                                 }
 { [ LENGTH = { line-count } ]                                                     }
 { [          { FEED       } ]                                                     }
 {                                                                                 }
 { [ BELL   [= periods] ]                                                          }
 {                                                                                 }
 { [ WIDTH = form-width ]                                                          }
```

```
NPUT = row, column

  {  [ATTRIBUTES = ( { ANY            } )  [    {  ___  }  ] ]  }
  {  [             ( { ALPHANUMERIC   } )  [    {  ___  }  ] ]  }
  {  [             ( { LETTERS        } )  [    {  ___  }  ] ]  }
  {  [             ( { NUMERIC        } )  [    {  ___  }  ] ]  }
  {  [             ( { SIGNED         } )  [    {  ___  }  ] ]  }
  {  [             (                    )  [    {       }  ] ]  }
  {  [             ( { LEFT       }     )  [    {  ___  }  ] ]  }
  {  [             ( { RIGHT      }     )  [    {       }  ] ]  }
  {  [             (                    )  [    {       }  ] ]  }
  {  [             ( { NOTAB      }     )  [    {  ___  }  ] ]  }
  {  [             ( { TAB        }     )  [    {       }  ] ]  }
  {  [             (                    )  [    {       }  ] ]  }
  {  [             ( { NOFULL     }     )  [ ,  {  ...  }  ] ]  }
  {  [             ( { FULL       }     )  [    {       }  ] ]  }
  {  [             (                    )  [    {       }  ] ]  }
  {  [             ( { NOREQUIRED }     )  [    {  ___  }  ] ]  }
  {  [             ( { REQUIRED   }     )  [    {  ___  }  ] ]  }
  {  [             (                    )  [    {       }  ] ]  }
  {  [             ( { MODIFY     }     )  [    {  ___  }  ] ]  }
  {  [             ( { NOMODIFY   }     )  [    {  ___  }  ] ]  }
  {  [             (                    )  [    {       }  ] ]  }
  {  [             ( { NORMAL     }     )  [    {       }  ] ]  }
  {  [             ( { REVERSE    }     )  [    {       }  ] ]  }
  {  [             (                    )  [    {       }  ] ]  }
  {  [             ( { NOECHO     }     )  [    {       }  ] ]  }
  {
  {  [VALUE =  { "string"                    }  [   { "string"                    }      ] ]
  {  [         { FILL ("character", count)   }  [   { FILL ("character", count)   }      ] ]
  {  [         { DATE                        }  [   { DATE                        }      ] ]
  {  [         { TIME                        }  [   { TIME                        }      ] ]
  {  [         { TRANSACTION                 }  [ , { TRANSACTION                 } ,... ] ]
  {  [         { NAME                        }  [   { NAME                        }      ] ]
  {  [         { STATION                     }  [   { STATION                     }      ] ]
  {  [         { REQUEST (position, count)   }  [   { REQUEST (position, count)   }      ] ]
  {
  {   [LENGTH = count]
  {
  {   [CLEAR = "character"]
  {
  {   [LABEL = label-name]

MENU = row, column

    VALUE =  { "string"                    }  [   { "string"                    }      ]
             { FILL ("character", count)   }  [ , { FILL ("character", count)   } ,... ]
             { REQUEST (position, count)   }  [   { REQUEST (position, count)   }      ]
```

```
MESSAGE = position

    VALUE =   "string"                          [  "string"                       ]
              FILL ("character", count)         [  FILL ("character", count)      ]
              DATE                              [  DATE                           ]
              TIME                              [  TIME                           ]
              TRANSACTION                       [  TRANSACTION                    ]
              NAME                              [, NAME                       ,•••]
              STATION                           [  STATION                        ]
              REQUEST (position, count)         [  REQUEST (position, count)      ]
              MENU                              [  MENU                           ]
              label-name                        [  label-name                     ]
              KEY                               [  KEY                            ]

PRINT = row, column

    VALUE =   "string"                          [  "string"                       ]
              FILL ("character, count)          [  FILL ("character", count)      ]
              DATE                              [  DATE                           ]
              TIME                              [, TIME                           ]
              TRANSACTION                       [  TRANSACTION                ,•••]
              NAME                              [  NAME                           ]
              STATION                           [  STATION                        ]
              REQUEST (position, count)         [  REQUEST (position, count)      ]

PROMPT = row, column

  [ VALUE =   "string"                          [  "string"                       ]]
  [           FILL ("character, count)          [  FILL ("character", count)      ]]
  [           DATE                              [  DATE                           ]]
  [           TIME                              [, TIME                           ]]
  [           TRANSACTION                       [  TRANSACTION                ,•••]]
  [           NAME                              [  NAME                           ]]
  [           STATION                           [  STATION                        ]]
  [           REQUEST (position, count)         [  REQUEST (position, count)      ]]

  [LENGTH = count]

  [ATTRIBUTE = NORMAL ]
  [            REVERSE]

  [LABEL = label-name]

REPEAT = number

          [WITH #n = start [,increment]]

          [WITH #n = "character" ]

REND      • • •
```

```
EPLY = number

   WRITE = field-label [ ⎧ "string"                     ⎫ [   ⎧ "string"                     ⎫     ] ] ⎫
                       [ ⎪ FILL ("character", count)    ⎪ [   ⎪ FILL ("character", count)    ⎪     ] ] ⎪
                       [ ⎪ DATE                         ⎪ [   ⎪ DATE                         ⎪     ] ] ⎪
                       [ ⎨ TIME                         ⎬ [ , ⎨ TIME                         ⎬ ,...] ] ⎪
                       [ ⎪ TRANSACTION                  ⎪ [   ⎪ TRANSACTION                  ⎪     ] ] ⎪
                       [ ⎪ NAME                         ⎪ [   ⎪ NAME                         ⎪     ] ] ⎪
                       [ ⎪ STATION                      ⎪ [   ⎪ STATION                      ⎪     ] ] ⎬
                       [ ⎩ REQUEST (position, count)    ⎭ [   ⎩ REQUEST (position, count)    ⎭     ] ] ⎪
                                                                                                       ⎪
     [CURSOR = field-label]                                                                            ⎪
                                                                                                       ⎪
     [ENABLE = keyname]                                                                                ⎪
                                                                                                       ⎪
     [DISABLE = keyname]                                                                               ⎪
                                                                                                       ⎪
     [BELL   [= periods ]]                                                                             ⎭
```

CHAPTER 3
EXAMPLES OF HOW TO CODE ATL FORM DEFINITIONS

3.1 CODING A FORM DEFINITION

Before attempting to code a form definition, you should be familiar with the material in Chapters 1 and 2 of this manual. Refer to Chapter 5 for complete definitions of the ATL language elements used in this chapter.

It is recommended that the application designer sketch the form in advance, using a form specification sheet as shown in Figure 3-1.

The form is coded using ATL language elements. The ATL statements are entered into a source statement file using the DEC Editor. These statements are then processed by the ATL utility (See Chapter 4) which converts them into a form definition record that can be used by a transaction processor.

3.1.1 Coding an Entry Form Definition

Entry forms are the most common type of form used in TRAX applications. They generally consist of a number of display fields which are used to give general information and instructions, prompt fields which tell the terminal user what data goes into a particular field, and input fields where the user types the data that is to be transmitted to the terminal station for formatting and subsequent processing.

In an entry form, you must also consider which function keys are enabled or disabled, whether replies must be specified as part of the form definition, and the structure of the exchange message that the terminal station constructs from the input fields on a form.

ADCUST is the example form used to describe how to code an entry form. This form is used to capture customer data that is processed by a transaction which adds a record to a customer file. Figure 3-2 shows a Form Specification Sheet for this form of a type that would be prepared by a TRAX application designer.

3.1.1.1 Setting the Default Specifications – If you have determined that certain field attributes are required throughout a form definition, or that certain function keys must always be enabled or disabled, or that a particular character should be used to "clear" (automatically fill) unused character positions, then you can specify these items in a DEFAULT statement at the beginning of your form definition source file.

ATL assumes a set of default conditions that remain in effect unless they are superseded by a clause in the statements used to specify the form definition. A listing of these assumed defaults appears in Section 2.2.7.

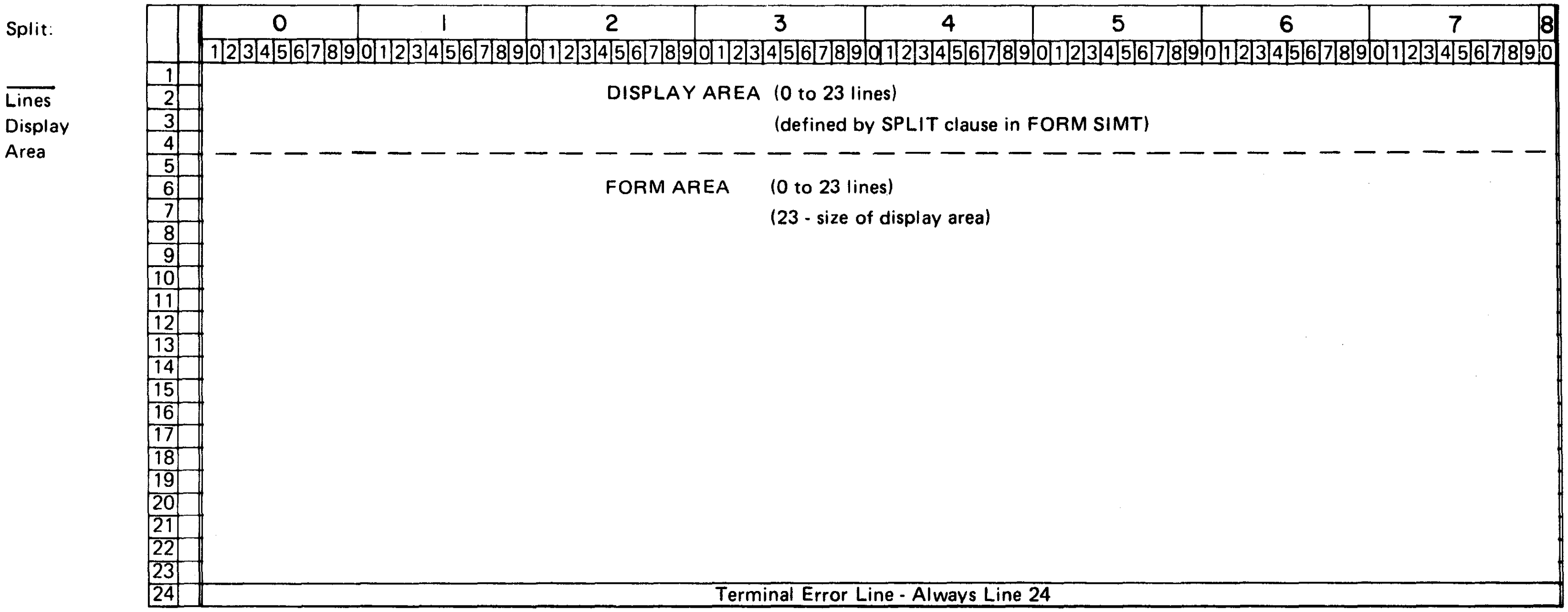
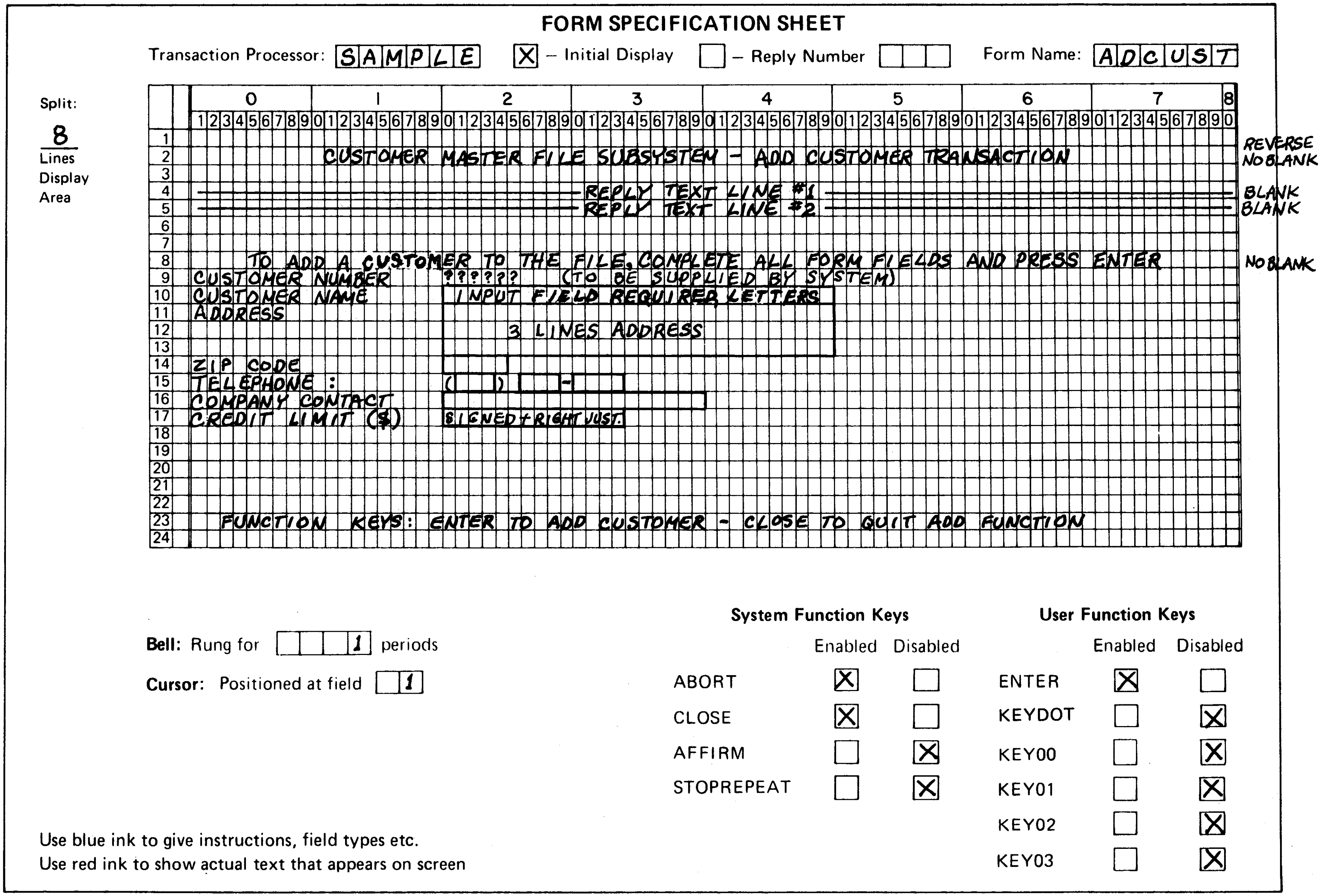

Split:
Lines
Display
Area
0
1
2
3
4
5
6
7
8
DISPLAY AREA (0 to 23 lines)
(defined by SPLIT clause in FORM SIMT)
FORM AREA (0 to 23 lines)
(23 - size of display area)
Terminal Error Line - Always Line 24

Figure 3-1 Sample Form Specification Sheet



FORM SPECIFICATION SHEET

Transaction Processor: SAMPLE ☒ – Initial Display □ – Reply Number □□□ Form Name: ADCUST

Split: 8 Lines Display Area

```
            0         1         2         3         4         5         6         7         8
   1234567890123456789012345678901234567890123456789012345678901234567890123456789 0
 1                                                                                      REVERSE
 2           CUSTOMER MASTER FILE SUBSYSTEM - ADD CUSTOMER TRANSACTION                  NOBLANK
 3
 4  ----------------------------REPLY TEXT LINE #1--------------------------------      BLANK
 5  ----------------------------REPLY TEXT LINE #2--------------------------------      BLANK
 6
 7
 8     TO ADD A CUSTOMER TO THE FILE, COMPLETE ALL FORM FIELDS AND PRESS ENTER          NOBLANK
 9 CUSTOMER NUMBER     ??????  (TO BE SUPPLIED BY SYSTEM)
10 CUSTOMER NAME       [INPUT FIELD REQUIRED LETTERS]
11 ADDRESS             [                             ]
12                     [    3 LINES ADDRESS          ]
13                     [                             ]
14 ZIP CODE            [    ]
15 TELEPHONE :         ([   ]) [   ]-[    ]
16 COMPANY CONTACT     [                    ]
17 CREDIT LIMIT ($)    [SIGNED+RIGHT JUST.]
18
19
20
21
22
23   FUNCTION KEYS: ENTER TO ADD CUSTOMER - CLOSE TO QUIT ADD FUNCTION
24
```

Bell: Rung for □□□1 periods

Cursor: Positioned at field □1

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 3-2 Form Specification Sheet for Entry Form ADCUST.

In the example form ADCUST, the DEFAULT statement took the following form:

```
DEFAULT                                      !To set defaults for whole form
         ENABLE = CLOSE                      !Enable control function keys
         ENABLE = AFFIRM
         CLEAR = " "                         !Set space as default clear character
```

3.1.1.2 **Coding the Form Statement** – The FORM statement allows you to specify:

- The size of a form's Display and Form areas.
- Function keys that are enabled or disabled for the initial form display.
- Defined text-strings for the user function keys.
- If the terminal bell should ring when an initial screen is displayed.

If you are defining a form with Display and Form areas, you can specify the screen division by using the SPLIT clause. In the example form ADCUST, the form is specified with an 8-line Display area, a 15-line Form area (15 = 23 lines – 8 lines), the default error line on Line 24. You have also specified that the initial screen display will cause the terminal bell to ring for two periods (310 ms.).

The following example illustrates the required FORM statement coding to perform the functions listed above:

```
FORM                                         !Form Statement
         SPLIT = 8                           !8 Line Display at Top of Form
         BELL = 2                            !Ring bell for two periods
```

3.1.1.3 **Coding the Display Area** – After you specify the number of lines in the Display area, you may define the contents of the Display fields. Display fields are defined using DISPLAY statements. Suppose that the example form ADCUST requires a title line at the top and two lines that can be used to display Reply message information to the terminal user. The first Display field that you would code is the title line:

```
DISPLAY = 2, 11                              !Indent 11 spaces on Display line 2.
         VALUE = "Customer Master File Subsystem - ",
                 "Add Customer Transaction"
         ATTRIBUTES = REVERSE,               !Highlight in reverse video
                      NOBLANK                !Don't erase during replies
                                             !display on initial screen
```

Specifying the NOBLANK attribute causes the Display field to be written to the initial screen display. Fields specified as NOBLANK remain on all reply screens of that form unless they are cleared or rewritten as part of the reply. The length of this field is defined implicitly by the text-string in the VALUE clause.

After coding the title line, ~~you can then~~ specify the two lines used to display reply information in the following way:

```
DISPLAY = 4,1                              !Skip 1 Line
        LABEL = REPLY.TEXT.A               !Label for 1st REPLY Line
        LENGTH = 80                        !Line covers Full Screen Width

DISPLAY = .+1,1                            !Move to start of next line
        LABEL = REPLY.TEXT.B               !Label for 2nd REPLY Line
        LENGTH = 80                        !Full Screen Width
```

For the purposes of this example, four DISPLAY statements fully specify the contents of the Display area. These four fields are also referenced later by the WRITE clause in the REPLY statement in this form definition. For this reason, these Display fields have been given label names.

The two Display fields labelled REPLY.TEXT.A and REPLY.TEXT.B. have the attribute BLANK assumed as a default. Having the BLANK attribute means that those fields are not written on the initial screen display, but can have information displayed in them as part of the reply screens defined for this form. Since no VALUE clause has been specified for either of these two fields, the LENGTH clause must also be specified. Also note the use of the dot construct to position the field REPLY.TEXT.B one line below REPLY.TEXT.A.

You can also use Display fields to give instructions to the user. Consider the example for ADCUST. To inform the user about how data is entered on the form and sent to the system, you could code a Display field at the bottom of the Display area that looks like the following:

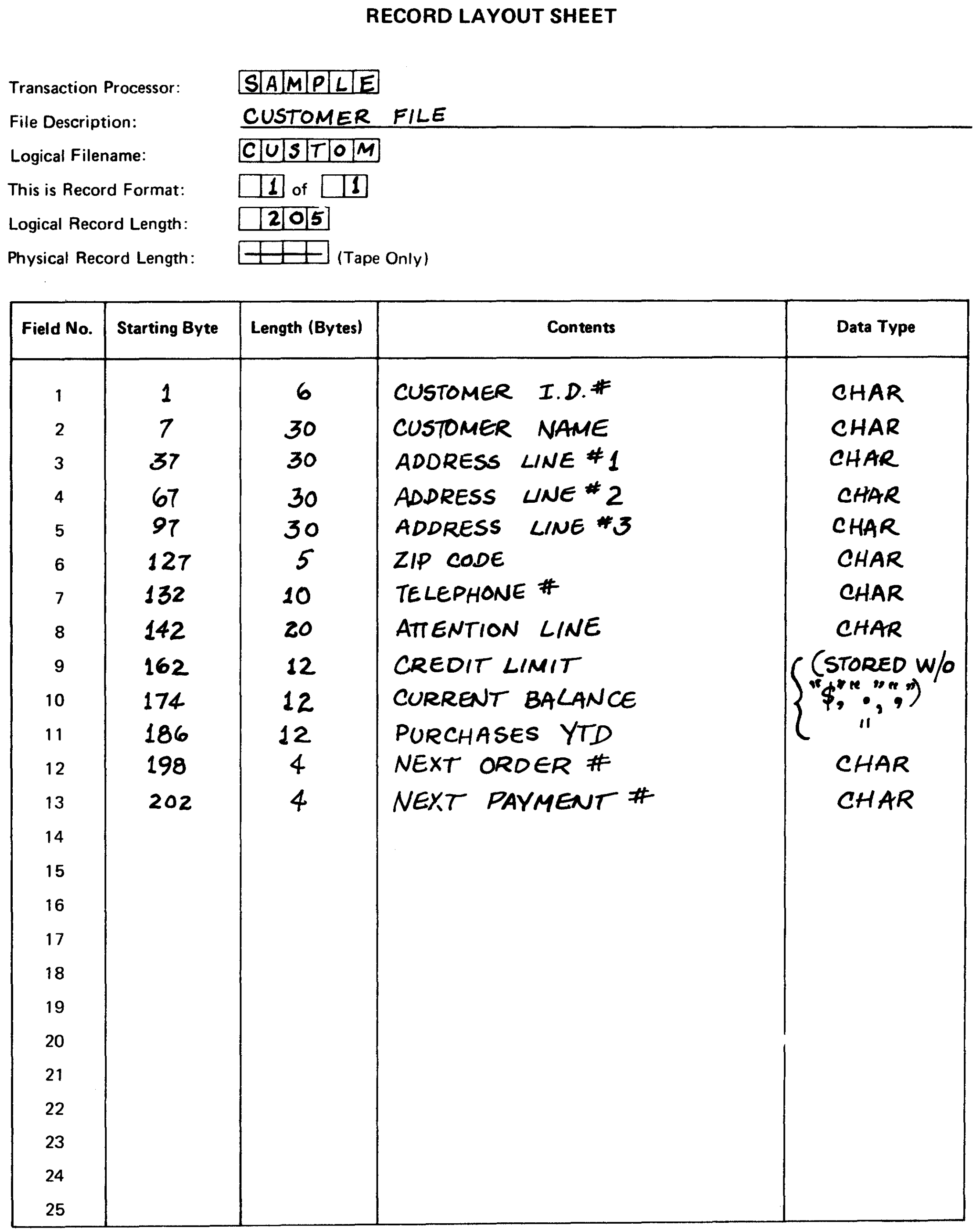
```
    DISPLAY = 8,5                                  !Last Line of Display Area
            VALUE = "To Add a Customer to the File,",
                    "Complete all Form Fields and Press ENTER."
            ATTRIBUTE = NOBLANK                    !Display on the First Screen
            LABEL = INSTR. TEXT                    !Label so you can blank in REPLY
```

Note that this example uses the NOBLANK attribute. This field will appear on the initial screen and, unless modified in a REPLY statement, will also appear on all reply screens. This instruction line appears on line 8 of the terminal screen, immediately above the first line of the Form area. The label clause is specified to allow replies to write into this field.

3.1.1.4 Coding the Form Area – The Form area is the section of the form where the user types in data to be transmitted directly to the terminal station. The two types of statements used in the Form area are the PROMPT statement and the INPUT statement.

To instruct the user in the mechanics of filling a given form, you can specify prompting text that is displayed before an Input field in the Form area. Prompts are specified by the ATL PROMPT statement. When taken as a whole, in this example, the Input fields describe the contents and data structure of the customer record. The customer record layout is shown in Figure 3-3. Since ADCUST is input to an application program that creates a new customer record, only the data fields through CREDIT-LIMIT need be entered on the form ADCUST. The remaining fields will be initialized by the application program.

The following ATL statements are required to specify the Prompt and Input fields used to format a screen display that can be used to input customer data.



RECORD LAYOUT SHEET

Transaction Processor: SAMPLE

File Description: CUSTOMER FILE

Logical Filename: CUSTOM

This is Record Format: 1 of 1

Logical Record Length: 205

Physical Record Length: (Tape Only)

Field No.	Starting Byte	Length (Bytes)	Contents	Data Type
1	1	6	CUSTOMER I.D.#	CHAR
2	7	30	CUSTOMER NAME	CHAR
3	37	30	ADDRESS LINE #1	CHAR
4	67	30	ADDRESS LINE #2	CHAR
5	97	30	ADDRESS LINE #3	CHAR
6	127	5	ZIP CODE	CHAR
7	132	10	TELEPHONE #	CHAR
8	142	20	ATTENTION LINE	CHAR
9	162	12	CREDIT LIMIT	(STORED W/O
10	174	12	CURRENT BALANCE	"$," ".", ",")
11	186	12	PURCHASES YTD	"
12	198	4	NEXT ORDER #	CHAR
13	202	4	NEXT PAYMENT #	CHAR
14				
15				
16				
17				
18				
19				
20				
21				
22				
23				
24				
25				

Figure 3-3 Customer File Record Layout

Note that the comments in the ATL statements briefly describe the purpose of the line of ATL code. Chapter 5 has detailed descriptions of each ATL language element.

```
PROMPT = 1,1                          !Move to Line 1, Column 1 of Form Area
        VALUE = "Customer Number"     !Identify the empty field that follows

INPUT = .,.+4                         !Field starts 4 spaces past prompt
        LABEL = CUSTOMER.NUMBER       !Assign LABEL for future reference
        VALUE = "??????"              !Fill with ?
        ATTRIB = REVERSE,             !Highlight in Reverse video
                 NOMODIFY             !Protect field from any entry
```

Notice the way the dot (.) construct is used in this statement to specify the same row as the preceding statement and a column position 4 spaces after the end of the preceding field.

```
PROMPT = .,30                         !Skip to Column 30 on this line
        LENGTH = 30                   !Define length for future reference
        VALUE = "(To be Supplied by System)"
        LABEL = CUSTNO.TEXT           !Label for future reference
```

In the prompt statement that defines CUSTNO.TEXT, the length implied by the VALUE statement is 26 characters. This length is overridden by the LENGTH clause.

```
PROMPT = .+1,1                        !Next line, column 1
        VALUE = "Customer Name"       !Prompt text identifies input field

INPUT = .,20                          !Field starts on same line, column 20
        LENGTH = 30                   !Define maximum name length
        LABEL = CUSTOMER.NAME         !Label field for later use
        ATTRIB = REVERSE,             !Highlight with Reverse Video
                 REQUIRED,            !Force user to Enter Data
                 LETTERS              !Restrict characters in field
```

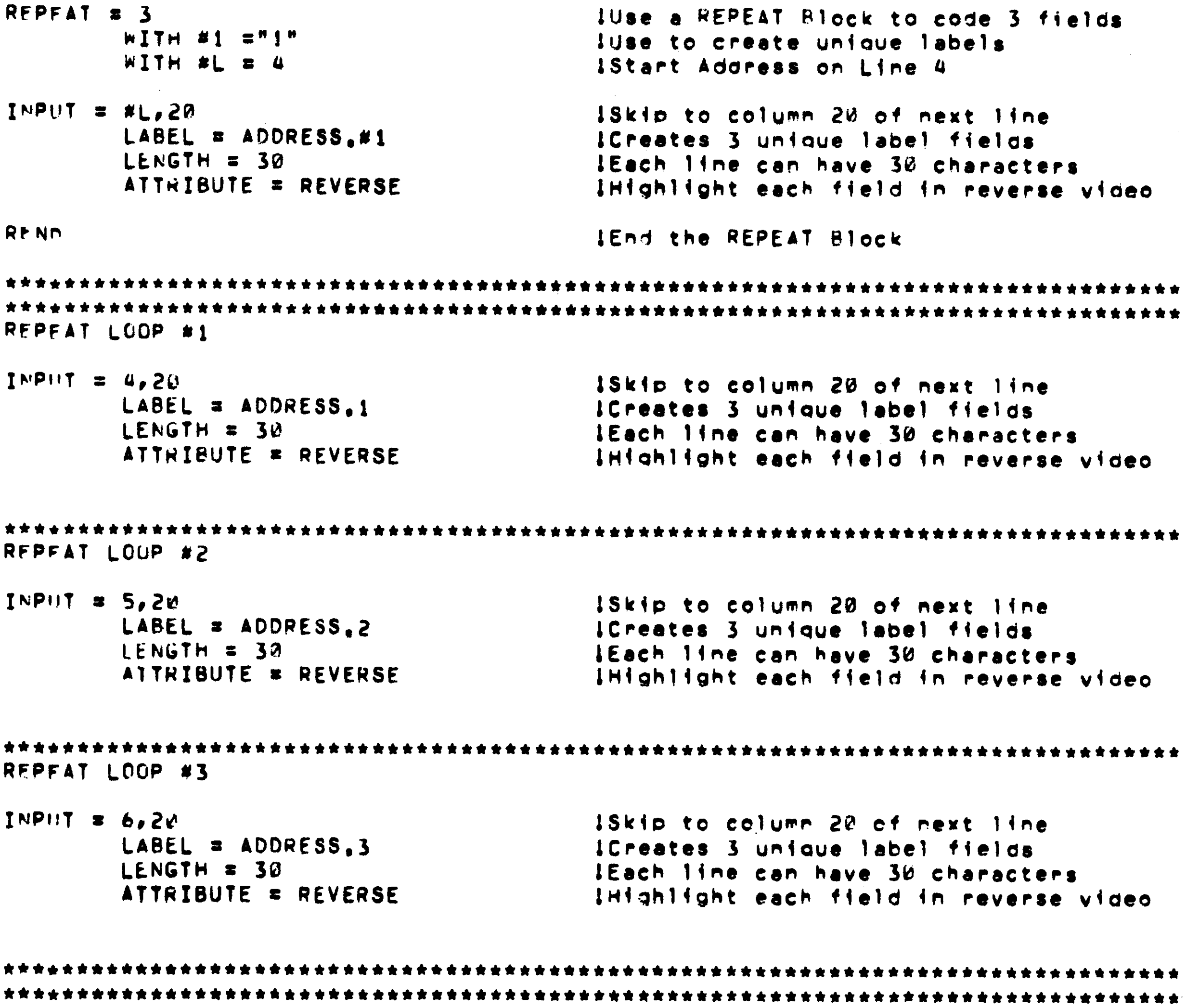
Specifying the REVERSE ATTRIBUTE for an Input field clearly defines the size of the field on the screen.

```
PROMPT = .+1,1                        !Skip a line, return to column 1
        VALUE = "Address"             !Prompt for 3-line address
```

The ATL REPEAT Statement can be used to speed the task of coding forms. In the current example, a 3-line address is required, each line consisting of a 30-character address field. The following examples shows how the REPEAT statment is used to specify three identical address lines:

```
REPEAT = 3                                    !Use a REPEAT Block to code three fields
          WITH #1 = "1"                       !Use to create unique labels
          WITH #L = 4                         !Start address on Line 4

INPUT = #L,20                                 !Skip to column 20 of line
          LABEL = ADDRESS.#1                  !Creates unique label
          LENGTH = 30                         !Line can have 30 characters
          ATTRIBUTE = REVERSE                 !Highlight field in reverse video

REND                                          !End the REPEAT Block
```

The statement listing shown in Figure 3-4 illustrates how the set of three INPUT statements describing the address fields is created by the ATL utility when it processes the REPEAT block specified in the preceding example. Notice how the dummy variables #1 and #L are replaced by values specified in the WITH clause. See the explanation of the REPEAT statement in Chapter 5 for a complete description of this feature.



```
REPEAT = 3                                    !Use a REPEAT Block to code 3 fields
        WITH #1 ="1"                          !Use to create unique labels
        WITH #L = 4                           !Start Address on Line 4

INPUT = #L,20                                 !Skip to column 20 of next line
        LABEL = ADDRESS.#1                    !Creates 3 unique label fields
        LENGTH = 30                           !Each line can have 30 characters
        ATTRIBUTE = REVERSE                   !Highlight each field in reverse video

REND                                          !End the REPEAT Block

********************************************************************
********************************************************************
REPEAT LOOP #1

INPUT = 4,20                                  !Skip to column 20 of next line
        LABEL = ADDRESS.1                     !Creates 3 unique label fields
        LENGTH = 30                           !Each line can have 30 characters
        ATTRIBUTE = REVERSE                   !Highlight each field in reverse video


********************************************************************
REPEAT LOOP #2

INPUT = 5,20                                  !Skip to column 20 of next line
        LABEL = ADDRESS.2                     !Creates 3 unique label fields
        LENGTH = 30                           !Each line can have 30 characters
        ATTRIBUTE = REVERSE                   !Highlight each field in reverse video


********************************************************************
REPEAT LOOP #3

INPUT = 6,20                                  !Skip to column 20 of next line
        LABEL = ADDRESS.3                     !Creates 3 unique label fields
        LENGTH = 30                           !Each line can have 30 characters
        ATTRIBUTE = REVERSE                   !Highlight each field in reverse video


********************************************************************
********************************************************************
```

Figure 3-4 ATL Utility Listing of REPEAT Block

```
PROMPT = .+1,1                              !Next line, column 1
         VALUE = "Zip Code"                 !Identify the Input field that follows

INPUT = .,20                                !Skip to column 20 of current line
         LABEL = ZIP.CODE                   !Label field for use by MESSAGE st'tment
         LENGTH = 5                         !Zip Code is five digits in U.S.A.
         ATTRIB = REVERSE,                  !Highlight in reverse video
                  NUMERIC,                  !Restrict character set
                  FULL,                     !Must type all five digits
                  REQUIRED                  !User cannot skip this field
```

In the INPUT statement defining the ZIP.CODE field, the ATTRIBUTES NUMERIC and FULL prevent the user from entering an obviously invalid ZIP CODE. In addition, the REQUIRED ATTRIBUTE forces the user to complete this field.

Using ATL, you can reduce the terminal user's effort by specifying fields in any level of detail that you require. Take the example of the telephone number with three distinct code values that must be completed. You can code the Telephone number in the manner shown on the example form ADCUST and produce a screen display of the form:

Telephone number: (617) 493-2211

The coding required to achieve this segmented display is:

```
PROMPT = .+1,1                              !Skip to column 1 of next line
         VALUE = "Telephone  :          ("  !Prompt tag for telephone line

INPUT = .,.                                 !Start immediately
         LABEL = AREA.CODE                  !Label for MESSAGE statement
         LENGTH = 3                         !3-digit area code in U.S.A.
         ATTRIBUTE = TAB,REVERSE,           !Allow auto-tab and reverse video
                     NUMERIC,FULL,          !Restrict Character set and force fill
                     REQUIRED               !User must enter data

PROMPT = .,. VALUE = ") "                   !Close area code bracket

INPUT = .,.                                 !Continue immediately
         LABEL = TEL.EXCHANGE               !Label for MESSAGE statement
         LENGTH = 3                         !3 digit exchange in U.S.A.
         ATTRIBUTE = TAB,REVERSE,           !Allow auto-tab and reverse video
                     NUMERIC,FULL,          !Restrict Character set and force fill
                     REQUIRED               !User must enter data

PROMPT = .,. VALUE = "-"                    !Hyphen exchange and extension

INPUT = .,.                                 !Continue immediately
         LABEL = TEL.EXTENSION              !Label for MESSAGE statement
         LENGTH = 4                         !4-digit extension in U.S.A.
         ATTRIBUTE = REVERSE,               !Highlight with reverse video
                     NUMERIC,FULL,          !Restrict character set and force fill
                     REQUIRED               !User must enter data
```

Defining the telephone field in this manner allows the user to fill in all three number fields by simply typing the ten characters of the telephone number in succession. Since the Input fields were specified with the TAB attribute, the terminal automatically moves the cursor to the exchange and extension fields.

The last fields to be defined in the Form area are the Company Contract and Credit Limit Amount fields. They are specified in the following manner:

```
PROMPT = .+1,1                          !Next line, column 1
          VALUE = "Company Contact"     !Identify the following Input field

INPUT = .,20                            !Skip to Column 20 on Same Line
          LABEL = ATTENTION             !Label for MESSAGE statement
          LENGTH = 20                   !Allow 20-character name
          ATTRIB = REVERSE              !Highlight empty field in reverse video

PROMPT = .+1,1                          !Skip a line; go to column 1
          Value = "Credit Limit ($)"    !Identify the following Input field

INPUT = .,20                            !Column 20 of this line
          LABEL = CREDIT.LIMIT          !Label for MESSAGE statement
          LENGTH = 12                   !Allows 12 spaces for amount
          ATTRIB = REVERSE,SIGNED,      !Highlight, allow commas and periods
                   RIGHT                !Right-justify for easier typing
          CLEAR = "0"                   !Clear character for this field is 0
```

Notice that, in the last INPUT statement, the CLEAR character of "0" has been specified and the field has been defined with the RIGHT ATTRIBUTE. Specifying a numeric field in this way is very helpful to the terminal user, since dollar values can be entered into the field in exactly the same way they would be typed on a cash register or calculator.

After you define all the INPUT fields on the screen, then you must tell the user what he should do with the information that has just been typed on the form. A prompt field allows you to give instructions on the bottom of this form.

```
PROMPT = 15,3                           !Center on last Form area line
          LABEL = KEY.PROMPT            !Label so we can change in replies
          LENGTH = 75                   !Define field length
          VALUE = "Function Keys:",     !Text of initial form display
                  "ENTER to Add Customer",
                  "- CLOSE to quit Add Function"
          ATTRIBUTE = REVERSE           !Highlight in reverse video
```

3.1.1.5 Defining the Reply Screens – Having defined all the fields in the Display and Form areas, you next define the possible Reply screens that may occur during the processing of this particular form. The application designer specifies the REPLY screen layout, and the format of the message sent back to the form as part of each reply message. In the case of the example form, two REPLY screens are specified: Reply 1 and Reply 2. Reply #1 writes the message "★★TRANSACTION

COMPLETE★★" onto the screen in the Display area and inserts the Customer Number into the previously restricted field

Reply #2 allows you to insert error messages and user instructions into the Display area fields REPLY.TEXT.A and REPLY.TEXT.B. These modifications occur when a REPLY message is sent from a TST with the value 2 specified as the REPLY-NUMBER.

The form and response message specifications for Reply 1 and Reply 2 are shown in Figures 3-5a, 3-5b, 3-6a, and 3-6b.

Reply #1 is specified by the following statement:

```
    REPLY = 1
            DISABLE = ENTER                         !Disable ENTER key for this reply
            WRITE = REPLY.TEXT.B," ★★★ TRANSACTION COMPLETE ★★★ "
            WRITE = INSTR.TEXT,FILL (" ",72)        !Blank original instructions
            WRITE = CUSTNO.TEXT,FILL (" ",30)       !Blank Number Text field
            WRITE = KEY.PROMPT, "Function Keys: Press AFFIRM to Add Another",
                    "Customer – Press CLOSE to quit"
            WRITE = CUSTOMER.NUMBER , REQUEST (1,6)
            !This WRITE Clause completes the customer information. !
```

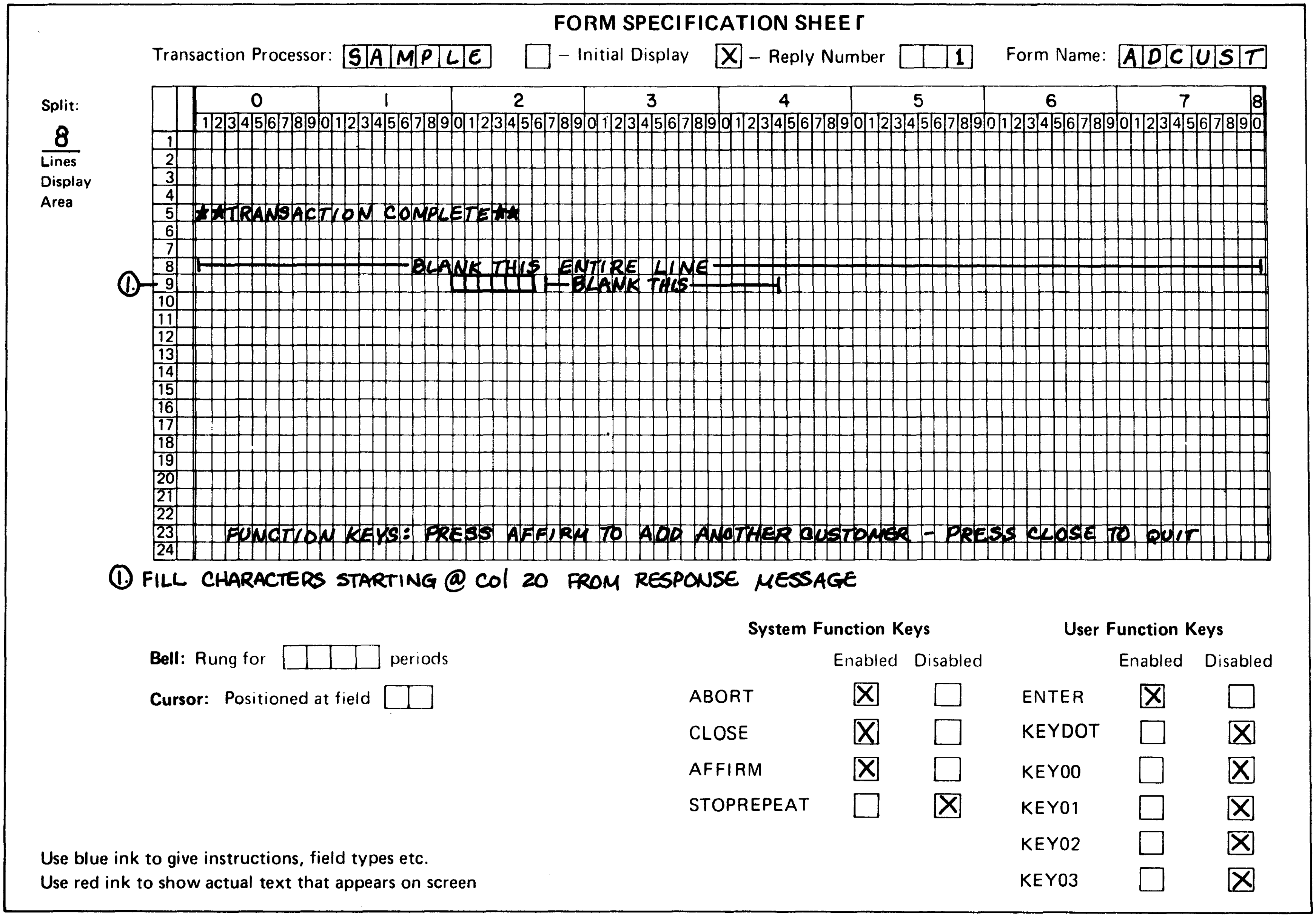
In the statement defining REPLY #1, the REQUEST function is used to extract data from the reply response message and placed in the CUSTOMER.NUMBER field. Also, in this reply, nothing is written to the BLANK Display field REPLY.TEXT.A. As a result, this field is blanked in the reply. This demonstrates one feature of the BLANK ATTRIBUTE. Note by contrast, that the two PROMPT fields must be explicitly blanked by use of the FILL function in the REPLY statement.

To code REPLY #2 simply, fill the Reply Text fields using the data in the Reply message:

```
    REPLY = 2
            DISABLE = AFFIRM                        !Disable AFFIRM Key
            WRITE = REPLY.TEXT.A , REQUEST (1,80)   !Fill 1st 80 characters
            WRITE = REPLY.TEXT.B , REQUEST (81,80)  !Fill rest of message
            WRITE = INSTR.TEXT,FILL (" ",72)        !Blank original instructions
```

3.1.1.6 Defining the Message Statement – The last step in the form definition process is the specification of the exchange message. Figure 3-7 shows the exchange message layout planned by the application designer.

Using the ATL MESSAGE statement you can specify the exchange message by specifying the field labels you assigned to Input fields.



FORM SPECIFICATION SHEET

Transaction Processor: SAMPLE □ – Initial Display ☒ – Reply Number 1 Form Name: ADCUST

Split: 8 Lines Display Area

Line	Text
1	
2	
3	
4	
5	**TRANSACTION COMPLETE**
6	
7	
8	├──BLANK THIS ENTIRE LINE──┤
①─9	□□□□□□ ├─BLANK THIS──┤
10–22	
23	FUNCTION KEYS: PRESS AFFIRM TO ADD ANOTHER CUSTOMER - PRESS CLOSE TO QUIT
24	

① FILL CHARACTERS STARTING @ COl 20 FROM RESPONSE MESSAGE

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 3-5a Form Specification for Reply #1 of ADCUST

RESPONSE MESSAGE SPECIFICATION SHEET

Transaction Processor SAMPLE

Transaction Name ADDCUS

Exchange Label ADDEX1

Type of Message ☒ – REPLY (Activates reply no. ☐☐1)

☐ – PRCEED

☐ – STPRPT

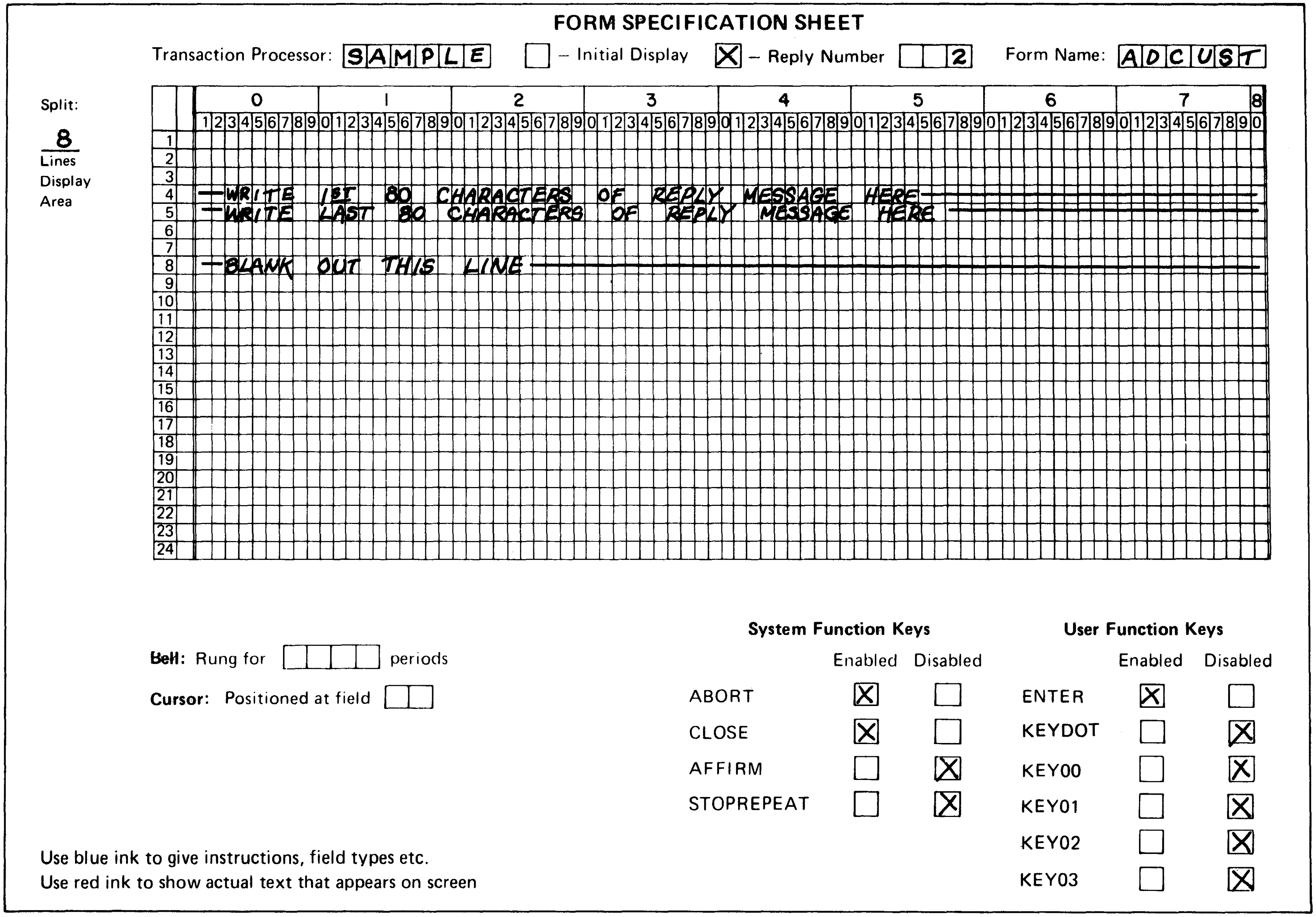
☐ – CLSTRN

☐ – ABORT (Activates reply no. ☐☐☐)

☐ – TRNSFR (To exchange ☐☐☐☐☐☐)

Field No.	Starting Byte	Length (Bytes)	Contents
1	1	6	CUSTOMER NUMBER ASSIGNED BY TST
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 3-5b Response Message Specification for Reply #1



FORM SPECIFICATION SHEET

Transaction Processor: SAMPLE □ – Initial Display ☒ – Reply Number 2 Form Name: ADCUST

Split: 8 Lines Display Area

1	
2	
3	
4	—WRITE 1ST 80 CHARACTERS OF REPLY MESSAGE HERE—
5	—WRITE LAST 80 CHARACTERS OF REPLY MESSAGE HERE —
6	
7	
8	—BLANK OUT THIS LINE —
9	
10	
11	
12	
13	
14	
15	
16	
17	
18	
19	
20	
21	
22	
23	
24	

Bell: Rung for □□□□ periods

Cursor: Positioned at field □□

System Function Keys

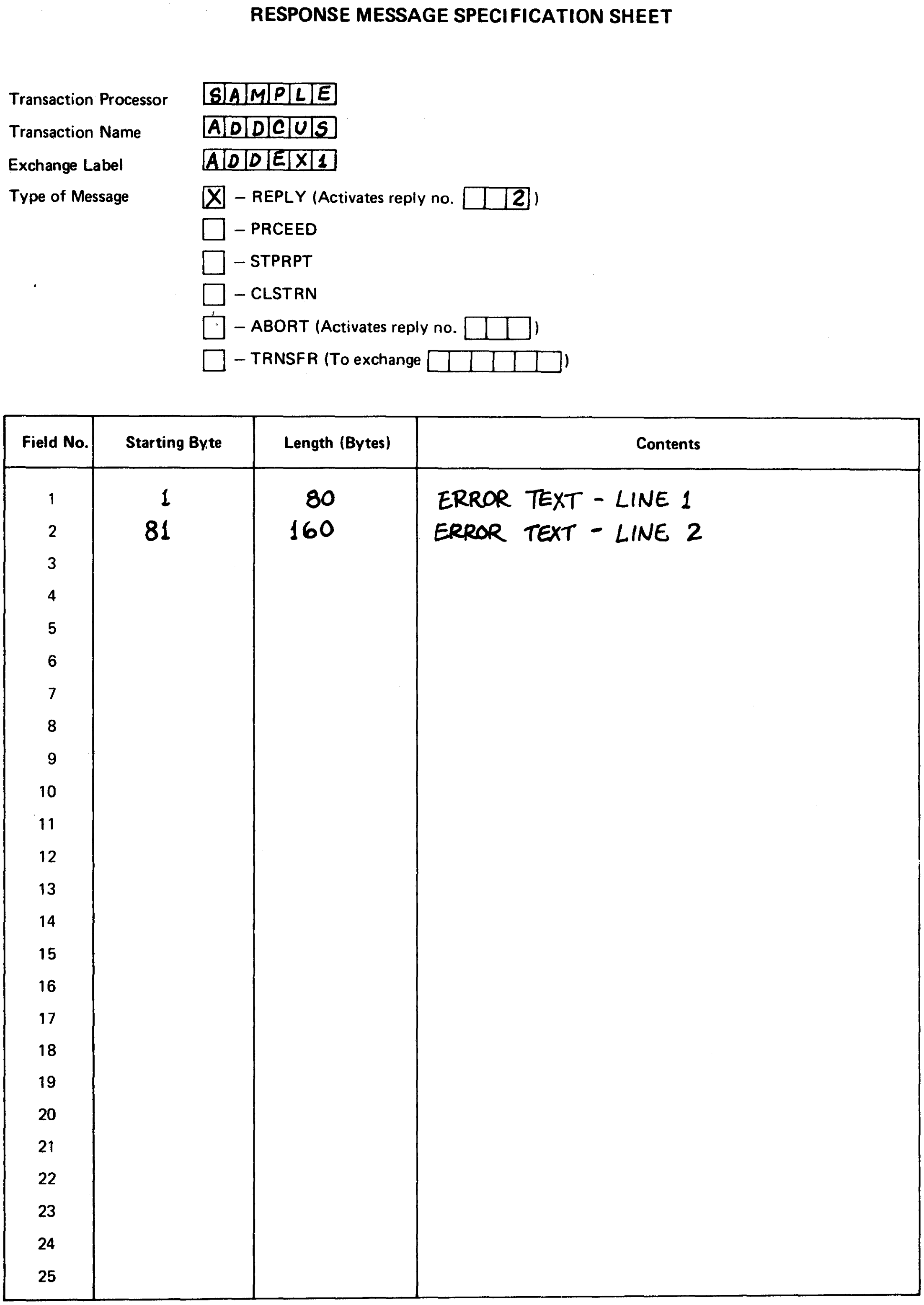
	Enabled	Disabled
ABORT	☒	□
CLOSE	☒	□
AFFIRM	□	☒
STOPREPEAT	□	☒

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 3-6a Form Specification for Reply #2 of ADCUST



RESPONSE MESSAGE SPECIFICATION SHEET

Transaction Processor SAMPLE

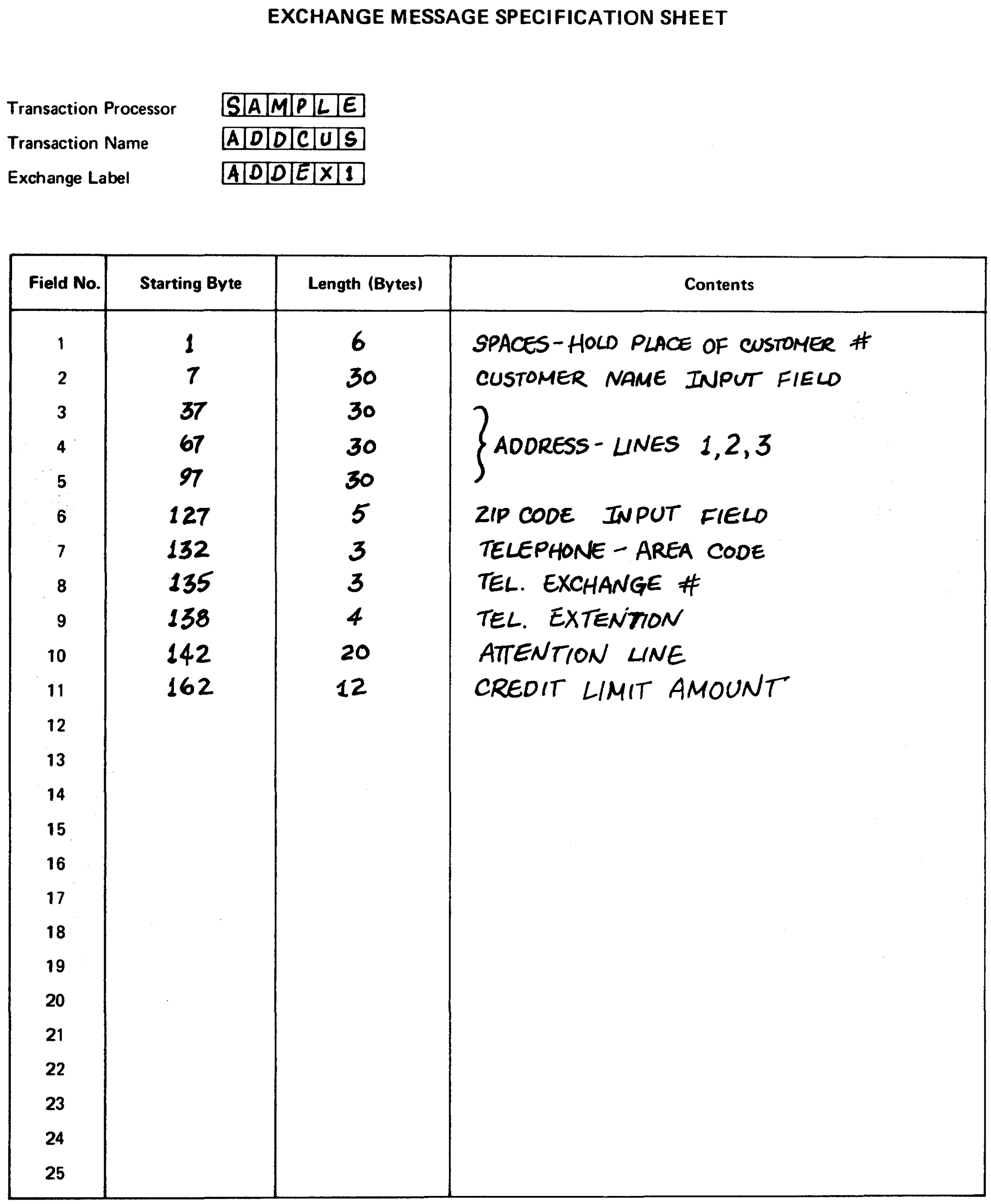
Transaction Name ADDCUS

Exchange Label ADDEX1

Type of Message ☒ – 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 TEXT - LINE 1
2	81	160	ERROR TEXT - LINE 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 3-6b Response Message Specification for Reply #2



EXCHANGE MESSAGE SPECIFICATION SHEET

Transaction Processor SAMPLE

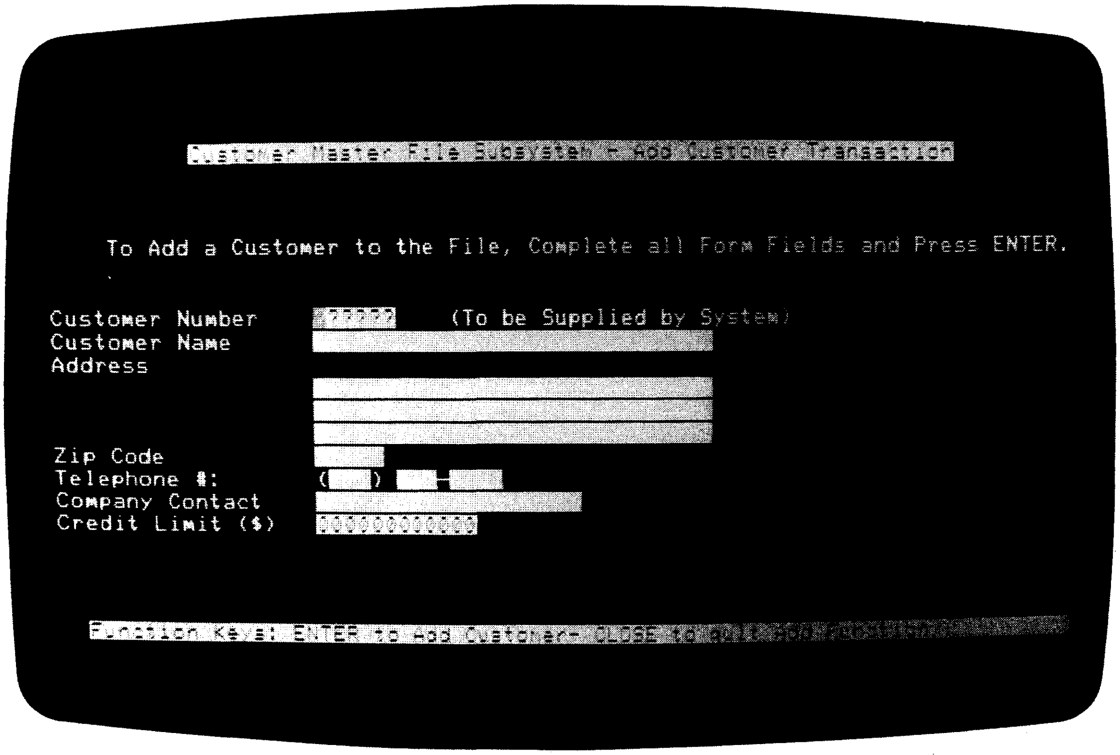
Transaction Name ADDCUS

Exchange Label ADDEX1

Field No.	Starting Byte	Length (Bytes)	Contents
1	1	6	SPACES-HOLD PLACE OF CUSTOMER #
2	7	30	CUSTOMER NAME INPUT FIELD
3	37	30	
4	67	30	ADDRESS- LINES 1,2,3
5	97	30	
6	127	5	ZIP CODE INPUT FIELD
7	132	3	TELEPHONE - AREA CODE
8	135	3	TEL. EXCHANGE #
9	138	4	TEL. EXTENTION
10	142	20	ATTENTION LINE
11	162	12	CREDIT LIMIT AMOUNT
12			
13			
14			
15			
16			
17			
18			
19			
20			
21			
22			
23			
24			
25			

Figure 3-7 Exchange Message Layout for Add Customer Exchange

```
MESSAGE = 7                                       !Start filling in 7th character position
                                                  !First six spaces align msg with record.
       VALUE =                                    !Value clause defines the data to be
                                                  !put into the message.
                   CUSTOMER.NAME,
                   ADDRESS.1,
                   ADDRESS.2,
                   ADDRESS.3,
                   ZIP.CODE,
                   AREA.CODE, TEL.EXCHANGE, TEL.EXTENSION,
                   ATTENTION,
                   CREDIT.LIMIT
END                                               !End of this Form Definition
```



```
        Customer Master File Subsystem - Add Customer Transaction


   To Add a Customer to the File, Complete all Form Fields and Press ENTER.

 Customer Number     ?????     (To be Supplied by System)
 Customer Name
 Address



 Zip Code
 Telephone #:        (   )    -
 Company Contact
 Credit Limit ($)

 Function Keys: ENTER to Add Customer- CLOSE to [illegible]
```

Figure 3-8 Initial Screen for Form ADCUST

3.1.1.7 Using the KEY and KEYCAP Parameters – ATL allows you to specify a text-string for each of the six data entry function keys. This keycap text-string can be specified as part of an exchange message, and transmitted to a TST where it can be used to differentiate the processing required by the exchange message.

Consider an example where a terminal user wants to examine the records in a customer file. The terminal user enters the customer number or name on the first form, and the customer record is displayed on the next form. At this point, the user can:

1. Press the ENTER key to see the next record in the file.
2. Press KEYDOT (SHIFT + ".") to make a hard copy of the data currently displayed on the screen.

Several steps are required to implement this functionality. In the simplest case, the default keycap text-strings are used, the exchange message is specified as follows:

```
MESSAGE = 1
          VALUE = KEY
```

In the TST to which this exchange message is routed, the programmer must specify an exchange message map of six ASCII characters. The TST tests this data item. If the value is "ENTER", the TST branches to a routine that reads the next record. If the value is "KYDOT", the TST branches to a routine that sends a REPORT message to an output-only terminal.

Figure 3-9 shows the coding from a TST that maps on to the exchange message, and tests the function key input to determine the correct processing path.

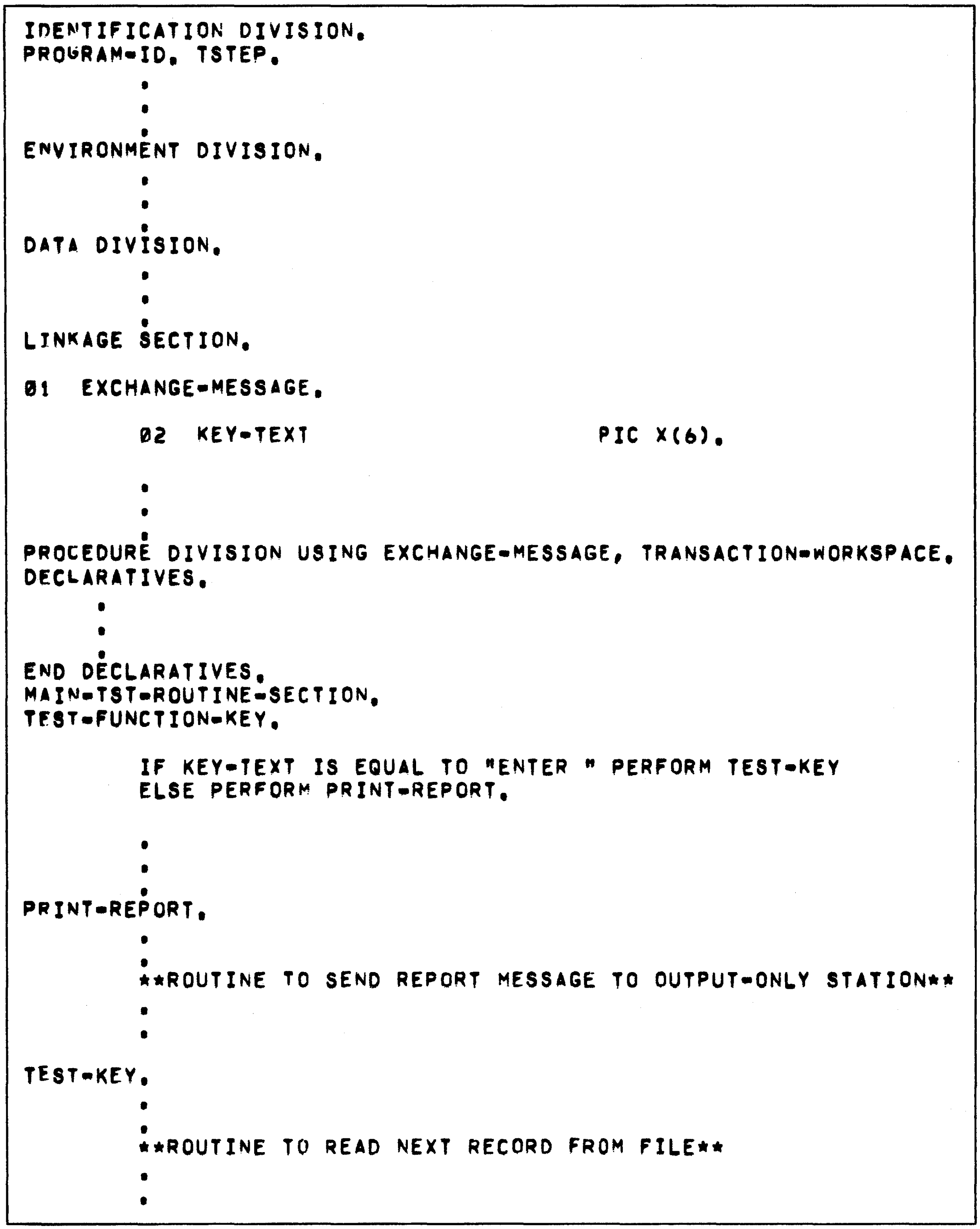
In some cases, you may want the data entry function keys to have different text strings associated with them. Suppose you were coding a form used to list goods that are sent from a warehouse to one of five different retail outlets. If you define the keycap text strings to contain the outlet name and location, you can save work for both the data entry clerk and the TST. In this example, suppose the outlets were located in the following cities:

Nashua, NH 03060
Maynard, MA 01754
Boston, MA 02108
Worcester, MA 01608
Merrimack, NH 03054

If you specified these city names in a FORM statement as the set of keycap text-strings in the form definition, the terminal user using that form can place the city, state, and zip code in the exchange message with a single keystroke. The following example shows how the KEYCAP clause is used.

```
FORM
          SPLIT   = 8
          KEYCAP  = KEYDOT,"Nashua, NH 03060"
          KEYCAP  = KEY0,"Maynard, MA 01754
          KEYCAP  = KEY1,"Boston, MA 02108
          KEYCAP  = KEY2,"Worcester, MA 01608"
          KEYCAP  = KEY3,"Merrimack, NH 03054"
          DISABLE = ENTER
```

On this form, you would use the KEY parameter in the MESSAGE statement to insert the appropriate text-string into the exchange message. For example:



```
IDENTIFICATION DIVISION.
PROGRAM-ID. TSTEP.
        .
        .
        .
ENVIRONMENT DIVISION.
        .
        .
        .
DATA DIVISION.
        .
        .
        .
LINKAGE SECTION.

Ø1  EXCHANGE-MESSAGE.

        Ø2  KEY-TEXT                     PIC X(6).

        .
        .
        .
PROCEDURE DIVISION USING EXCHANGE-MESSAGE, TRANSACTION-WORKSPACE.
DECLARATIVES.
     .
     .
     .
END DECLARATIVES.
MAIN-TST-ROUTINE-SECTION.
TEST-FUNCTION-KEY.

        IF KEY-TEXT IS EQUAL TO "ENTER " PERFORM TEST-KEY
        ELSE PERFORM PRINT-REPORT.

        .
        .
        .
PRINT-REPORT.
        .
        .
        **ROUTINE TO SEND REPORT MESSAGE TO OUTPUT-ONLY STATION**
        .
        .

TEST-KEY.
        .
        .
        **ROUTINE TO READ NEXT RECORD FROM FILE**
        .
        .
```

Figure 3-9 TST Coding (COBOL) to Test Function Key Input

```
MESSAGE = 1
          VALUE =  "Receiving Department",
                   "TRAX Coin and Stamp",
                   KEY,
                   .
                   .
                   .
```

You should note that the keycap text-strings in this example have differing lengths. In order to pass a fixed length exchange message to the TST, ATL pads all keycap text-strings to the length of the longest string defined. In this example, the KEY parameter represents a string of 19 characters. The padding character used by ATL is the null character, OCTAL 000.

3.2 CODING A REPORT FORM DEFINITION

You use Report forms to display information on an output-only terminal. The example form to illustrate this section is a hard-copy Report form that prints Customer Account information. Figure 3-10 shows the form layout envisioned by the application designer.

3.2.1 Coding the FORM Statement

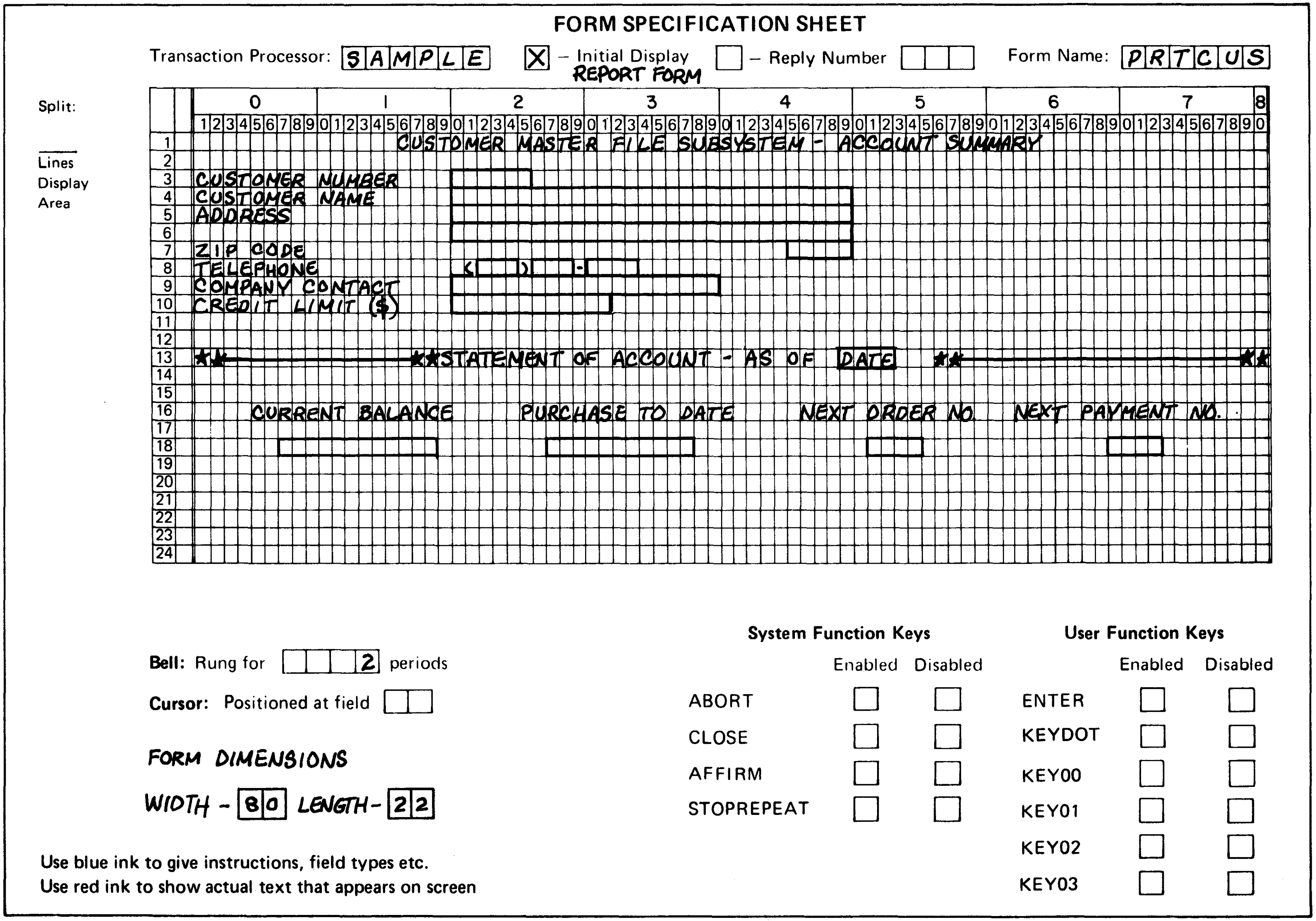
When you use a Report form on a hard-copy terminal, you may find it useful to specify the dimensions of the form, page length and column width, in the FORM statement. The FORM statement allow you to define the form so that alignment with preprinted paper stock is possible.

You specify the page size for a REPORT form through clauses in the FORM statement. For example, a 22-line by 80-column report would have the following FORM statement:

```
FORM
          LENGTH = 22                     !22 lines per form
          WIDTH = 80                      !80 columns per line
          BELL = 2                        !Ring bell twice
```

A form specified in this fashion prints one record every 22 lines (an 11-inch form has 66 lines), the hard-copy terminal bell sounds for about a half second when the form is printed.

Coding the remainder of the Report form is a matter of specifying the location and contents of Print fields using PRINT statements and VALUE clauses. You must also use the REQUEST construct to extract information from the Report message sent by a TST that initiated the printing of the report. In the example used in this section, a TST sends a 205-character Report message that contains all fields in the Customer Record. Figure 3-11 describes the Report message sent to this form.



FORM SPECIFICATION SHEET

Transaction Processor: SAMPLE ☒ – Initial Display REPORT FORM □ – Reply Number □□□ Form Name: PRTCUS

Split:

Lines Display Area

```
    0         1         2         3         4         5         6         7        8
    1234567890123456789012345678901234567890123456789012345678901234567890123456789
 1                 CUSTOMER MASTER FILE SUBSYSTEM - ACCOUNT SUMMARY
 2
 3  CUSTOMER NUMBER
 4  CUSTOMER NAME
 5  ADDRESS
 6
 7  ZIP CODE
 8  TELEPHONE          (    )    -
 9  COMPANY CONTACT
10  CREDIT LIMIT ($)
11
12
13  ★★———————————————★★STATEMENT OF ACCOUNT - AS OF DATE  ★★———————————————————————★★
14
15
16      CURRENT BALANCE     PURCHASE TO DATE     NEXT ORDER NO.  NEXT PAYMENT NO.
17
18
19
20
21
22
23
24
```

Bell: Rung for □□□2 periods

Cursor: Positioned at field □□

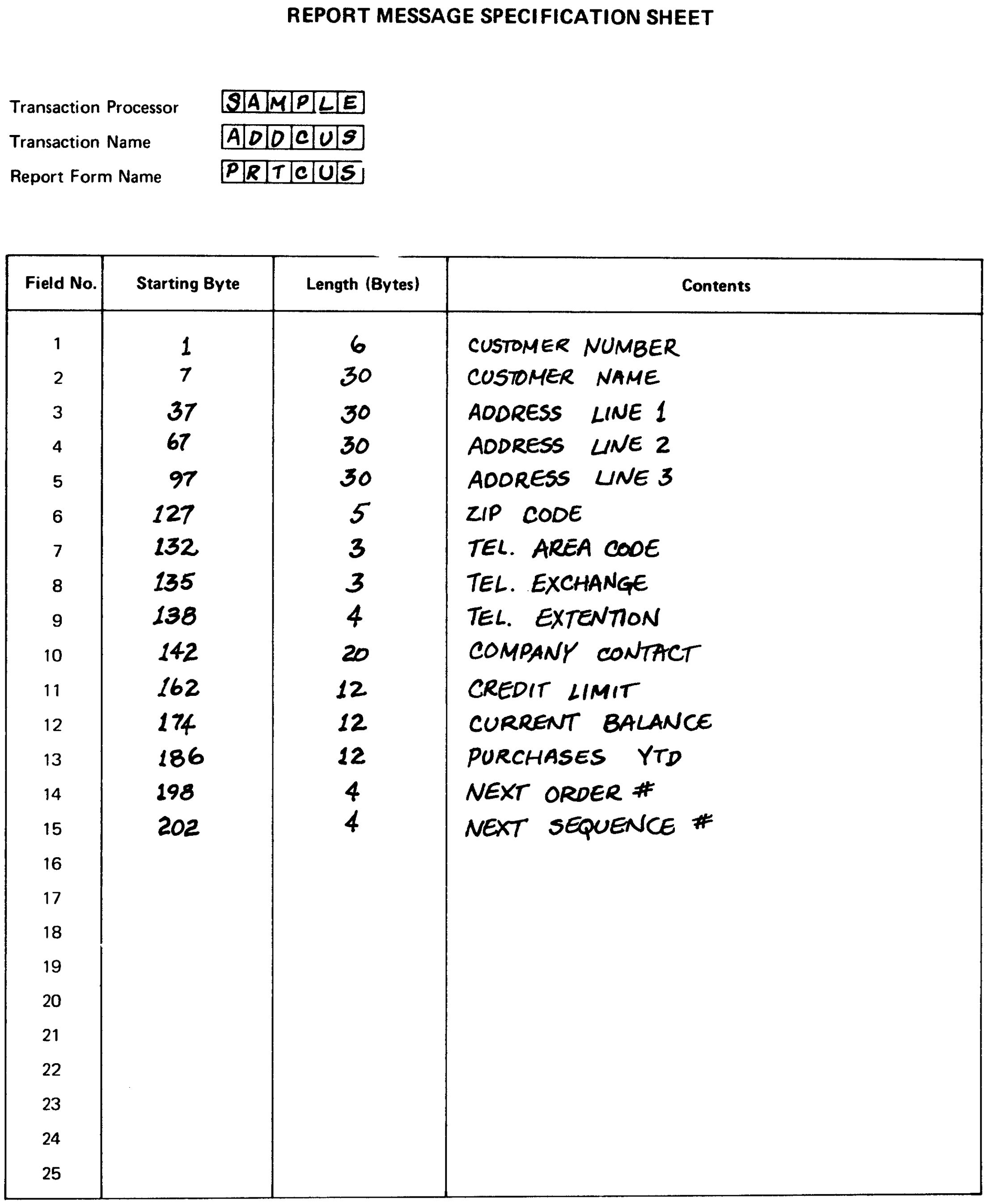
FORM DIMENSIONS

WIDTH - 80 LENGTH - 22

System Function Keys	Enabled	Disabled	User Function Keys	Enabled	Disabled
ABORT	□	□	ENTER	□	□
CLOSE	□	□	KEYDOT	□	□
AFFIRM	□	□	KEY00	□	□
STOPREPEAT	□	□	KEY01	□	□
			KEY02	□	□
			KEY03	□	□

Use blue ink to give instructions, field types etc.
Use red ink to show actual text that appears on screen

Figure 3-10 Report Form Specification Sheet – "PRTCUS"



REPORT MESSAGE SPECIFICATION SHEET

Transaction Processor SAMPLE

Transaction Name ADDCUS

Report Form Name PRTCUS

Field No.	Starting Byte	Length (Bytes)	Contents
1	1	6	CUSTOMER NUMBER
2	7	30	CUSTOMER NAME
3	37	30	ADDRESS LINE 1
4	67	30	ADDRESS LINE 2
5	97	30	ADDRESS LINE 3
6	127	5	ZIP CODE
7	132	3	TEL. AREA CODE
8	135	3	TEL. EXCHANGE
9	138	4	TEL. EXTENTION
10	142	20	COMPANY CONTACT
11	162	12	CREDIT LIMIT
12	174	12	CURRENT BALANCE
13	186	12	PURCHASES YTD
14	198	4	NEXT ORDER #
15	202	4	NEXT SEQUENCE #
16			
17			
18			
19			
20			
21			
22			
23			
24			
25			

Figure 3-11 Report Message Specification Form

In the following set of PRINT statement, notice how the fields are filled from this requesting message through the use of the REQUEST construct. Also, note the use of the dot construct for positioning fields on the Report form and for extracting data from the requesting message.

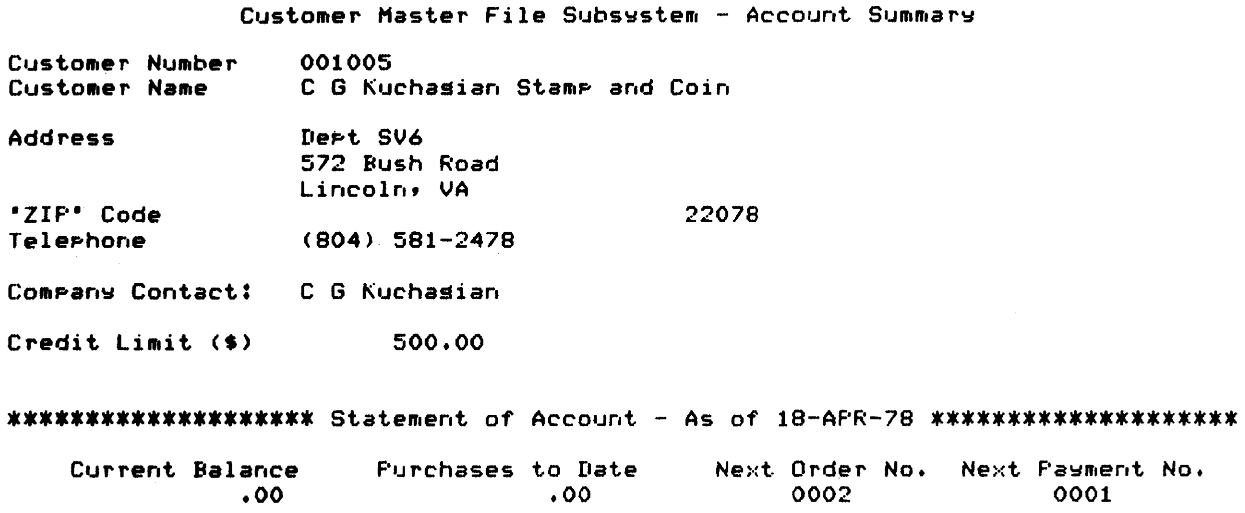
```
PRINT = 1,16
        VALUE =“Customer Master File Subsystem–Account Summary

PRINT = 3,1
        VALUE =“Customer Number”

PRINT = .,20
        VALUE = REQUEST (1,6)        !Print customer number from response message

PRINT = .+1,1
        VALUE =“Customer Name”

PRINT = .,20
        VALUE =REQUEST (.,30)        !Print customer name from response message

PRINT = .+2,1
        VALUE =“Address”

PRINT = .,20
        VALUE =REQUEST (.,30)        !Print first address line from response message

PRINT = .+1,20
        VALUE =REQUEST (.,30)        !Second address line

PRINT = .+1,20
        VALUE =REQUEST (.,30)        !Third address line

PRINT = .+1,1
        VALUE =‘ “ZIP” Code’

PRINT = .,45
        VALUE =REQUEST (.,5)         !Print Zip Code from response message

PRINT = .+1,1
        VALUE =“Telephone”

PRINT = .,20
        VALUE =“(”

PRINT = .,.
        VALUE =REQUEST (.,3)         !Print telephone area code
PRINT = .,.
        VALUE =“)”

PRINT = .,.
        VALUE =REQUEST (.,3)            !Print telephone exchange number

PRINT = .,.
        VALUE =“–”

PRINT = .,.
        VALUE =REQUEST (.,4)            !Print telephone extension number

PRINT = .+2,1
        VALUE =“Company Contact:”

PRINT = .,20
        VALUE =REQUEST (.,20)           !Print attention line name

PRINT = .+2,1
        VALUE =“Credit Limit ($)”

PRINT = .,20
        VALUE =REQUEST (.,12)           !Print credit limit amount

PRINT = .+3,1
        VALUE =“******************** Statement of Account - As of ”,
                DATE,“ ********************”
```

Note the use of the “dot” construct in the following set of statements. Rather than forcing you to specify a row of headings followed by a row of data items, ATL allows you to define a heading, drop down to define the associated data field, and to use the “.-” notation to return and code the header information for the next column of information.

```
PRINT = .+2,5                           !Skip to summary heading line
        VALUE =“Current Balance”

PRINT = .+1,7                           !Information on this line
        VALUE =REQUEST (.,12)           !Print current balance amount

PRINT = .-1,25                          !Return for next column header
        VALUE =“Purchases to Date”

PRINT = .+1,27
        VALUE =REQUEST (.,12)           !Print purchase amount to date

PRINT = .-1,47                          !Return for next column header
        VALUE =“Next Order No.”
```

```
PRINT = .+1,52
        VALUE =REQUEST (.,4)          !Print order sequence number

PRINT = .-1,63                        !Return for next column header
        VALUE ="Next Payment No."

PRINT = .+1,69
        VALUE =REQUEST (.,4)          !Print payment sequence number

END
```

Figure 3-12 shows an example of the hard-copy Report form that is printed when a Customer Record is supplied to the form as the requesting message.



```
                 Customer Master File Subsystem - Account Summary

Customer Number     001005
Customer Name       C G Kuchasian Stamp and Coin

Address             Dept SV6
                    572 Bush Road
                    Lincoln, VA
'ZIP' Code                                  22078
Telephone           (804) 581-2478

Company Contact:    C G Kuchasian

Credit Limit ($)        500.00


******************** Statement of Account - As of 18-APR-78 ********************

     Current Balance       Purchases to Date      Next Order No.  Next Payment No.
                 .00                     .00                0002              0001
```

Figure 3-12 LA180 listing of Report Form

3.3 CODING THE TRANSACTION SELECTION FORM

The Transaction Selection form is a specific type of ATL form that enables a terminal user to select a defined transaction, and initiate it from an application terminal.

Most commonly, a Transaction Selection Form consists of a list of defined transactions in the form of a Menu. One menu item (in this case, a transaction name) can be selected by the user using the SELECT key. When a form has been specified as a Transaction Selection Form, the user must select exactly one menu item. If more or less than one item is selected, the bell sounds, and a VT62 error message appears on the terminal error line.

When the user presses ENTER after selecting a MENU item, the transaction named by that selection is invoked by the transaction processor. The first form defined for that transaction is then displayed and the user begins the data entry operation.

Optionally, you may specify that the terminal user must type the desired transaction name into an INPUT field.

To illustrate the manner in which a Transaction Selection Form is coded, assume that you wish to code such a form for a set of four transactions:

1. ADDCUS – Adding a Customer Record
2. CHCUS – Changing a Customer Record
3. DELCUS – Deleting a Customer Record
4. DPYCUS – Displaying a Customer Record.

The form envisioned by the application designer is shown in Figure 3-13.

Since you can specify MENU Fields only in the Display area of a Form, a Transaction Selection Form using the MENU parameter is generally defined with the entire screen, except the terminal error line, as a Display area.

3.3.1 Coding the FORM Statement

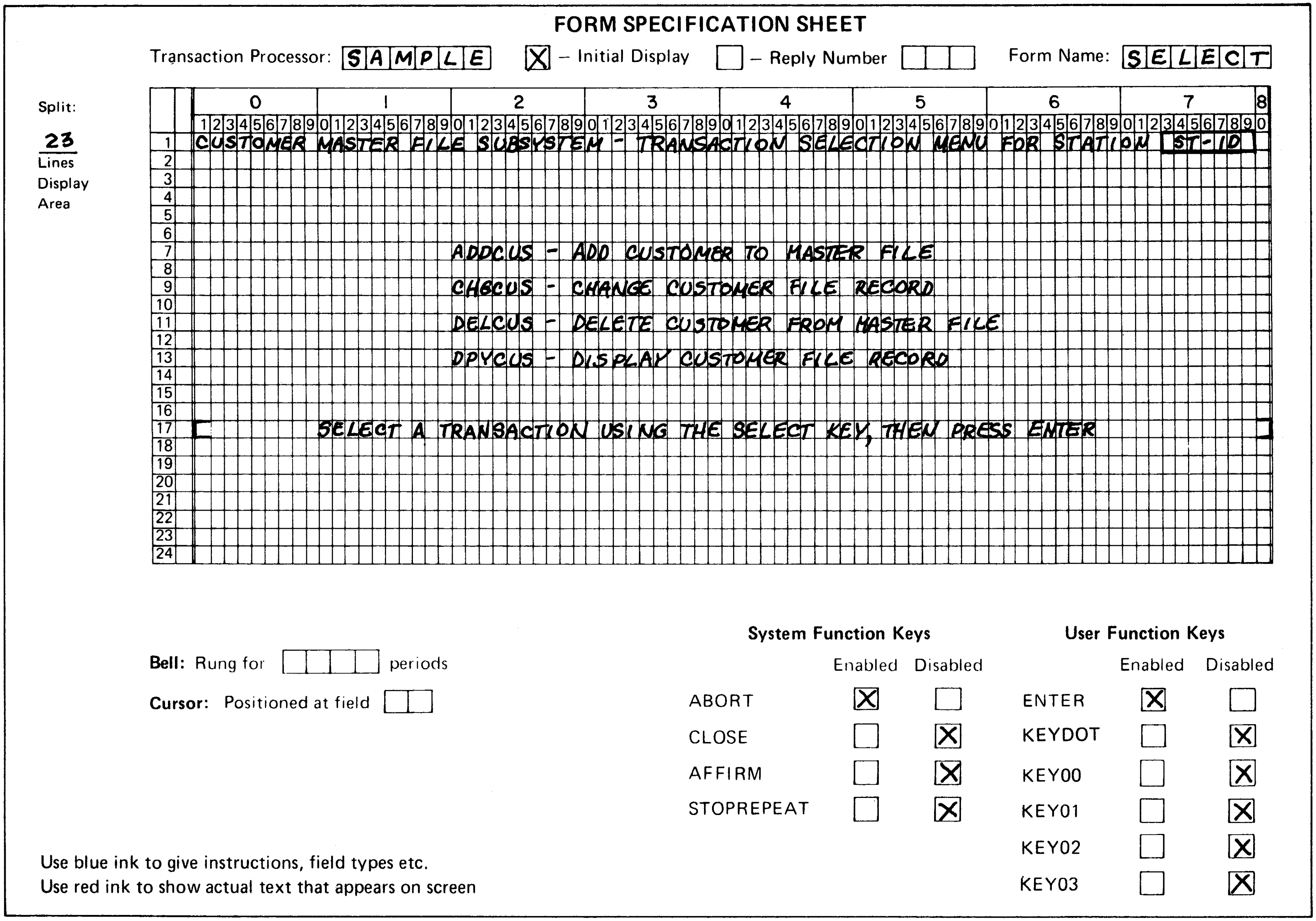
To specify a form with only display area, use the FORM statement:

```
FORM
          SPLIT = 23                     !All Display Area
          SELECT = MENU,1,2              !Selection is made from a Menu.
```

When you specify the SELECT clause in the FORM statement, you are instructing the ATL utility to compile a form definition that allows the terminal user to specify the transaction to be initiated at the application terminal. Two methods are available: MENU selection or typing the transaction name into a labeled INPUT field.

In the SELECT clause shown above, MENU selection has been chosen. The number 1 specifies that Reply #1 is displayed when the user selects a non-existent transaction. The number 2 in the SELECT clause parameter list means that Reply #2 of this form is displayed when a terminal user selects a transaction not included in the work class for that terminal.

If the keyword NOAUTHORIZE is specified in the last parameter position of the SELECT clause, no user authorization checking is performed. (See the *TRAX Application Programmers Guide* AA-D329A-TC for an explanation of work class assignments.)



FORM SPECIFICATION SHEET

Transcription Processor: SAMPLE ☒ – Initial Display □ – Reply Number □□□ Form Name: SELECT

Split: 23 Lines Display Area

```
             0         1         2         3         4         5         6         7         8
    1234567890123456789012345678901234567890123456789012345678901234567890123456789 0
 1  CUSTOMER MASTER FILE SUBSYSTEM - TRANSACTION SELECTION MENU FOR STATION [ST-ID]
 2
 3
 4
 5
 6
 7                      ADDCUS - ADD CUSTOMER TO MASTER FILE
 8
 9                      CHGCUS - CHANGE CUSTOMER FILE RECORD
10
11                      DELCUS - DELETE CUSTOMER FROM MASTER FILE
12
13                      DPYCUS - DISPLAY CUSTOMER FILE RECORD
14
15
16
17  [         SELECT A TRANSACTION USING THE SELECT KEY, THEN PRESS ENTER                ]
18
19
20
21
22
23
24
```

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 3-13 Menu-type Transaction Selection Form

3.3.2 Coding the Display Area Using Menu Fields

Since you specified the transaction Selection Form with only Display area, you may use two types of statements to define the screen layout. You use DISPLAY statements to code the explanatory text and user instructions, and you use MENU statements to define the contents of the Menu data items.

```
DISPLAY = 1,1                          !Give terminal ID and title line
            VALUE =“Customer Master File Subsystem – ”,
                      “Transaction Selection Menu”,
                      “ for station ”,STATION
            ATTRIBUTE = NOBLANK,REVERSE

MENU    = 7,20                         !Put Add Customer in menu list
            VALUE =“ADDCUS”

DISPLAY = .,.
            VALUE =“ – Add Customer to Master File”
            ATTRIBUTE = NOBLANK

MENU    = .+2,20                       !Put Change Customer in menu list
            VALUE =“CHGCUS”

DISPLAY = .,.
            VALUE =“ – Change Customer File Record”
            ATTRIBUTE = NOBLANK

MENU    = .+2,20                       !Put Delete Customer in menu list
            VALUE =“DELCUS”

DISPLAY = .,.
            VALUE =“ – Delete Customer from Master File”
            ATTRIBUTE = NOBLANK

MENU    = .+2,20                       !Put Display Customer in menu list
            VALUE = “DPYCUS”

DISPLAY = .,.
            VALUE =“ – Display Customer File Record”
            ATTRIBUTE = NOBLANK

DISPLAY = .+4,10                       !For use in explaining error replies
            LABEL = REPLY.TEXT
            LENGTH = 71

DISPLAY = .+2,10                       !User instructions
            VALUE =“Select a transaction using the SELECT key, then press ENTER”
            ATTRIBUTE = NOBLANK
```

3.3.3 Coding Replies

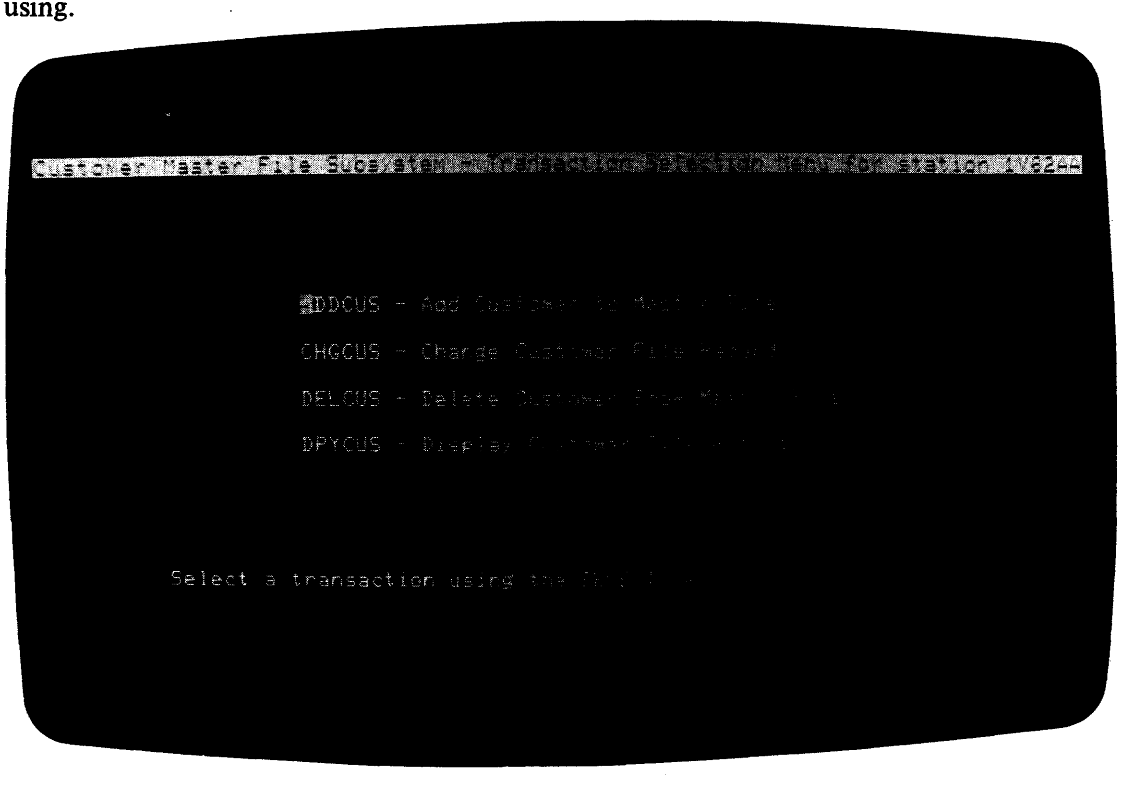
The two REPLY statements correspond to the second and third parameters in the SELECT Clause of the FORM statement. Reply 1 informs the user that the transaction selected is not known to the transaction processor:

```
REPLY = 1                                   !Error Reply 1
          WRITE = REPLY.TEXT,"You have selected an unknown transaction. ",
                    "Please choose again."
```

Reply 2 allows you to inform the terminal that the transaction selected does not have the same work class as the terminal station.

```
REPLY = 2                                   !Error Reply 2
          WRITE = REPLY.TEXT,"You have selected a transaction you are ",
                    "not authorized to use."
END                                         !End of Form: SELECT
```

Figure 3-14 is a photograph of the VT62 screen display that appears on an application terminal after you code the Transaction Selection Form definition shown in this section. You use the ATL utility (see Chapter 4) to compile it and place it into a transaction processor forms definition file. Then you use the STADEF utility (see the *TRAX Application Programmers Guide*) to assign that form name as the initial form for the terminal station associated with the application terminal you are using.



```
Customer Master File Subsystem - Transaction Selection Menu for Station [illegible]

            ADDCUS - Add Customer to Master File

            CHGCUS - Change Customer File Record

            DELCUS - Delete Customer From Master File

            DPYCUS - Display Customer [illegible]



      Select a transaction using the [illegible]
```

Figure 3-14 Transaction Selection Form

3.3.4 Coding a Selection Form for User Input

You can also use an Input field to specify the transaction to be selected. In this case, you must specify a SPLIT clause as part of the FORM statement, define an Input field using an INPUT statement, and supply the label of that field in the SELECT clause of the form statement. The following statements describe the FORM and INPUT statements and clauses required for this method.

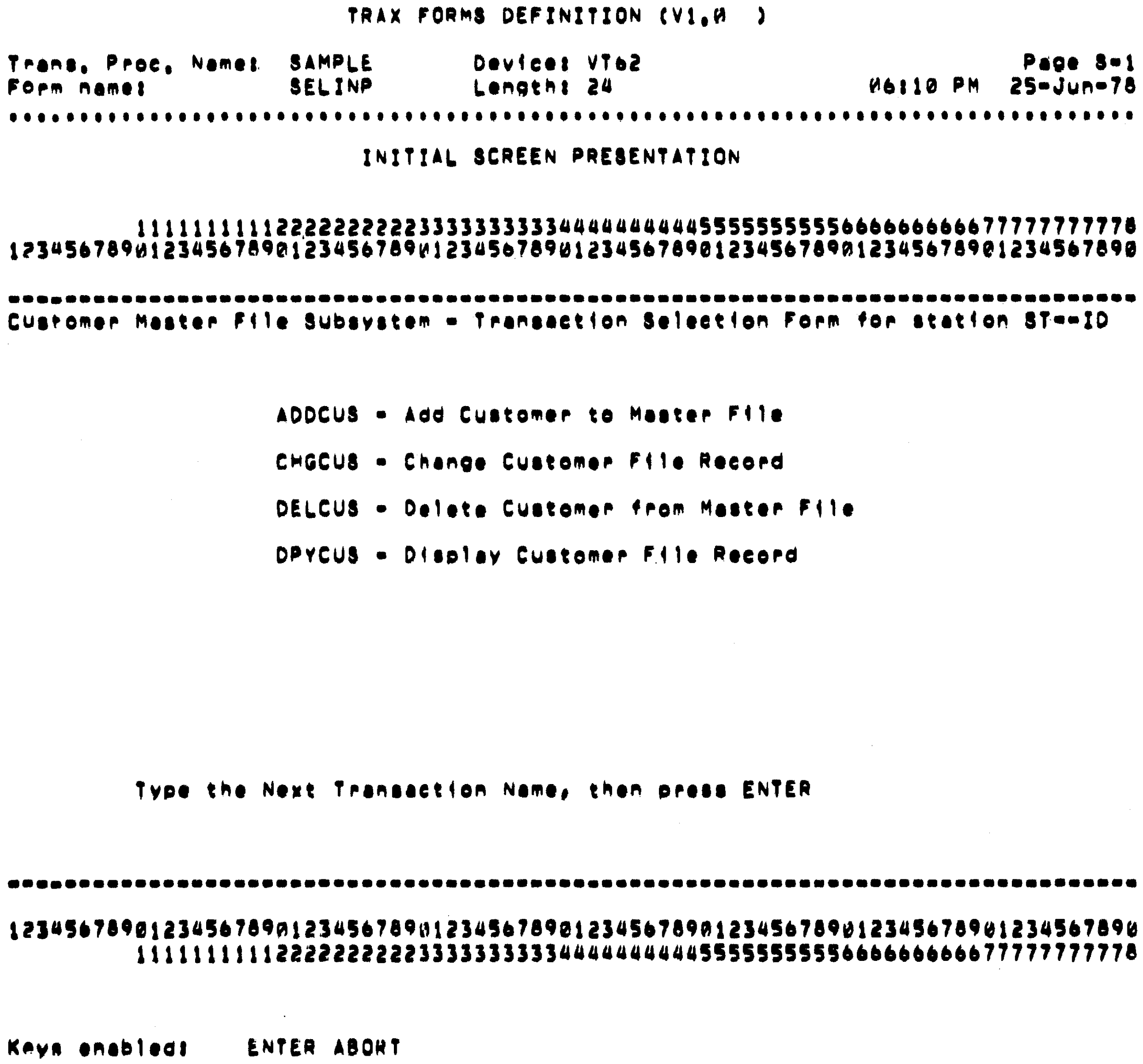
```
FORM
            SPLIT = 20                       !Top 20 lines are Display area
            SELECT = TRANS-NAME,1,2          !Specify field label and reply number
```

The Display area is coded as before, except the menu fields are coded as Display fields.

3.3.4.1 Coding the Prompt and Input Fields – The form area coding requires the following two statements.

```
PROMPT =    1,1                              !First position in Form area
            VALUE = "Enter the Transaction Name you wish to perform:"

INPUT =     .,.+3                            !Start Input field
            LABEL - TRANS-NAME               !Identify for use by SELECT clause
            LENGTH = 6                       !Length of transaction name
            ATTRIBUTES = REQUIRED,           !Require input
                         FULL,               !Field must be full
                         REVERSE             !Highlight with reverse video
```

The rest of the form remains the same as before, except that the Menu fields have been merged with the existing Display fields. Figure 3-15 shows the screen display that appears when this form is used instead of the menu-type Transaction Selection Form coded earlier in this chapter.



```
                        TRAX FORMS DEFINITION (V1.0  )

Trans. Proc. Name:  SAMPLE       Device: VT62                           Page S-1
Form name:          SELINP       Length: 24                 06:10 PM  25-Jun-78
................................................................................
                         INITIAL SCREEN PRESENTATION

         11111111112222222222333333333344444444445555555555666666666677777777778
12345678901234567890123456789012345678901234567890123456789012345678901234567890

--------------------------------------------------------------------------------
Customer Master File Subsystem - Transaction Selection Form for station ST--ID


                   ADDCUS - Add Customer to Master File

                   CHGCUS - Change Customer File Record

                   DELCUS - Delete Customer from Master File

                   DPYCUS - Display Customer File Record





         Type the Next Transaction Name, then press ENTER


--------------------------------------------------------------------------------

12345678901234567890123456789012345678901234567890123456789012345678901234567890
         11111111112222222222333333333344444444445555555555666666666677777777778


Keys enabled:    ENTER ABORT
```

Figure 3-15 Initial Screen Display for Input-type Transaction Selection Form

CHAPTER 4
USING THE ATL UTILITY

4.1 THE FORMS DEFINITION FILE

Each transaction processor has associated with it a forms definition file. This file is created by the TRAX definition utility TPDEF at the time a transaction processor is defined. A transaction processor's forms definition file resides in UFD [1,300], and takes its filename from the name of the transaction processor. The filetype for a forms definition file is .FDF. For example, if you name a transaction processor SAMPLE, its forms definition file will have the following file specification:

```
SY:[1,300]SAMPLE,FDF
```

The forms definition file is the library for all compiled form definitions used by that transaction processor. Each form definition has its own record in the forms definition file. Additions, deletions, and updates to the forms definition file are performed by the ATL utility program, which is described in the remaining sections of this chapter.

4.2 TRAX UTILITY DIALOG CONVENTIONS

The ATL Utility adheres to the following TRAX utility dialog conventions:

1. Help Text – If the question does not give enough information to enable you to answer, you can access help text by typing a question mark followed by the RETURN (RET) key. For example:

   ```
   List <ALL>? ?(RET)
   ```

   The utility responds with an explanation and repeats the question. For example:

   ```
   The allowed options are:  ALL, STATEMENTS, FORMS, NONE

   List <ALL>?
   ```

2. The RETURN Key – Pressing the RETURN key (RET) terminates your response. If you have specified valid input, the utility proceeds to the next question.

   ```
   Command <COMPILE>? (RET)
   Form name?  SELECT (RET)
   ATL source file <SELECT.ATL>?
   ```

3. Responses – The content of each question indicates the type of response expected by the utility. The utility checks the input as you enter it. If the input is incorrect in any way, the utility returns an error message and repeats the question.

4. Defaults – If the utility assumes a default value as a response to a question, that value is shown in brackets following the question.

   ```
   Command <COMPILE>?
   ```

   You accept the default by pressing the RETURN key. Values other than the default are specified as a normal response to the question. For example, to accept the default, simply press RETURN:

   ```
   Command <COMPILE>? RET
   ```

   To specify a different value, type the response followed by a carriage return.

   ```
   COMMAND <COMPILE>? ADD RET
   ```

5. YES/NO Answers – If a question requires a YES or NO answer, you can respond by typing Y, YE, YES, N, or NO as a response to the question.

   ```
   Really DELETE forms definition record "ADCUS1"? YES RET
   ```

6. Abbreviating Responses – If a question asks you to select an item from a known set (utility commands, for example), you need to enter only the number of characters required to uniquely identify the selected item within the set.

   ```
   Command? I RET
   ```

   I stands for INDEX, one command from a set that includes ADD, COMPILE, DELETE, INDEX, EXIT, PURGE, RENAME, REPLACE, and SHOW.

7. The ESC Key – All utilities ask questions in a specific order. To reverse the order, that is, to return to the last question asked, press ESC.

   ```
   Device Type <VT62>? RET

   LIST <ALL>? ESC

   Device Type <VT62>?
   ```

8. CTRL Z to Exit – You can make an orderly exit from a TRAX utility after a question by typing CTRL/Z in response to the question.

   ```
   Command <COMPILE> CTRL/Z

   >
   ```

4.3 INVOKING THE ATL UTILITY

You invoke the ATL utility from a TRAX support environment terminal by typing the command line:

```
RUN $ATL
```

The ATL utility program responds with identifying text:

```
ATL      V1.0
TRAX Forms Definition File Utility
```

The utility then issues the question line:

```
Transaction processor name <" ">?
```

If you are simply compiling a form, you can respond with a carriage return. If you are performing any other operation, you must supply the name of the previously defined transaction processor whose forms definition file is accessed by the ATL utility during this terminl session.

If you were using the TRAX Sample Application, you would type the transaction processor name SAMPLE:

```
Transaction processor name <" ">? SAMPLE RET
```

If the ATL utility is able to find an existing forms definition file under [1,300] SAMPLE.FDF, the utility then issues the question line:

```
Command <COMPILE>?
```

You respond by typing one of the following command keywords:

```
ADD
COMPILE
DELETE
EXIST
INDEX
PURGE
RENAME
REPLACE
SHOW
```

The ATL utility processes each of these commands in a different fashion. To simplify the explanation of how each command works, the interaction between you and the utility is explained for each of the commands in a separate subsection. These command descriptions are in alphabetical order beginning with the ADD command.

4.4 ADDING FORMS TO A FORM DEFINITION FILE

The ADD Command
Before you can use a form definition as part of a transaction processor, you must first add it to the transaction processor's forms definition file. To do this, you invoke the ATL utility and specify the ADD command. The ATL utility compiles the form definition source file and flags fatal and warning errors. If no errors are encountered, the form definition is added to the forms definition file. If warning errors are encountered, you are asked to decide if the form definition should be added. If fatal errors are encountered, the forms definition file remains unchanged.

Every time you use the ADD command, the ATL utility creates a new version of the forms definition file. To remove unwanted earlier versions of forms definition files, use the ATL PURGE command.

The dialog for the ADD command is illustrated in the following example. In this case, the form SELECT, coded in Chapter 3, is being added to the forms definition file SAMPLE (specified in the first section of this discussion).

First, you specify ADD in response to the command question:

```
Command <COMPILE>? ADD RET
```

The ATL utility issues the question for the form name, and you specify SELECT in response.

```
Form name?  SELECT RET
```

The ATL utility then asks you for the name of the form definition source file. If your file has the same filename as the form name you just specified, and a filetype of .ATL. you simply respond with a carriage return. If you answer this question by specifying a file, ATL compiles the source statements in that file. If you do not specify a filetype, the ATL utility assumes a default filetype of .ATL.

```
ATL source file <SELECT.ATL>?  RET
```

Since the default value was correct, the carriage return response was sufficient.

The ATL utility then asks you for the device on which the form is to be displayed. The default device assumption is the VT62 CRT terminal. The only other legal device you can specify is the LA180 output-only device. The legal response keywords are VT62 and OUTPUT ONLY.

```
Device type <VT62>?  RET
```

The ATL utility then asks you for the types of output listings that you want it to generate. Typing a question mark in response to any question causes help text to be displayed and the question reissued. For example:

```
List <ALL>?  ? RET

The allowed options are:  ALL, STATEMENTS, FORMS, NONE.
```

The option keywords cause the following listings to be generated:

1. ALL – You want the ATL utility to create a complete set of listings for this form definition. This is the default.
2. STATEMENTS – You want only the compiled statements to be listed.
3. FORMS – You want the listings of the form field and message layout tables as well as the screen display listings.
4. NONE – You don't want any listings to be produced. This option increases ATL compiler speed, but provides no means for tracking down errors that occur.

Once you specify the types of listings to be generated, the ATL utility asks you for the device or file where the output listings are written. The default is LP:. If you want to save form definition listings in a file, you must specify a filename where the information is to be stored. If you specify a filename, and omit the filetype, the ATL utility assumes .LST as the filetype for the listing file. In the following example, you are requesting the ATL utility to store the generated listings of the form definition SELECT on your system device with the file specification SELECT.LST:

```
Listing device or file <LP:>?  SELECT.LST RET
```

After you respond to this question, the ATL utility then processes the source file. When processing is completed, the utility issues the question:

```
Command <COMPILE>? EXIT RET
```

You can specify another command, accept the default command, COMPILE, or exit from the ATL utility. Specifying a command name causes the utility to enter that command. Responding to the Command question by typing EXIT causes the ATL utility to terminate normally.

Figure 4-1 is a complete terminal listing showing the sequence of questions and responses described in this section.

4.5 COMPILING FORMS FOR DEBUGGING PURPOSES

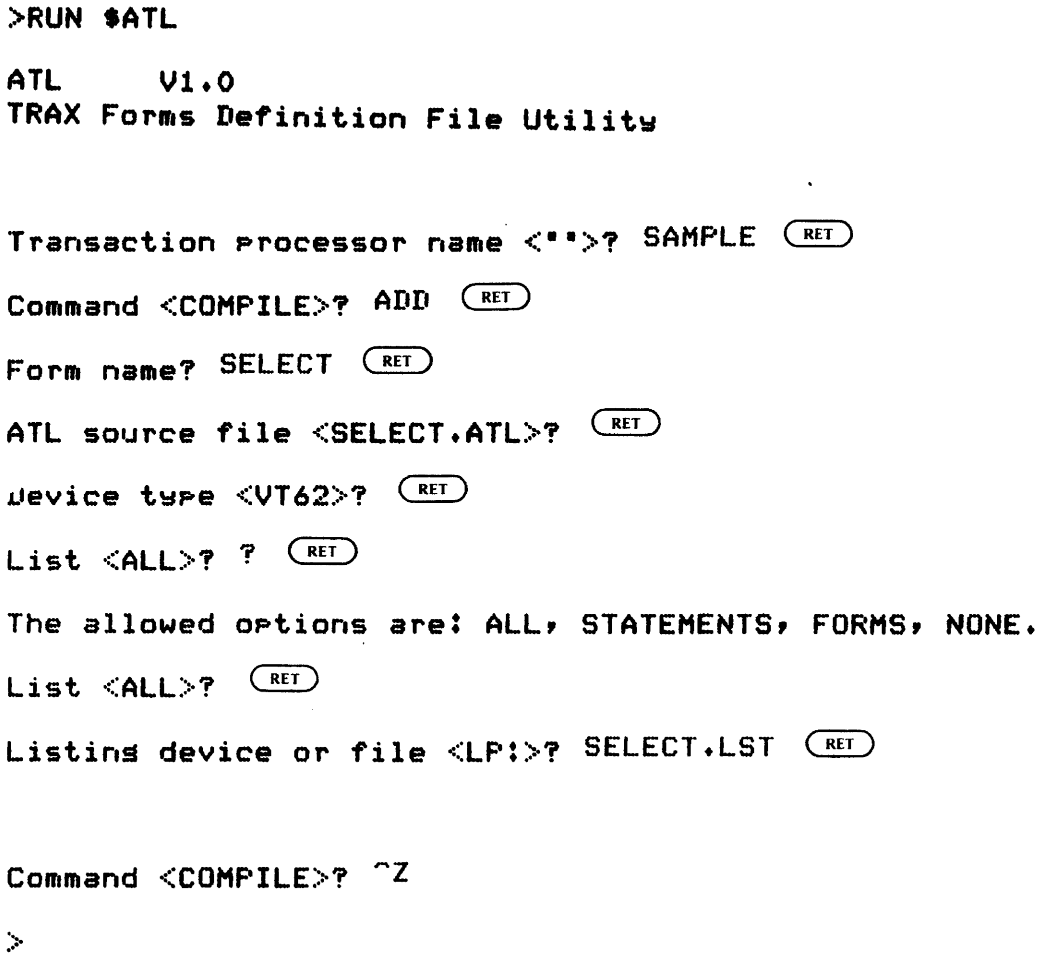
The COMPILE Command

The COMPILE command allows you to process your ATL source statements without altering a transaction processor's forms definition file. A recommended practice is to run the initial form definition source statements through the ATL utility in the COMPILE command, then use the listings generated by the COMPILE pass to debug and streamline your form definition. Once you have compiled a form without errors, you then use the ADD or REPLACE commands to insert the new form definition in the form definition file.

If you specify the COMPILE command, the ATL utility discards the transaction processor name that you previously specified. If you were to REPLACE a form, COMPILE a second form, and finally REPLACE the second form, you must reenter the transaction processor name before specifying the REPLACE command for the second form. You do this by pressing the ESCAPE ESC key when the "Command?" question appears. Pressing ESC returns you to the previous question, where you can enter the transaction processor name. If you have neglected to reenter the transaction processor name, when you specify the REPLACE command, the ATL utility issues the error message:

% Selected command requires TP name – restarting dialog.

followed by the transaction processor name question.



```
>RUN $ATL

ATL      V1.0
TRAX Forms Definition File Utility


Transaction processor name <"">? SAMPLE  RET

Command <COMPILE>? ADD  RET

Form name? SELECT  RET

ATL source file <SELECT.ATL>?  RET

Device type <VT62>?  RET

List <ALL>? ?  RET

The allowed options are: ALL, STATEMENTS, FORMS, NONE.

List <ALL>?  RET

Listing device or file <LP:>? SELECT.LST  RET


Command <COMPILE>? ^Z

>
```

Figure 4-1 ADD Command Dialog

The COMPILE command dialog is similar to that of the ADD command. However, you are not required to specify a transaction processor name in COMPILE. You may simply type a carriage return in response to the question:

Transaction Processor Name <" "> RET

The ATL utility then asks you to select a command. Since COMPILE is the default command assumed by the ATL utility, you can simply type a carriage return:

Command <COMPILE>? RET

The next question asks you to specify the name of the ATL source file to be compiled. In response to this question, you specify the source file ADCUST.ATL.

```
ATL source file? ADCUST.ATL RET
```

If you specify only the filename, the ATL utility assumes that the filetype is .ATL.

Next, you are asked to supply the type of device on which the form is intended to be displayed. This information is required by the ATL compiler to generate the screen layouts and field descriptions.

```
Device type <VT62>?
```

Typing a question mark in response to this question causes the utility to display help text for that question, and then reissue the question. For example:

```
Device type <VT62>? ? RET

Currently allowed forms devices are: VT62, OUTPUT ONLY.

Device type <VT62>? RET
```

After you specify the device type, the ATL utility then asks you to specify the types of listings that you want the utility to generate as it processes the source statements.

In the case of initial debugging runs, it is very useful to ask for all listings, statements, form field descriptions, and screen display listings. These listings are shown and described at the end of this chapter.

```
List <ALL>? RET
```

You may direct the ATL utility to produce output listings that are stored in a file, or direct these listings to a physical device. If you accept the ATL default, the listings produced by the ATL utility are directed to the line printer LP:. To produce the listings on your terminal, you must specify TI:. If you want to save the compile listings on a file, respond with a file specification. The filename is required. If no filetype is specified, the ATL utility assumes .LST as a default. In this example, where a form is being compiled for debugging purposes, the logical place for the output would be the line printer, since you need to examine the compiler output to debug the form. If you do not specify a device or file, but simply type a carriage return, the ATL utility sends the specified listings directly to the line printer.

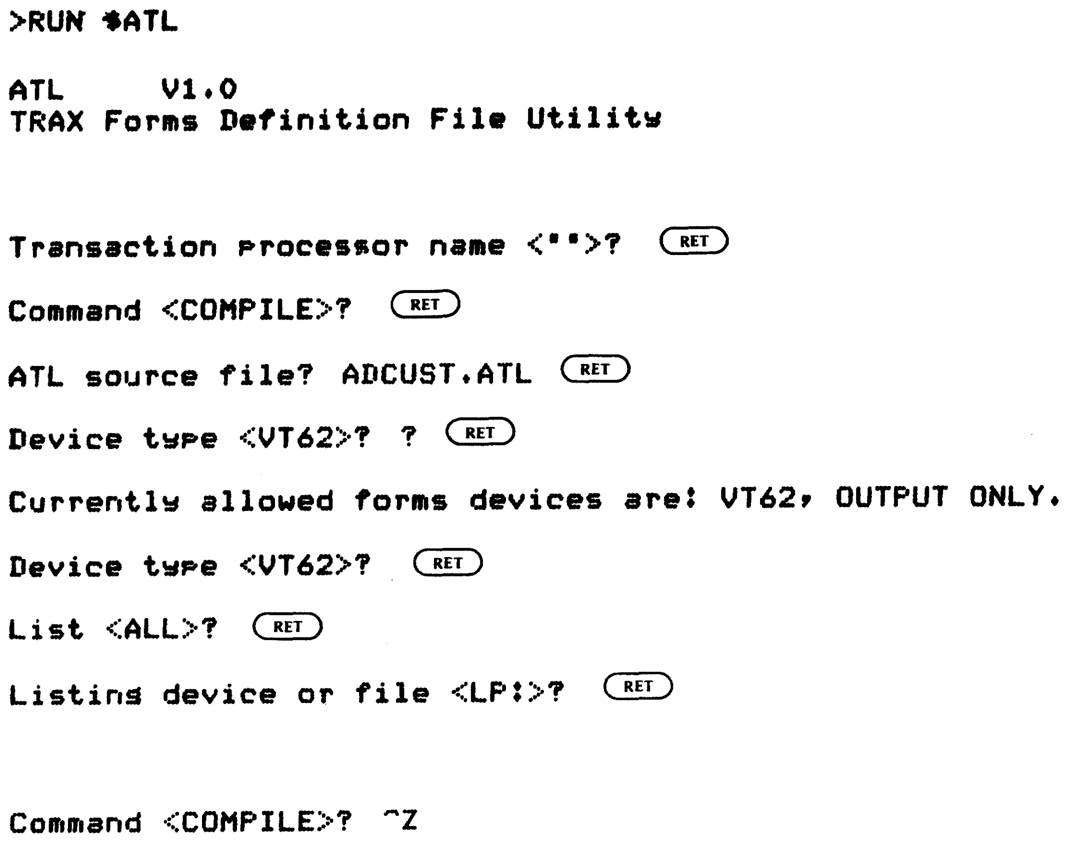
```
Listing device or file <LP:>? RET
```

At this point, the ATL utility processes the source file and creates the listing that you specified. When processing is completed, the ATL utility issues the question:

```
Command <COMPILE>?
```

Specifying EXIT causes the ATL utility to terminate in a normal manner. You can terminate the ATL utility at any time by typing CTRL/Z in response to any ATL utility question.

Figure 4-2 is an actual hard-copy terminal listing of the dialog described in this section.



```
>RUN $ATL

ATL      V1.0
TRAX Forms Definition File Utility


Transaction processor name <"">?  RET

Command <COMPILE>?  RET

ATL source file? ADCUST.ATL  RET

Device type <VT62>?  ?  RET

Currently allowed forms devices are: VT62, OUTPUT ONLY.

Device type <VT62>?  RET

List <ALL>?  RET

Listing device or file <LP:>?  RET


Command <COMPILE>? ^Z
```

Figure 4-2 COMPILE Command Dialog

4.6 DELETING FORMS FROM A FORMS DEFINITION FILE

The DELETE Command

As your transactions change, you may find it necessary to delete form definitions from a transaction processor. You can delete form definitions by invoking the DELETE command. To do this, you must specify the DELETE keyword in response to the ATL utility's command question.

After you type the keyword DELETE, the utility asks you to specify the name of the form to be deleted. The ATL utility creates a new version of the forms definition file. To remove unwanted earlier versions of forms definition files, use the ATL PURGE command.

```
Command <COMPILE>? DELETE RET
```

The utility responds by asking you to specify the name of the form you wish to delete.

```
Form name? ADCUS1 RET
```

The utility then asks you if you really want to delete the form you specified. This allows you to avoid erroneous deletions due to misspelling the form name.

```
Really DELETE forms definition record "ADCUS1"?
```

Typing a question mark in response to this question causes the utility to display help text and then reissue the question. For example:

```
Really DELETE forms definition record "ADCUS1"? ? RET

Respond with Y, YE or YES to permit DELETE to occur. Respond with N or NO to skip
DELETE operation. An explicit response MUST be given.

Really DELETE forms definition record "ADCUS1"? YES RET
```

Typing YES causes the utility to delete the specified form definition record. The ATL utility deletes the record by creating a new version of the forms definition file that omits the deleted record. Once the new forms definition file is created, the utility again issues the Command question, allowing you to go on to other functions or to terminate the session.

```
Command <COMPILE>? CTRL/Z
```

Typing CTRL/Z in response to any question brings the ATL utility to a normal conclusion.

Figure 4-3 is an actual terminal listing of the DELETE command dialog described in this section.

4.7 DISPLAYING THE FORM DEFINITION FILE INDEX

The INDEX Command
During the development of a transaction processor you may require an index of all form definitions contained in the transaction processor's forms definition file. The INDEX command allows you to obtain information regarding the state of the .FDF file.

After you invoke the utility and specify the appropriate transaction processor name, you invoke the INDEX command by typing INDEX in response to the Command question:

```
Command <COMPILE>? INDEX RET
```

The ATL utility then asks you to supply the filename where the ATL utility can write the index, or the device on which you want to display the index. If you specify a file name, the index will be written to your account on the system device with the default filetype .LST. The assumed default is the line printer, but the following example specifies that the index is to be displayed at your support environment terminal.

```
Listing device or file <LP:>? TI: RET
```

```
>RUN $ATL

ATL     V1.0
TRAX Forms Definition File Utility


Transaction processor name <"">? SAMPLE (RET)

Command <COMPILE>? DELETE (RET)

Form name? ADCUS1 (RET)

Really DELETE forms definition record "ADCUS1"? ? (RET)

Respond with Y, YE or YES to permit DELETE to occur. Respond
with N or NO to skip DELETE operation. An explicit
response MUST be given.

Really DELETE forms definition record "ADCUS1"? YES (RET)
```

Figure 4-3 DELETE Command Dialog

The utility then prints an index listing on your terminal in the form shown in the example listing in Figure 4-4. When the utility completes printing the index listing, it reissues the command question, allowing you to terminate the session or enter another command.

```
Command <COMPILE>? EXIT (RET)
```

Typing EXIT causes the ATL utility to terminate in an orderly fashion.

Figure 4-4 is an actual terminal listing of the INDEX command dialog described in this section.

4.8 PURGING VERSIONS OF THE FORM DEFINITION FILE

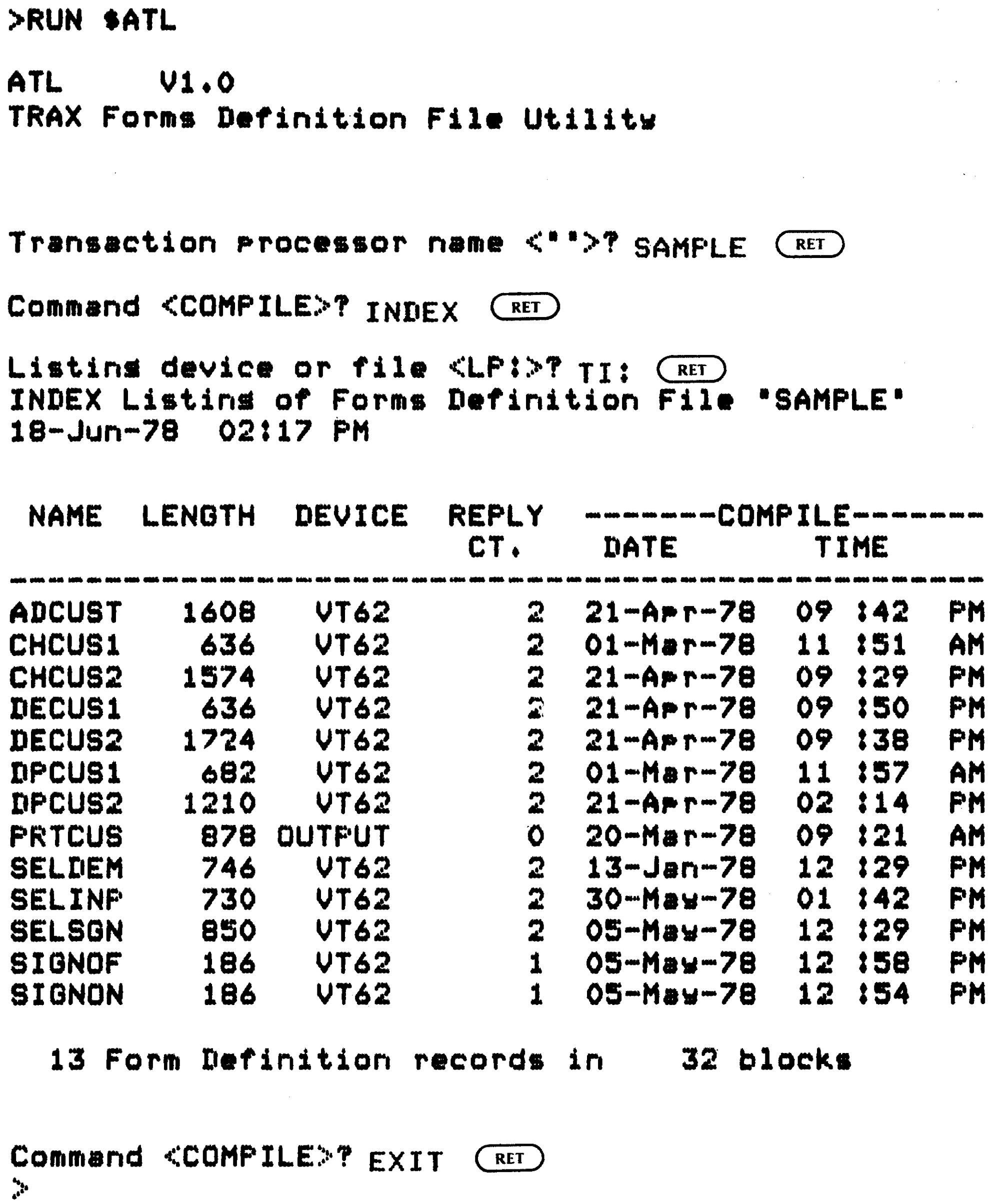
The PURGE COMMAND

Any time that you add, replace, rename, or delete a form definition record from a forms definition file, a new version of that file is created by the ATL utility. Since large applications can create very large forms definition files, you may want to purge some of the outdated versions. This procedure can be done by invoking the ATL utility, specifying the transaction processor name whose forms definition files you wish to purge, and responding to the Command question with the keyword PURGE.

```
Command <COMPILE>? PURGE (RET)
```

The ATL utility then finds the earliest (lowest numbered) version of the file and issues the question:

```
Earliest version <42>?
```



```
>RUN $ATL

ATL     V1.0
TRAX Forms Definition File Utility


Transaction processor name <"">? SAMPLE  RET

Command <COMPILE>? INDEX  RET

Listing device or file <LP:>? TI:  RET
INDEX Listing of Forms Definition File "SAMPLE"
18-Jun-78  02:17 PM

 NAME  LENGTH  DEVICE  REPLY  -------COMPILE-------
                        CT.    DATE        TIME
---------------------------------------------------
ADCUST   1608    VT62      2  21-Apr-78  09 :42  PM
CHCUS1    636    VT62      2  01-Mar-78  11 :51  AM
CHCUS2   1574    VT62      2  21-Apr-78  09 :29  PM
DECUS1    636    VT62      2  21-Apr-78  09 :50  PM
DECUS2   1724    VT62      2  21-Apr-78  09 :38  PM
DPCUS1    682    VT62      2  01-Mar-78  11 :57  AM
DPCUS2   1210    VT62      2  21-Apr-78  02 :14  PM
PRTCUS    878  OUTPUT      0  20-Mar-78  09 :21  AM
SELDEM    746    VT62      2  13-Jan-78  12 :29  PM
SELINP    730    VT62      2  30-May-78  01 :42  PM
SELSGN    850    VT62      2  05-May-78  12 :29  PM
SIGNOF    186    VT62      1  05-May-78  12 :58  PM
SIGNON    186    VT62      1  05-May-78  12 :54  PM

  13 Form Definition records in    32 blocks


Command <COMPILE>? EXIT  RET
>
```

Figure 4-4 INDEX Command Dialog

Typing a question mark in response to a question causes the ATL utility to display the help text for that question and then reissue the question. For example:

Earliest version <42>? ? RET

Respond with starting version # of range to be PURGEd.

Earliest version <42>? 43 RET

The value in brackets (<42> in this example) represents the lowest version number of the forms definition file currently found on the directory for UFD [1,300].

If you type a carriage return, the utility assumes that you agree that the proposed version number is the earliest version you wish to purge. In this example, version 42 was saved. The lowest version number purged was 43. Any version number you type must be an octal value.

The next question asks for the latest version to the purged. The default value is one less than the highest existing version number. In this example:

```
Latest version <44>?? RET
```

Displaying the help text tells you to:

```
Respond with ending version # of range to be PURGEd.

Latest version <44>?
```

As before, typing a carriage return means that you accept the proposed version as the highest version number to be purged. If you type a lower version number in response to this question, any versions with numbers higher than the value you specify are retained. Any version number you type must be an octal value.

The utility then asks you:

```
Really PURGE version 43 through version 44?YES RET
```

Typing YES causes the ATL utility to purge the files specified in the question. Note that the PURGE removes versions 43 through 44 inclusive. The utility responds with a brief message, and then reissues the Command question, allowing you to specify some other command, or make an orderly exit from the utility.

```
Version(s) 43 thru 44 PURGEd.
```

If, for any reason, the ATL utility cannot purge a version, it will issue the warning message:

```
Unable to PURGE version nn.

Command <COMPILE>?EXIT RET
```

Figure 4-5 is an actual terminal listing of the PURGE command dialog described in this section.

4.9 RENAMING FORMS IN A FORMS DEFINITION FILE

The RENAME Command
At times, you may wish to rename existing form definitions.

Every time you use the RENAME command, the ATL utility creates a new version of the forms definition file. To remove unwanted earlier versions of forms definition files, use the ATL PURGE command.

```
Command <COMPILE>? PURGE

Earliest version <42>? ?  RET

Respond with starting version # of range to be PURGEd.

Earliest version <42>? 43  RET

Latest version <44>? ?  RET

Respond with ending version # of range to be PURGEd.

Latest version <44>?  RET

Really PURGE version 43 through version 44? Y  RET
Version(s) 43 thru 44 PURGEd.


Command <COMPILE>? EXIT  RET
>
```

Figure 4-5 PURGE Command Dialog

In the example shown here, the form ADCUST is renamed to be ADCUS1 for purposes of testing it in the transaction processor SAMPLE. After invoking the utility and specifying the transaction processor in the normal manner, you must type RENAME in response to the Command question.

Command <COMPILE>?RENAME RET

The ATL utility then asks you to supply the existing form name to which you respond ADCUST.

Form name?ADCUST RET

You are then asked to supply the new name for that form.

New form name?? RET

Typing a question mark causes help text to be displayed. The form name help text gives the following information:

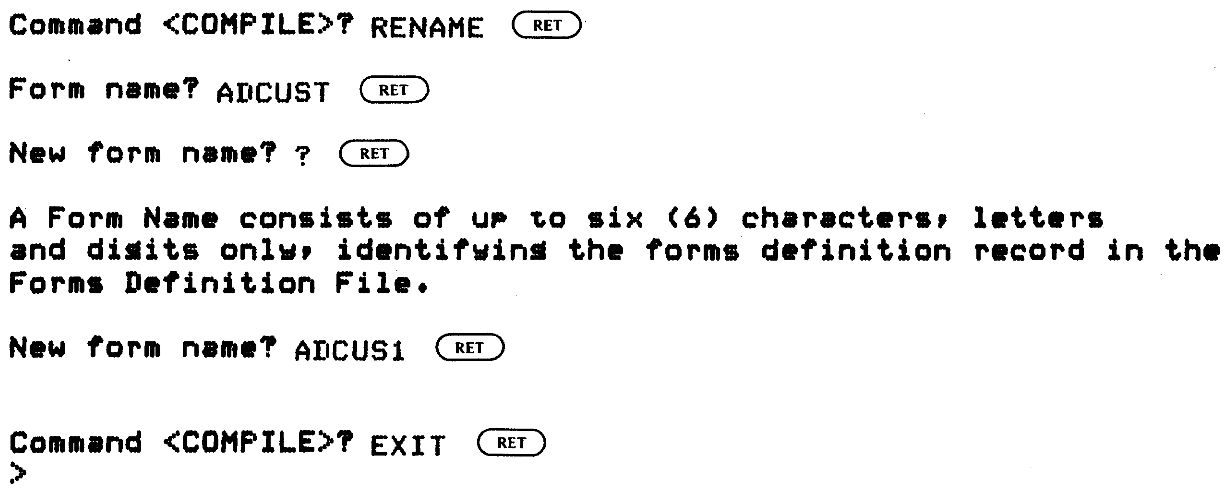
A Form Name consists of up to six (6) characters, letters, and digits only, identifying the forms definition record in the Forms Definition File.

New form name?ADCUS1 RET

After you specify the new form name, the ATL utility updates the form definition record name and reissues the Command question.

Command <COMPILE>?EXIT RET

Figure 4-6 is an actual terminal listing showing RENAME command dialog described in this section.



```
Command <COMPILE>? RENAME RET

Form name? ADCUST RET

New form name? ? RET

A Form Name consists of up to six (6) characters, letters
and digits only, identifying the forms definition record in the
Forms Definition File.

New form name? ADCUS1 RET


Command <COMPILE>? EXIT RET
>
```

Figure 4-6 RENAME Command Dialog

4.10 REPLACING FORMS IN A FORM DEFINITION FILE

The REPLACE Command
To modify an existing form definition, first you must edit the source statement file using the TRAX Editor, then you may invoke the ATL utility, specify the transaction processor name, and type the keyword REPLACE to enter the REPLACE command. In the REPLACE command, the ATL utility compiles the specified source statement file, creates a new form definition record, and then creates a new version of the form definition file, replacing the existing form definition with the one just compiled.

The dialog for the REPLACE command is illustrated in the following example. In this case, the form ADCUST, coded in Chapter 3, is being modified and replaced in the forms definition file SAMPLE that was specified in the first section of this discussion.

Every time you use the REPLACE command, the ATL utility creates a new version of the forms definition file. To remove unwanted earlier versions of forms definition files, use the ATL PURGE command.

To replace an existing form definition, you first must specify REPLACE in response to the Command question:

```
Command <COMPILE>? REPLACE RET
```

The ATL utility then responds with the question for the form name; you specify ADCUST in response.

```
Form name? ADCUST RET
```

The ATL utility then asks you for the name of your form definition source file. If your file has the same filename as the form name you just specified, and a filetype of .ATL, you can enter a carriage return and accept the default value. If you answer this question with a full file specification, ATL compiles the source statements in that file. If you omit a filetype, the ATL utility assumes a default of .ATL. In the following example, the name suggested by the utility was correct, but the source file was stored in UFD [350,227]. For this reason, the UFD was specified along with the filename and filetype.

```
ATL source file <ADCUST.ATL>? [350,227] ADCUST.LST (RET)
```

The ATL utility then asks you for the device on which the form is to be displayed. The default device assumption is the VT62 video terminal. The only other legal device that may be specified is the LA180 output-only device. The legal responses to this question are VT62 and OUTPUT ONLY.

```
Device type <VT62>? (RET)
```

The ATL utility then asks you for the types of output listings that you desire. Typing a question mark in response to a question causes help text to be displayed, and the question reissued, for example:

```
List <ALL>? ? (RET)

The allowed options are: ALL, STATEMENTS, FORMS, NONE.
```

The option keywords cause the following listings to be generated.

1. ALL – You want the ATL utility to create a complete set of listings for this form definition.
2. STATEMENTS – You want only the compiled statements to be listed.
3. FORMS – You want only the listings of the form field and message layout tables as well as the screen display listings.
4. NONE – You don't want any listings to be produced. This option increases ATL compiler speed, but provides no means for tracking down errors that occur.

```
List <ALL>?
```

Finally, the ATL utility asks you for the device or file where the output listings are written. The default is LP:. Should you wish to save Form Definition Listings, you must specify a file name where the information may be stored. If you specify only a filename, .LST is the assumed filetype. In the following example, ADCUST.LST is a disk file containing the listings from the Form Definition ADCUST.

```
Listing device or file <LP:>? ? (RET)
```

Typing a question mark in response to a question results in the help text being displayed on the terminal.

Supply the file specification for the listing device and/or file name where the listing output is to be written. The default device and file is the line printer.

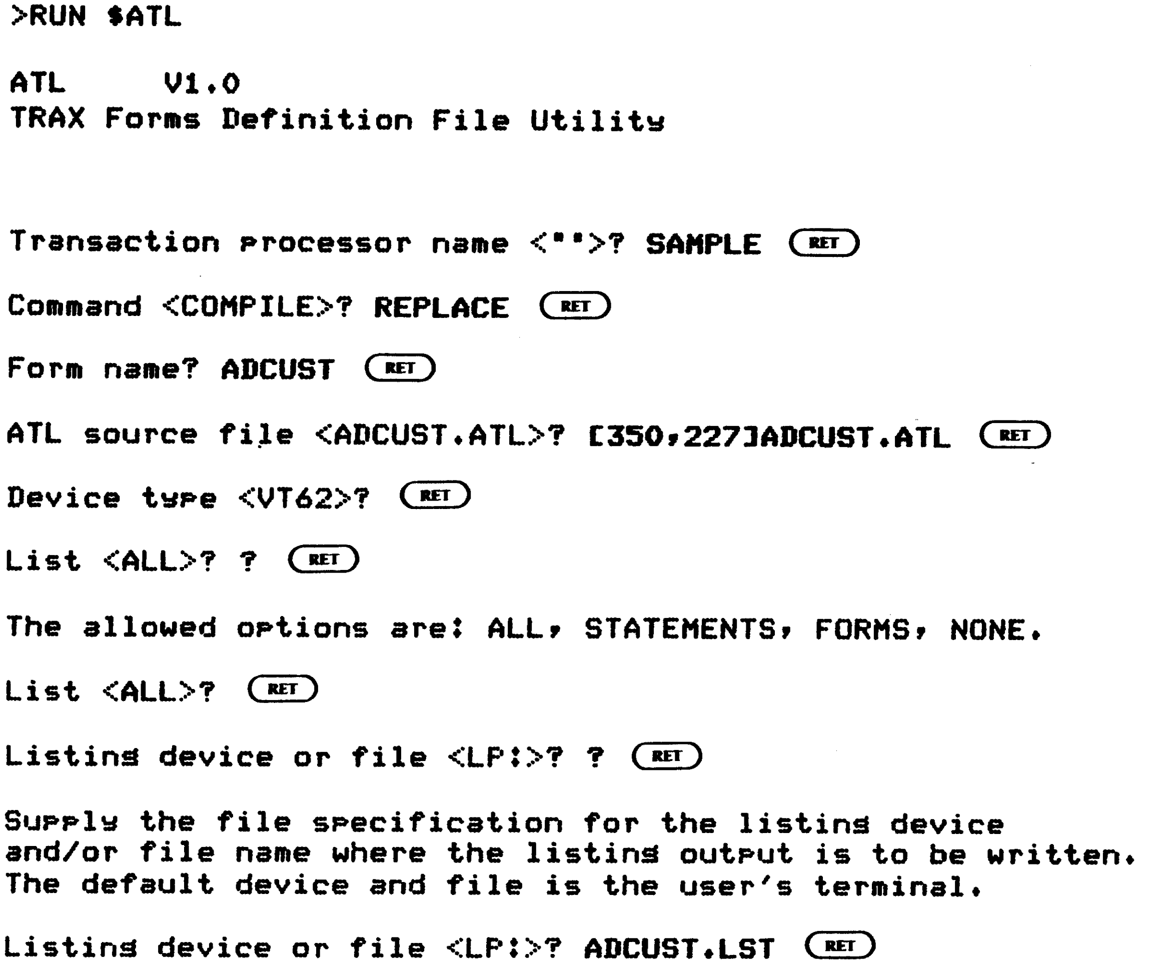
> Listing device or file <LP:>? **ADCUST.LST** (RET)

This is the last question in the dialog. The ATL utility then processes the source file. If no errors are detected, the new form definition replaces the old one in a new version of the forms definition file. If warning errors are detected by the ATL utility, you are asked to decide if the erroneous version is to be posted to the forms definition file. Fatal errors preclude any update of the forms definition file. When processing is completed, the utility issues the question:

> Command <COMPILE>? **EXIT** (RET)

Responding with EXIT causes the ATL utility to terminate normally. The utility may also be terminated by typing a (CTRL/Z) character in response to any question.

Figure 4-7 is an actual terminal listing showing the REPLACE command dialog described in this section.



```
>RUN $ATL

ATL      V1.0
TRAX Forms Definition File Utility


Transaction processor name <"">? SAMPLE (RET)

Command <COMPILE>? REPLACE (RET)

Form name? ADCUST (RET)

ATL source file <ADCUST.ATL>? [350,227]ADCUST.ATL (RET)

Device type <VT62>? (RET)

List <ALL>? ? (RET)

The allowed options are: ALL, STATEMENTS, FORMS, NONE.

List <ALL>? (RET)

Listing device or file <LP:>? ? (RET)

Supply the file specification for the listing device
and/or file name where the listing output is to be written.
The default device and file is the user's terminal.

Listing device or file <LP:>? ADCUST.LST (RET)
```

Figure 4-7 REPLACE Command Dialog

4.11 DISPLAYING A FORM DEFINITION RECORD

The SHOW Command
You can display the screen formats of a form definition from the forms definition file, by invoking the ATL utility, supplying the desired transaction processor name, and then specifying the SHOW keyword to cause the utility to invoke the SHOW command. For example:

```
Command <COMPILE>? SHOW RET
```

The ATL utility then asks you which form you want to display:

```
Form name?
```

Typing a question mark in response to a question causes help text for that question to be displayed on your terminal. For example:

```
Form name? ? RET

A Form Name consists of up to six (6) characters, letters, and digits only, identifying the
forms definition record in the Forms Definition File.

Form name? ADCUST RET
```

In this example, the form ADCUST was requested. You may also respond with the keyword ALL, causing all forms in that forms definition file to be shown. The ATL utility then asks you to supplythe filename where the listings should be stored or the device where the display screen listings could be directed. The line printer is assumed as the default device for this command. If you specify a filename, the default filetype .LST is assumed.

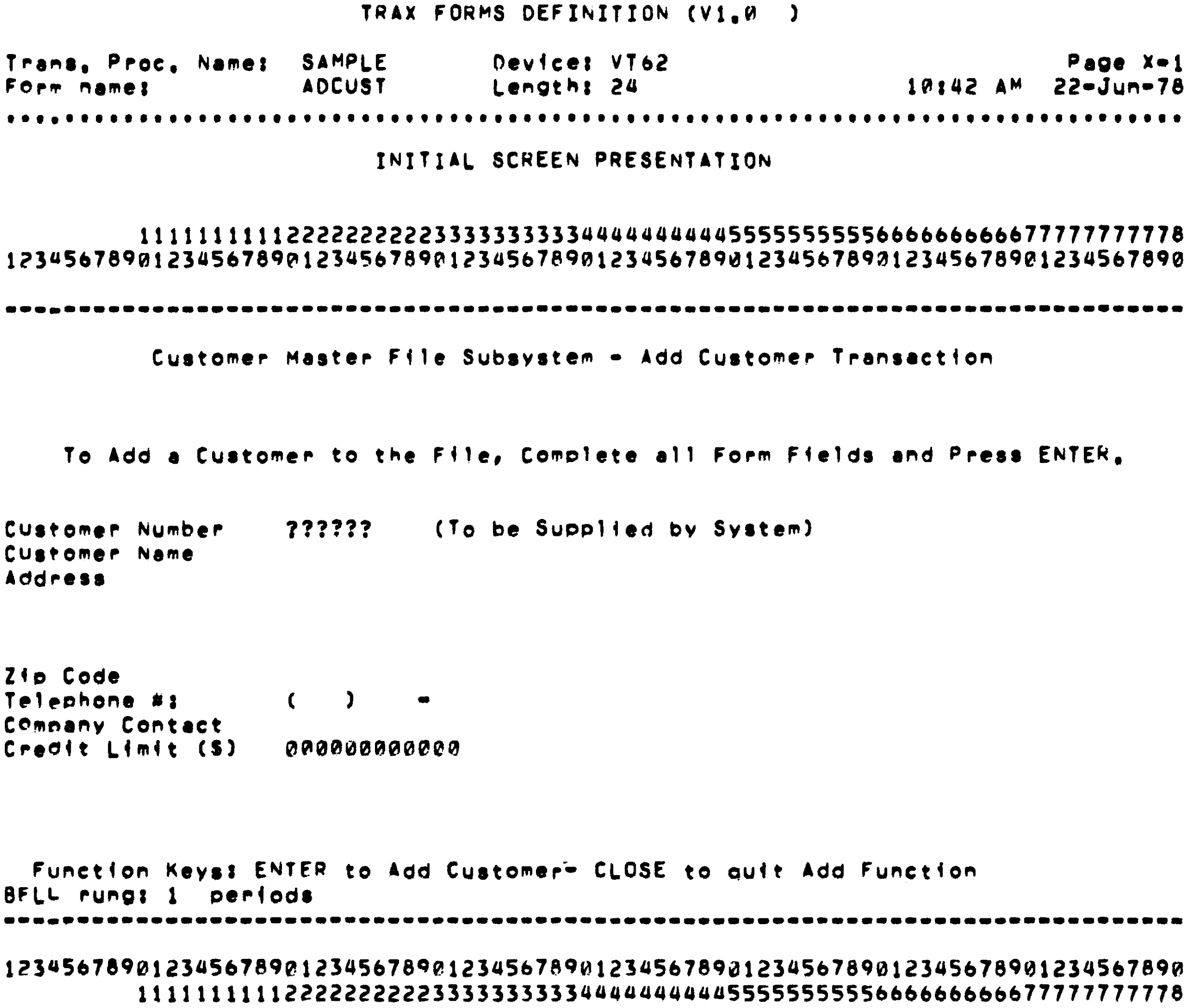
```
Listing device or file <LP:>? RET

Command <COMPILE>? EXIT RET
```

Figure 4-8 is an actual terminal listing of the SHOW command dialog described in this section.

Figures 4-9a, 4-9b, and 4-9c are the listings of screen display formats produced by the ATL utility in the SHOW command. Figure 4-9a is the initial screen display, Figure 4-9b shows the screen after REPLY 1, and Figure 4-9c shows the screen display after REPLY 2.

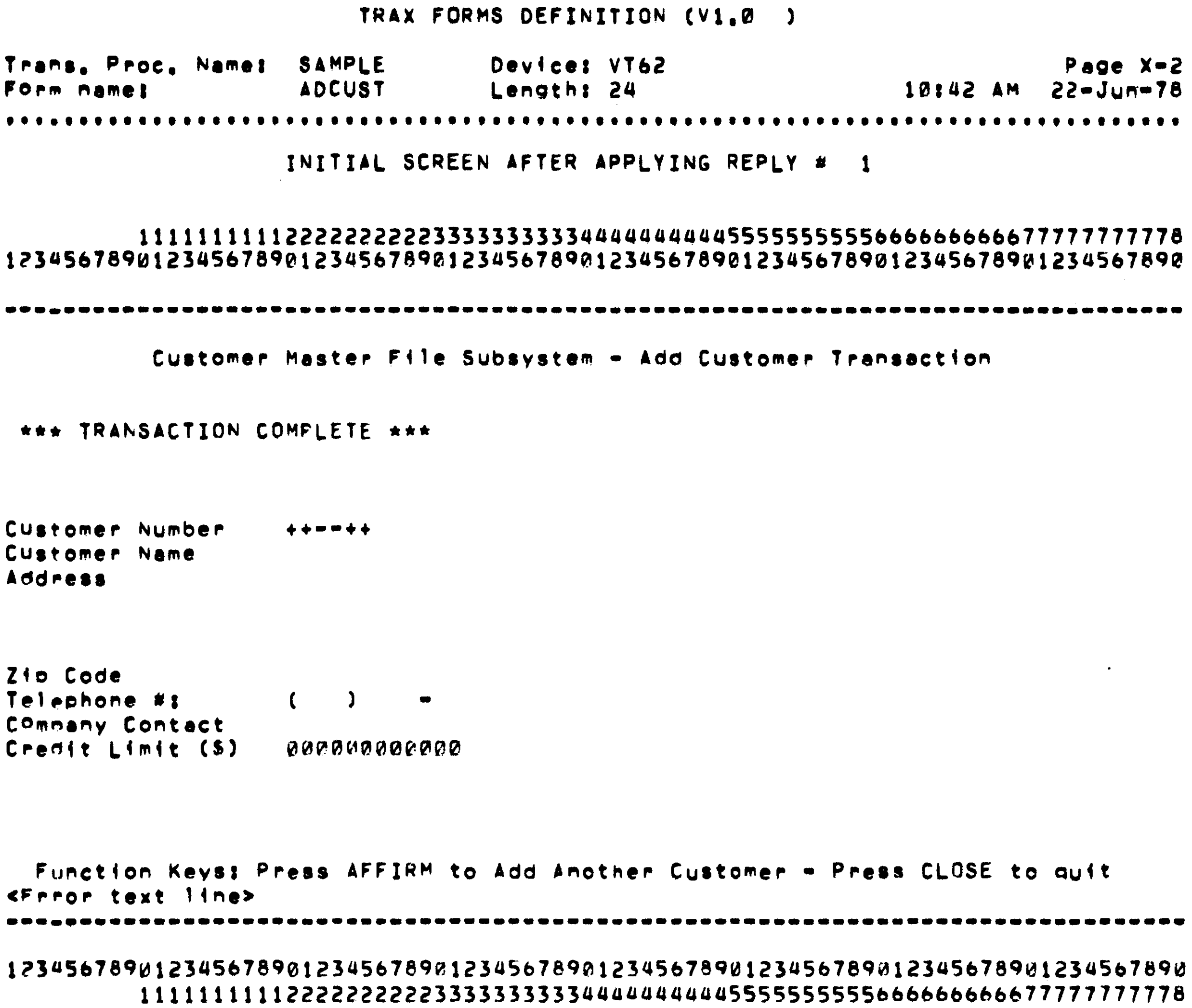
```
Command <COMPILE>? SHOW (RET)

Form name? ? (RET)

A Form Name consists of up to six (6) characters, letters
and digits only, identifying the forms definition record in the
Forms Definition File.

Form name? ADCUST (RET)

Listing device or file <LP:>? (RET)


Command <COMPILE>? ^Z
```

Figure 4-8 SHOW Command Dialog



```
                        TRAX FORMS DEFINITION (V1.0  )

Trans, Proc, Name:  SAMPLE       Device: VT62                           Page X=1
Form name:          ADCUST       Length: 24                 10:42 AM  22-Jun-78
................................................................................
                         INITIAL SCREEN PRESENTATION


         11111111112222222222333333333344444444445555555555666666666677777777778
12345678901234567890123456789012345678901234567890123456789012345678901234567890

--------------------------------------------------------------------------------

          Customer Master File Subsystem - Add Customer Transaction


    To Add a Customer to the File, Complete all Form Fields and Press ENTER,


Customer Number    ??????    (To be Supplied by System)
Customer Name
Address



Zip Code
Telephone #:       (   )    -
Company Contact
Credit Limit ($)   000000000000



  Function Keys: ENTER to Add Customer- CLOSE to quit Add Function
BFLL rung: 1  periods
--------------------------------------------------------------------------------

12345678901234567890123456789012345678901234567890123456789012345678901234567890
         11111111112222222222333333333344444444445555555555666666666677777777778
```

Figure 4-9a Initial Screen Display Listing



```
                        TRAX FORMS DEFINITION (V1.0  )

Trans. Proc. Name:  SAMPLE       Device: VT62                           Page X-2
Form name:          ADCUST       Length: 24                   10:42 AM  22-Jun-78
................................................................................
                   INITIAL SCREEN AFTER APPLYING REPLY #  1


          1111111111222222222233333333334444444444555555555566666666667777777777 8
12345678901234567890123456789012345678901234567890123456789012345678901234567890

--------------------------------------------------------------------------------

          Customer Master File Subsystem - Add Customer Transaction


 *** TRANSACTION COMPLETE ***



Customer Number    ++--++
Customer Name
Address



Zip Code
Telephone #:       (   )    -
Company Contact
Credit Limit ($)   000000000000




  Function Keys: Press AFFIRM to Add Another Customer - Press CLOSE to quit
<Error text line>
--------------------------------------------------------------------------------

12345678901234567890123456789012345678901234567890123456789012345678901234567890
          1111111111222222222233333333334444444444555555555566666666667777777777 8
```

Figure 4-9b Reply 1 Screen Display Listing

4.12 ANNOTATED ATL UTILITY OUTPUT LISTINGS

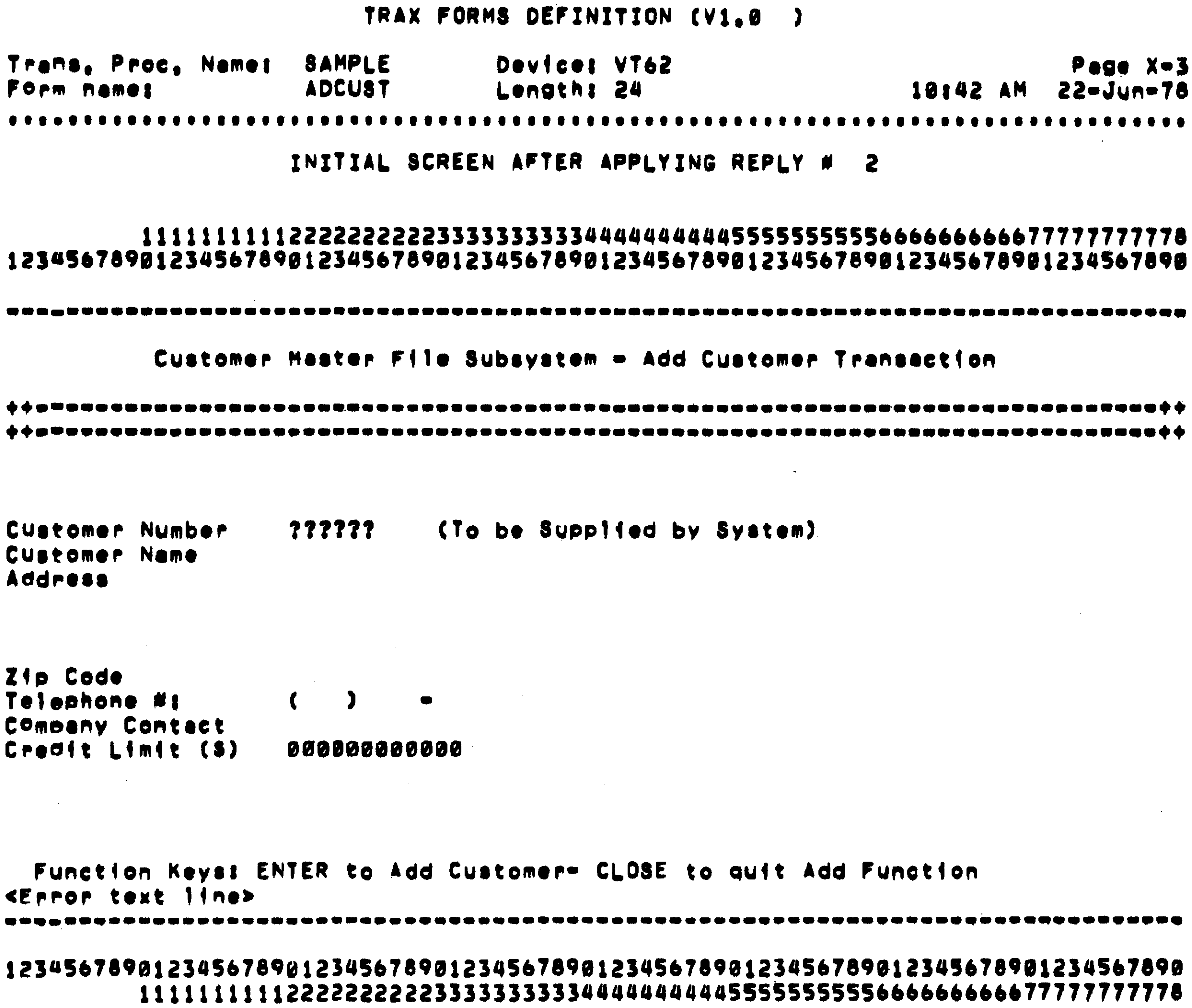
This section contains the output listings produced by the ATL utility when you specify the ADD, COMPILE, or REPLACE commands.

The example used is the data entry and display form, ADCUST, that was coded in Chapter 3 and processed through the ATL utility in this chapter.

4.12 1 Forms Definition Statements – Pages L-1 to L-5

The ATL utility checks your ATL source statements for errors, and then places them into a listing format. The listing has a header for every page which gives:

1. the transaction processor name.
2. the form name.
3. the device type on which the form will run.
4. the number of lines on the form.



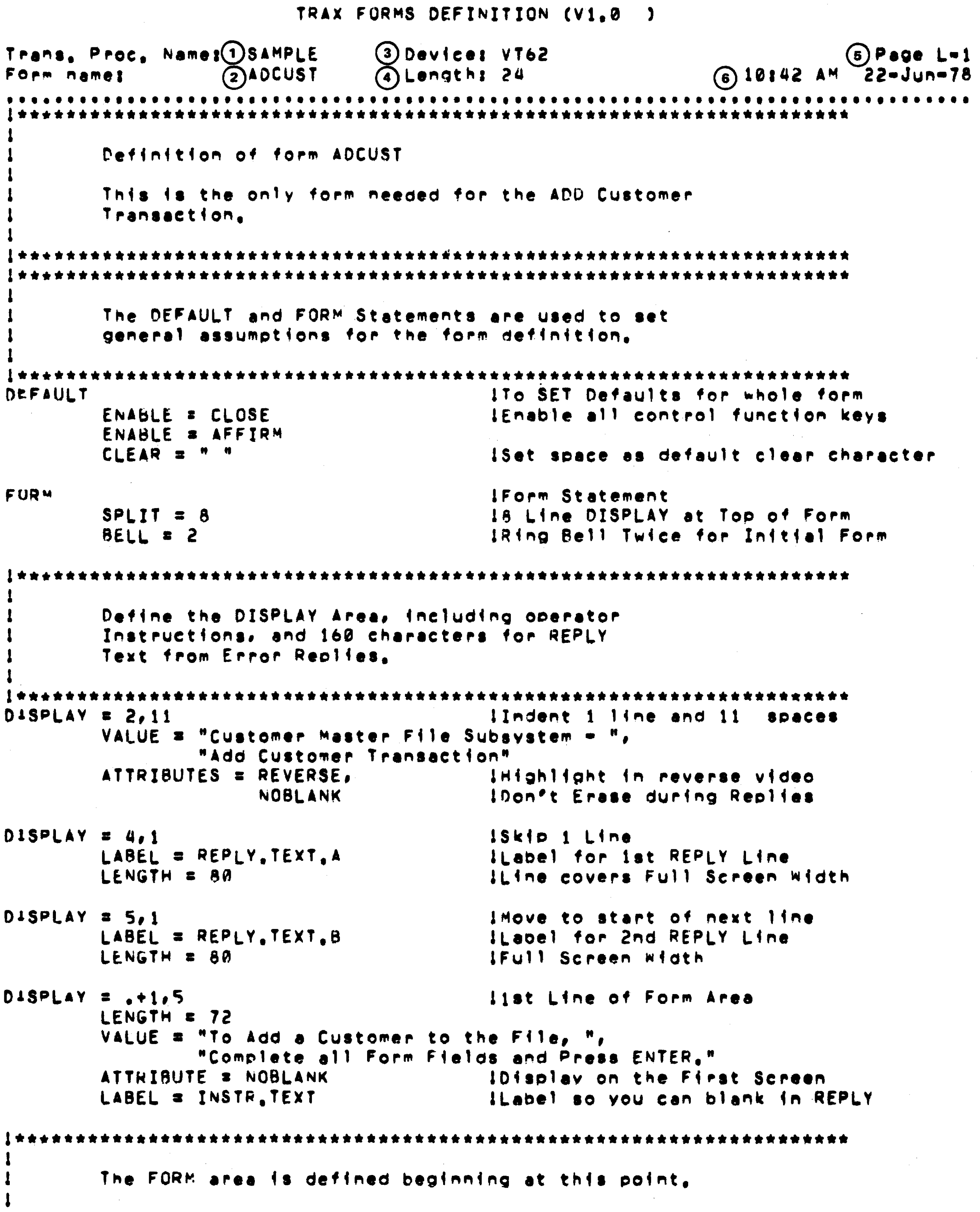
```
                        TRAX FORMS DEFINITION (V1.0  )

Trans. Proc. Name:  SAMPLE       Device: VT62                           Page X-3
Form name:          ADCUST       Length: 24                   10:42 AM  22-Jun-78
................................................................................
                   INITIAL SCREEN AFTER APPLYING REPLY #  2

         11111111112222222222333333333344444444445555555555666666666677777777778
12345678901234567890123456789012345678901234567890123456789012345678901234567890

--------------------------------------------------------------------------------
         Customer Master File Subsystem - Add Customer Transaction

++----------------------------------------------------------------------------++
++----------------------------------------------------------------------------++


Customer Number    ??????  (To be Supplied by System)
Customer Name
Address


Zip Code
Telephone #:       (   )    -
Company Contact
Credit Limit ($)   000000000000



  Function Keys: ENTER to Add Customer- CLOSE to quit Add Function
<Error text line>
--------------------------------------------------------------------------------

12345678901234567890123456789012345678901234567890123456789012345678901234567890
         11111111112222222222333333333344444444445555555555666666666677777777778
```

Figure 4-9c Reply 2 Screen Display Listing

5. the page number. F stands for a FORM listing, L stands for a source statement listing, and S stands for a sample screen display.
6. the date and time that the form definition was compiled by the ATL utility.

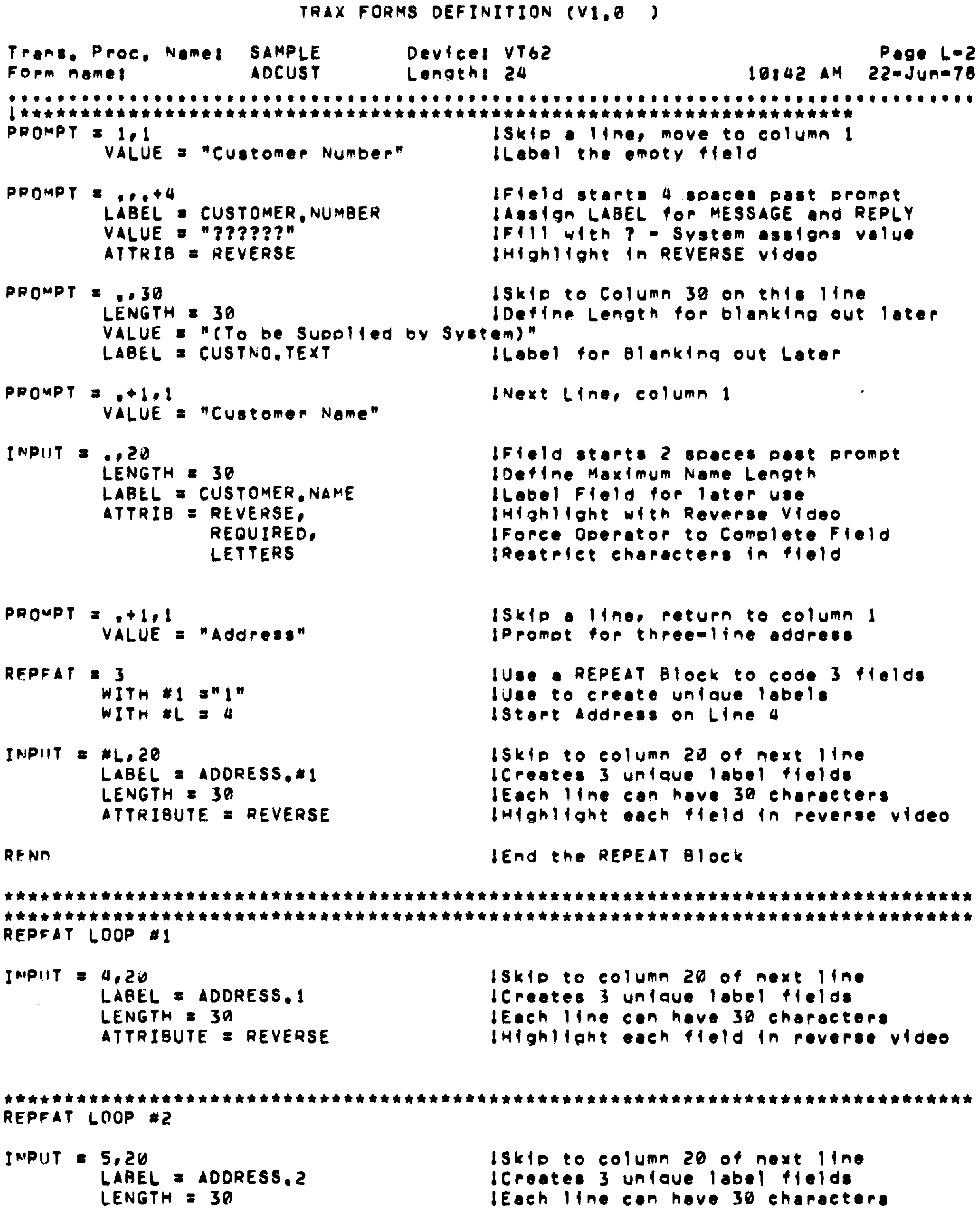
If a syntax error is detected, it appears following the incorrect statement. Sometimes syntax errors are not detected until the utility begins processing the following ATL statement. Thus some ATL statements that have a syntax error message are in themselves correct, but the preceding statement is in error.

Additional error messages are printed following the END statement as part of the statement listing section of ATL utility output listings.



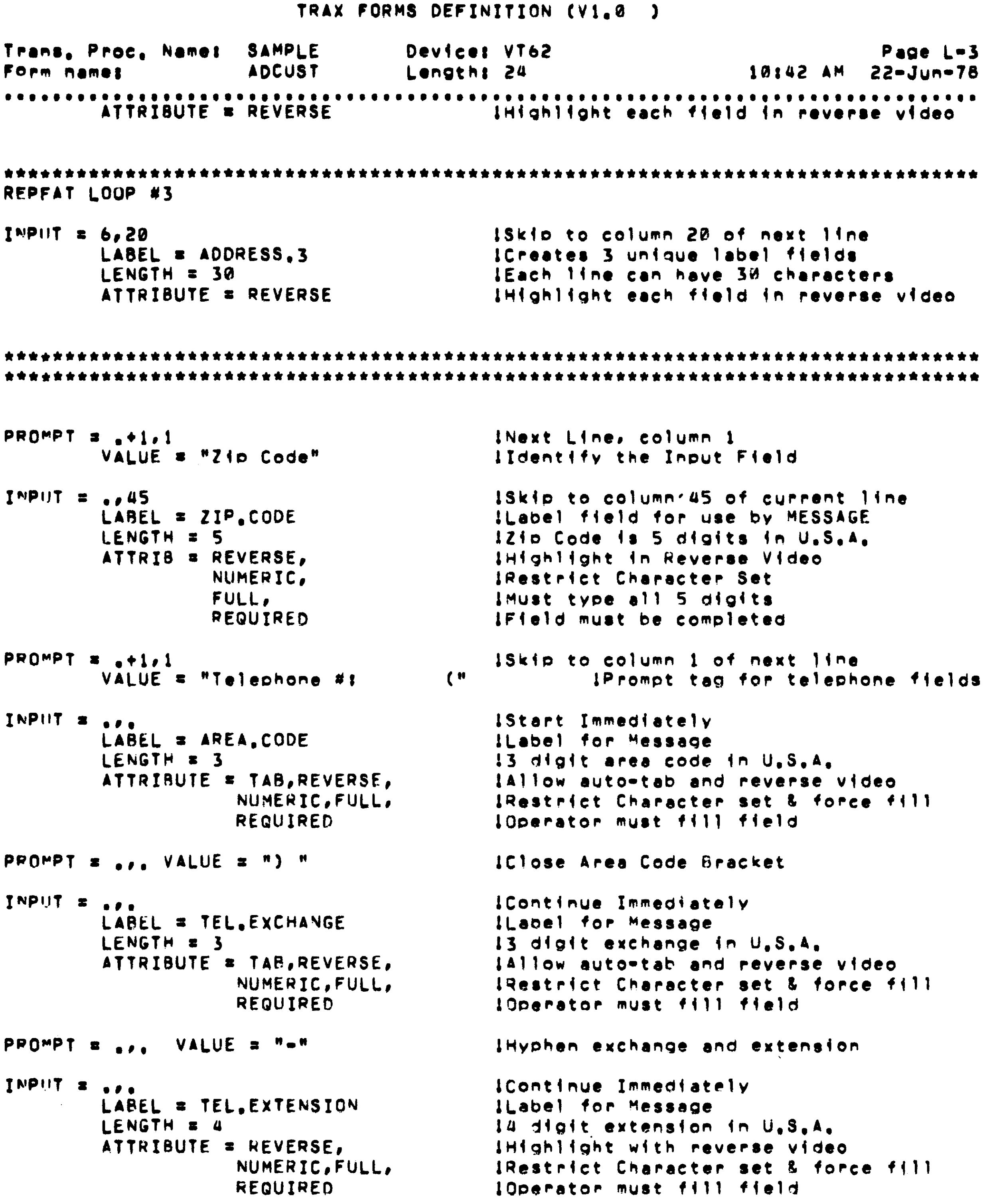
```
                         TRAX FORMS DEFINITION (V1.0  )

Trans. Proc. Name:①SAMPLE      ③Device: VT62                         ⑤Page L-1
Form name:        ②ADCUST      ④Length: 24                 ⑥10:42 AM  22-Jun-78
.................................................................................
!************************************************************************
!
!       Definition of form ADCUST
!
!       This is the only form needed for the ADD Customer
!       Transaction.
!
!************************************************************************
!************************************************************************
!
!       The DEFAULT and FORM Statements are used to set
!       general assumptions for the form definition.
!
!************************************************************************
DEFAULT                                 !To SET Defaults for whole form
        ENABLE = CLOSE                  !Enable all control function keys
        ENABLE = AFFIRM
        CLEAR = " "                     !Set space as default clear character

FORM                                    !Form Statement
        SPLIT = 8                       !8 Line DISPLAY at Top of Form
        BELL = 2                        !Ring Bell Twice for Initial Form

!************************************************************************
!
!       Define the DISPLAY Area, including operator
!       Instructions, and 160 characters for REPLY
!       Text from Error Replies.
!
!************************************************************************
DISPLAY = 2,11                          !Indent 1 line and 11  spaces
        VALUE = "Customer Master File Subsystem - ",
                "Add Customer Transaction"
        ATTRIBUTES = REVERSE,           !Highlight in reverse video
                     NOBLANK            !Don't Erase during Replies

DISPLAY = 4,1                           !Skip 1 Line
        LABEL = REPLY.TEXT.A            !Label for 1st REPLY Line
        LENGTH = 80                     !Line covers Full Screen Width

DISPLAY = 5,1                           !Move to start of next line
        LABEL = REPLY.TEXT.B            !Label for 2nd REPLY Line
        LENGTH = 80                     !Full Screen Width

DISPLAY = .+1,5                         !1st Line of Form Area
        LENGTH = 72
        VALUE = "To Add a Customer to the File, ",
                "Complete all Form Fields and Press ENTER."
        ATTRIBUTE = NOBLANK             !Display on the First Screen
        LABEL = INSTR.TEXT              !Label so you can blank in REPLY

!************************************************************************
!
!       The FORM area is defined beginning at this point.
!
```

Figure 4-10 ANNOTATED Statement Listings (1 of 5)



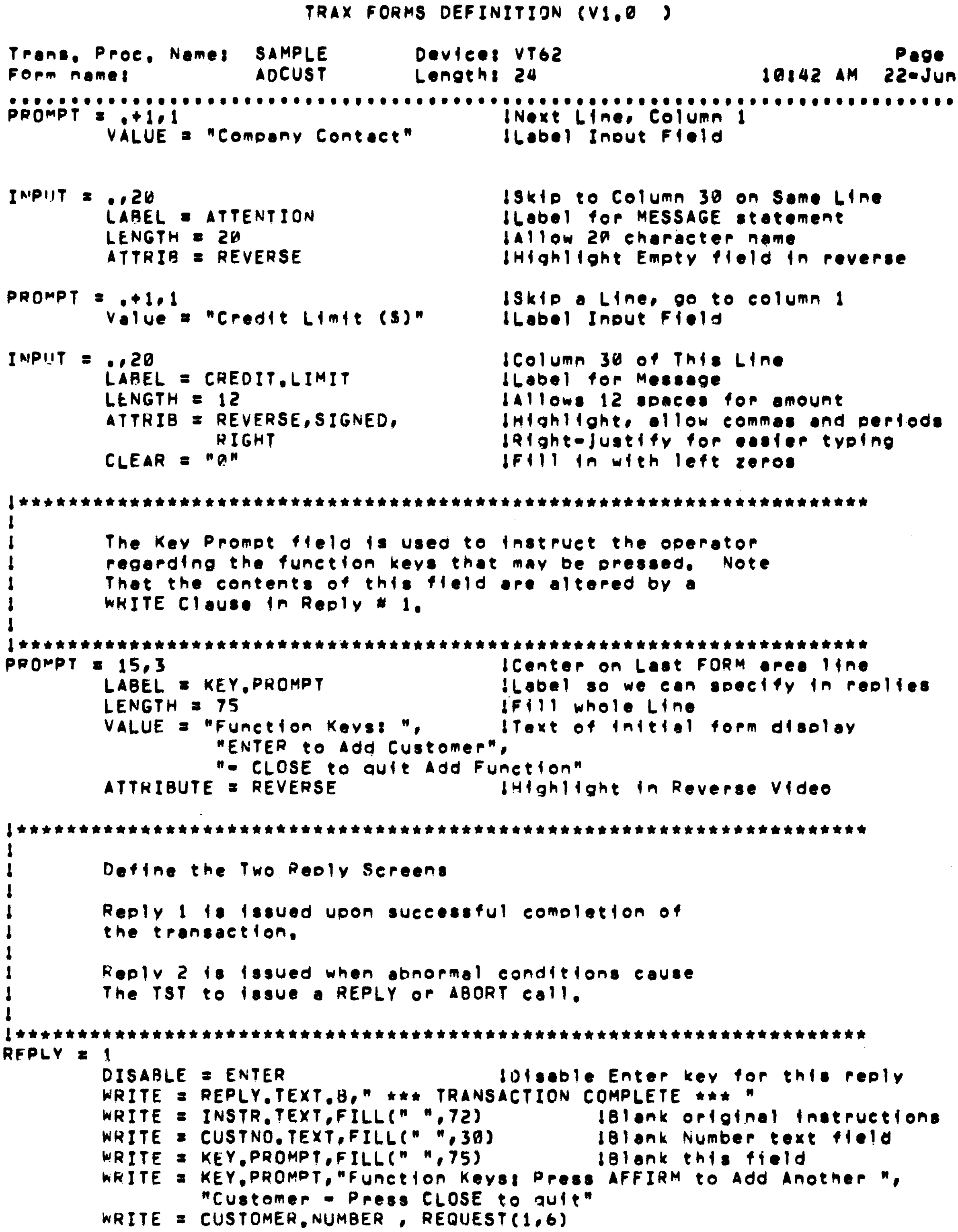
```
                         TRAX FORMS DEFINITION (V1.0  )

Trans. Proc. Name:  SAMPLE      Device: VT62                          Page L-2
Form name:          ADCUST      Length: 24                 10:42 AM  22-Jun-78
........................................................................
!**************************************************************
PROMPT = 1,1                        !Skip a line, move to column 1
        VALUE = "Customer Number"   !Label the empty field

PROMPT = ,,,+4                      !Field starts 4 spaces past prompt
        LABEL = CUSTOMER.NUMBER     !Assign LABEL for MESSAGE and REPLY
        VALUE = "??????"            !Fill with ? - System assigns value
        ATTRIB = REVERSE            !Highlight in REVERSE video

PROMPT = ,,30                       !Skip to Column 30 on this line
        LENGTH = 30                 !Define Length for blanking out later
        VALUE = "(To be Supplied by System)"
        LABEL = CUSTNO.TEXT         !Label for Blanking out Later

PROMPT = ,+1,1                      !Next Line, column 1
        VALUE = "Customer Name"

INPUT = ,,20                        !Field starts 2 spaces past prompt
        LENGTH = 30                 !Define Maximum Name Length
        LABEL = CUSTOMER.NAME       !Label Field for later use
        ATTRIB = REVERSE,           !Highlight with Reverse Video
                 REQUIRED,          !Force Operator to Complete Field
                 LETTERS            !Restrict characters in field


PROMPT = ,+1,1                      !Skip a line, return to column 1
        VALUE = "Address"           !Prompt for three-line address

REPEAT = 3                          !Use a REPEAT Block to code 3 fields
        WITH #1 ="1"                !Use to create unique labels
        WITH #L = 4                 !Start Address on Line 4

INPUT = #L,20                       !Skip to column 20 of next line
        LABEL = ADDRESS.#1          !Creates 3 unique label fields
        LENGTH = 30                 !Each line can have 30 characters
        ATTRIBUTE = REVERSE         !Highlight each field in reverse video

REND                                !End the REPEAT Block

************************************************************************
************************************************************************
REPEAT LOOP #1

INPUT = 4,20                        !Skip to column 20 of next line
        LABEL = ADDRESS.1           !Creates 3 unique label fields
        LENGTH = 30                 !Each line can have 30 characters
        ATTRIBUTE = REVERSE         !Highlight each field in reverse video


************************************************************************
REPEAT LOOP #2

INPUT = 5,20                        !Skip to column 20 of next line
        LABEL = ADDRESS.2           !Creates 3 unique label fields
        LENGTH = 30                 !Each line can have 30 characters
```

Figure 4-10 ANNOTATED Statement Listings (2 of 5)



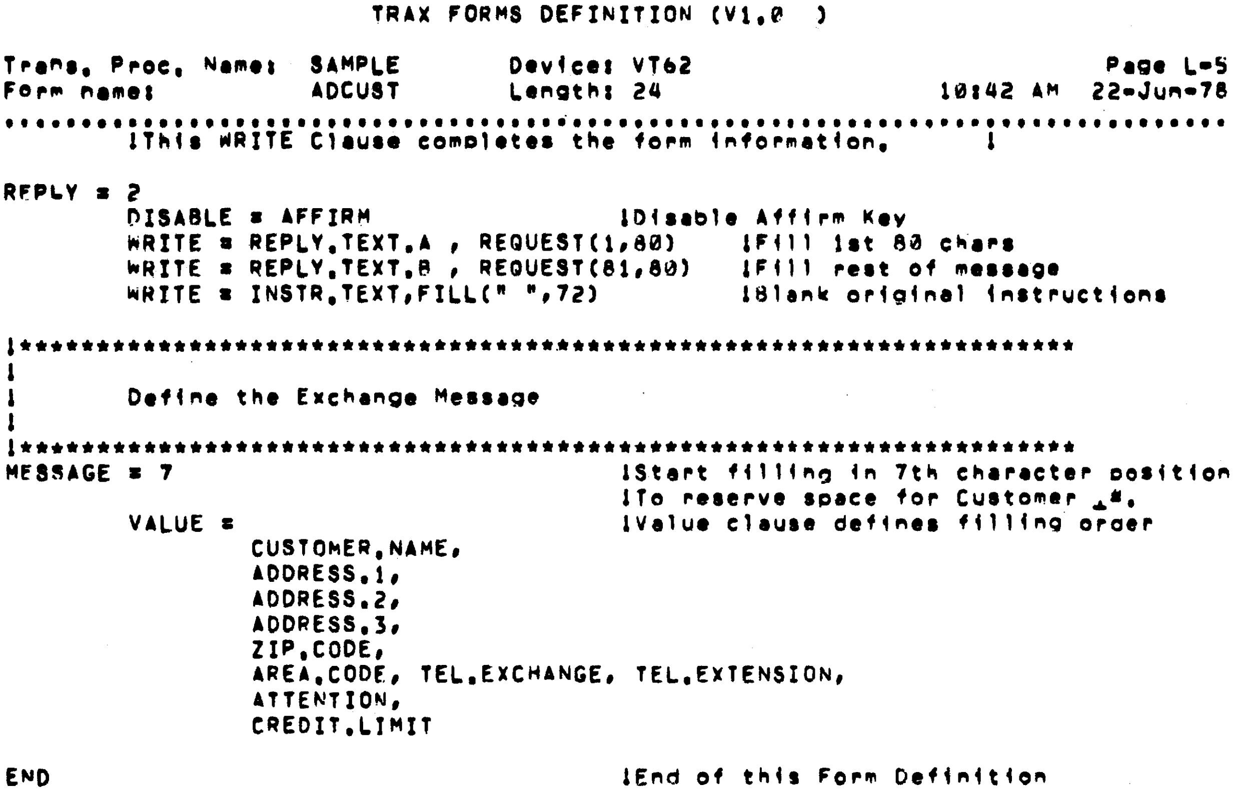
```
                          TRAX FORMS DEFINITION (V1.0  )

Trans, Proc, Name: SAMPLE      Device: VT62                        Page L-3
Form name:         ADCUST      Length: 24                 10:42 AM  22-Jun-78
........................................................................
       ATTRIBUTE = REVERSE            !Highlight each field in reverse video


************************************************************************
REPFAT LOOP #3

INPUT = 6,20                          !Skip to column 20 of next line
       LABEL = ADDRESS,3              !Creates 3 unique label fields
       LENGTH = 30                    !Each line can have 30 characters
       ATTRIBUTE = REVERSE            !Highlight each field in reverse video


************************************************************************
************************************************************************


PROMPT = .+1,1                        !Next Line, column 1
       VALUE = "Zip Code"             !Identify the Input Field

INPUT = .,45                          !Skip to column'45 of current line
       LABEL = ZIP.CODE               !Label field for use by MESSAGE
       LENGTH = 5                     !Zip Code is 5 digits in U.S.A.
       ATTRIB = REVERSE,              !Highlight in Reverse Video
                NUMERIC,              !Restrict Character Set
                FULL,                 !Must type all 5 digits
                REQUIRED              !Field must be completed

PROMPT = .+1,1                        !Skip to column 1 of next line
       VALUE = "Telephone #:      ("          !Prompt tag for telephone fields

INPUT = .,.                           !Start Immediately
       LABEL = AREA.CODE              !Label for Message
       LENGTH = 3                     !3 digit area code in U.S.A.
       ATTRIBUTE = TAB,REVERSE,       !Allow auto-tab and reverse video
                   NUMERIC,FULL,      !Restrict Character set & force fill
                   REQUIRED           !Operator must fill field

PROMPT = .,. VALUE = ") "             !Close Area Code Bracket

INPUT = .,.                           !Continue Immediately
       LABEL = TEL.EXCHANGE           !Label for Message
       LENGTH = 3                     !3 digit exchange in U.S.A.
       ATTRIBUTE = TAB,REVERSE,       !Allow auto-tab and reverse video
                   NUMERIC,FULL,      !Restrict Character set & force fill
                   REQUIRED           !Operator must fill field

PROMPT = .,.  VALUE = "-"             !Hyphen exchange and extension

INPUT = .,.                           !Continue Immediately
       LABEL = TEL.EXTENSION          !Label for Message
       LENGTH = 4                     !4 digit extension in U.S.A.
       ATTRIBUTE = REVERSE,           !Highlight with reverse video
                   NUMERIC,FULL,      !Restrict Character set & force fill
                   REQUIRED           !Operator must fill field
```

Figure 4-10 ANNOTATED Statement Listings (3 of 5)



```
                          TRAX FORMS DEFINITION (V1.0  )

Trans. Proc. Name:  SAMPLE       Device: VT62                          Page L-4
Form name:          ADCUST       Length: 24                 10:42 AM  22-Jun-78
................................................................................
PROMPT = ,+1,1                            !Next Line, Column 1
        VALUE = "Company Contact"         !Label Inout Field


INPUT = ,,20                              !Skip to Column 30 on Same Line
        LABEL = ATTENTION                 !Label for MESSAGE statement
        LENGTH = 20                       !Allow 20 character name
        ATTRIB = REVERSE                  !Highlight Empty field in reverse

PROMPT = ,+1,1                            !Skip a Line, go to column 1
        Value = "Credit Limit ($)"        !Label Inout Field

INPUT = ,,20                              !Column 30 of This Line
        LABEL = CREDIT.LIMIT              !Label for Message
        LENGTH = 12                       !Allows 12 spaces for amount
        ATTRIB = REVERSE,SIGNED,          !Highlight, allow commas and periods
                 RIGHT                    !Right-Justify for easier typing
        CLEAR = "0"                       !Fill in with left zeros

!*******************************************************************
!
!       The Key Prompt field is used to instruct the operator
!       regarding the function keys that may be pressed.  Note
!       That the contents of this field are altered by a
!       WRITE Clause in Reply # 1.
!
!*******************************************************************
PROMPT = 15,3                             !Center on Last FORM area line
        LABEL = KEY.PROMPT                !Label so we can specify in replies
        LENGTH = 75                       !Fill whole Line
        VALUE = "Function Keys: ",        !Text of initial form display
                "ENTER to Add Customer",
                "- CLOSE to quit Add Function"
        ATTRIBUTE = REVERSE               !Highlight in Reverse Video

!*******************************************************************
!
!       Define the Two Reply Screens
!
!       Reply 1 is issued upon successful completion of
!       the transaction.
!
!       Reolv 2 is issued when abnormal conditions cause
!       The TST to issue a REPLY or ABORT call.
!
!*******************************************************************
REPLY = 1
        DISABLE = ENTER                    !Disable Enter key for this reply
        WRITE = REPLY.TEXT.8," *** TRANSACTION COMPLETE *** "
        WRITE = INSTR.TEXT,FILL(" ",72)           !Blank original instructions
        WRITE = CUSTNO.TEXT,FILL(" ",30)          !Blank Number text field
        WRITE = KEY.PROMPT,FILL(" ",75)           !Blank this field
        WRITE = KEY.PROMPT,"Function Keys: Press AFFIRM to Add Another ",
                "Customer - Press CLOSE to quit"
        WRITE = CUSTOMER.NUMBER , REQUEST(1,6)
```

Figure 4-10 ANNOTATED Statement Listings (4 of 5)



```
                          TRAX FORMS DEFINITION (V1.0  )

Trans. Proc. Name:  SAMPLE        Device: VT62                        Page L-5
Form name:          ADCUST        Length: 24                10:42 AM  22-Jun-78
..........................................................................
        !This WRITE Clause completes the form information.      !

REPLY = 2
        DISABLE = AFFIRM                        !Disable Affirm Key
        WRITE = REPLY.TEXT.A , REQUEST(1,80)    !Fill 1st 80 chars
        WRITE = REPLY.TEXT.B , REQUEST(81,80)   !Fill rest of message
        WRITE = INSTR.TEXT,FILL(" ",72)         !Blank original instructions

!*******************************************************************
!
!       Define the Exchange Message
!
!*******************************************************************
MESSAGE = 7                                     !Start filling in 7th character position
                                                !To reserve space for Customer _#.
        VALUE =                                 !Value clause defines filling order
                CUSTOMER.NAME,
                ADDRESS.1,
                ADDRESS.2,
                ADDRESS.3,
                ZIP.CODE,
                AREA.CODE, TEL.EXCHANGE, TEL.EXTENSION,
                ATTENTION,
                CREDIT.LIMIT

END                                             !End of this Form Definition
```

Figure 4-10 ANNOTATED Statement Listings (5 of 5)

4.12.2 Summary of General Form Parameters – Page F-1

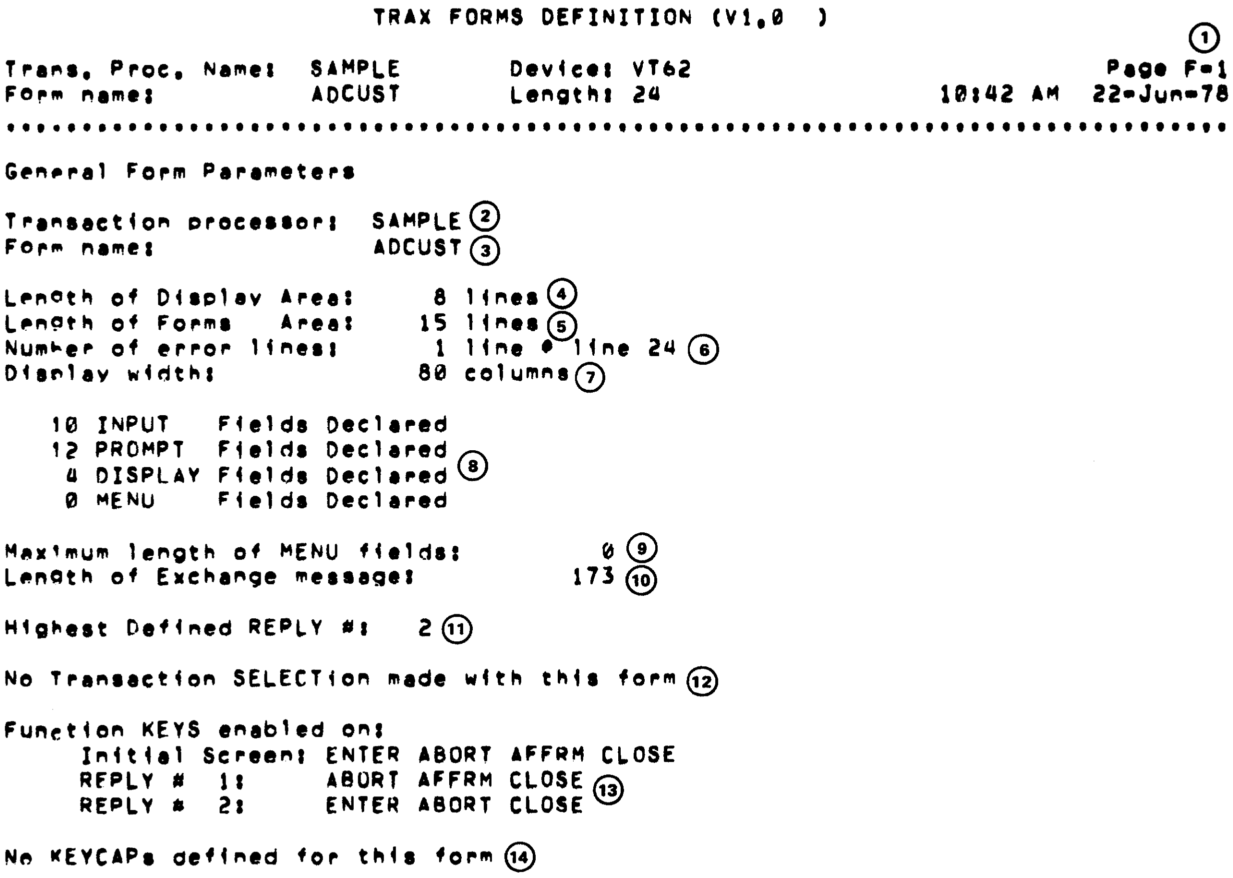
Page F-1 is produced by the ATL utility from the source statement file ADCUST.ATL. The information on this page is a general summary of the transaction processor and form name, the form dimensions, the number and type of field declarations, the length of MENU fields and exchange messages, the number of replies, and a listing of enabled function keys and non-standard KEYCAP text defined for this form.

1. The "F" prefix is assigned by the ATL utility to pages that describe the parameters and fields of a form definition. The "F" pages of the output listing are produced by the ATL utility when you specify ALL or FORMS in response to the ATL utility prompt:

   List <ALL>?

2. SAMPLE is the six-character name of the transaction processor that will use the form. The form is added to the form definition file for SAMPLE. ([1,300] SAMPLE.FDF) The ATL utility also prints this name at the top of each compiler output page. If you specified the COMPILE command to the ATL utility, this name would not be displayed since COMPILE does not modify a forms definition file.
3. The form name is ADCUST. This is the six-character form name used to specify this form definition in the forms definition file, as well as in the transaction and station definitions for the transaction processor SAMPLE. The form name also appears at the top of each output page. If you specified the COMPILE command to the ATL utility, this name would be displayed as ?????? since COMPILE does not modify a forms definition file.
4. When you specified SPLIT = 8 in the FORM statement, you defined this form with a Display area of eight screen lines.
5. A VT62 terminal has 24 lines on its screen, one of which is always used for terminal-generated error messages. Specifying an additional 8 lines for a Display area leaves 15 screen lines for the FORM area.
6. Form definitions used on a VT62 terminal always reserve the 24th (last) screen line for terminal error messages.
7. The VT62 terminal has 80 columns on its screen. The ATL utility assumes forms used on a VT62 have 80 column screen width.
8. The ATL utility counts the number of each type of field defined on the form.
9. No MENU field has been defined. If one had been, the length of the longest MENU field would appear here. That length would be used to define the length of a MENU entry in the exchange message when the VALUE = MENU clause was specified in a MESSAGE statement.
10. The length of the exchange message that will be constructed from this form. The exchange message length is determined by the position of the last field defined in the exchange message.
11. You have defined replies 1 and 2, so the highest reply number is 2. If you had defined replies 1 and 5, the highest reply number would have been 5.
12. This form is not a transaction selection form. If a form definition includes a SELECT clause in its FORM statement, it is a transaction selection form.
13. This section summarizes the function keys enabled when the form is initially displayed, and those enabled after each of the defined replies has been sent. The function keys are those normally enabled, as modified by DEFAULT statements in the source listings and those keys changed by explicity ENABLE or DISABLE clauses in FORM and REPLY statements.

14. This line informs you that function key identifiers have not been used in the generation of the exchange message. A function key identifier can be inserted as part of the exchange message by using the KEY argument in the VALUE clause of the MESSAGE statement. This causes a predefined text string to be included in the exchange message; this identifies which function key you pressed. Each data entry function key has a default identifying text string, called a KEYCAP. You can modify this text by using the KEYCAP clause in the FORM statement. The function key identification text will be shown in this section of the compiler output listing whenever KEYCAP text is used in the exchange message.



```
                           TRAX FORMS DEFINITION (V1.0  )
                                                                              ①
Trans. Proc. Name:  SAMPLE        Device: VT62                              Page F-1
Form name:          ADCUST        Length: 24                     10:42 AM  22-Jun-78
*********************************************************************************

General Form Parameters

Transaction processor:   SAMPLE②
Form name:               ADCUST③

Length of Display Area:    8 lines④
Length of Forms   Area:   15 lines⑤
Number of error lines:     1 line - line 24 ⑥
Display width:            80 columns⑦

    10 INPUT   Fields Declared
    12 PROMPT  Fields Declared
     4 DISPLAY Fields Declared ⑧
     0 MENU    Fields Declared

Maximum length of MENU fields:        0 ⑨
Length of Exchange message:         173 ⑩

Highest Defined REPLY #:   2 ⑪

No Transaction SELECTion made with this form ⑫

Function KEYS enabled on:
     Initial Screen: ENTER ABORT AFFRM CLOSE
     REPLY #  1:     ABORT AFFRM CLOSE ⑬
     REPLY #  2:     ENTER ABORT CLOSE

No KEYCAPs defined for this form ⑭
```

Figure 4-11 ANNOTATED Form Listing Page F-1

4.12.3 Summary of Input Field Declarations – Page F-2

1. The standard attributes are the set of default attributes assumed for Input fields by the ATL utility. If you specify a field with attributes other than those in the Standard attributes list, they will be shown in parentheses to the right of the field name in the input field declarations table.
2. This line shows the table column headings. The columns contain the following information:

   FLD # – is a sequential number given to each INPUT field in the source listing. When you press the NEXT FIELD key on your terminal, the cursor moves from field to field in ascending order.

ROW # – is the row on the screen where the field is located. For INPUT fields, this is a row in the Form area.

COL # – is the column (1 is the leftmost column, 80 the rightmost) at which the field begins.

LNG – is the length of the field in character positions.

CLEAR CHAR – is the character used to fill the field in the initial form display.

LABEL – is the label, or name, assigned to the field. Only those fields referenced by other statements in the form definition need labels. Only the first 16 characters of the label are shown on the listing.

(ATTRIBUTES) – are the attributes specified for a field that are not listed as standard attributes at the top of the table.

4.12.4 Summary of Prompt Field Declarations – Page F-2

3. The standard attributes are the set of default attributes assumed for Prompt fields by the ATL utility. If you specify a field with attributes other than those in the Standard attributes list, they will be shown in parentheses to the left of the field name in the Prompt field declarations table.
4. This line shows the table column headings. The columns contain the following information:

FLD # – is a sequential number given to each Prompt field in the source listing. You cannot move the cursor to a Prompt field.

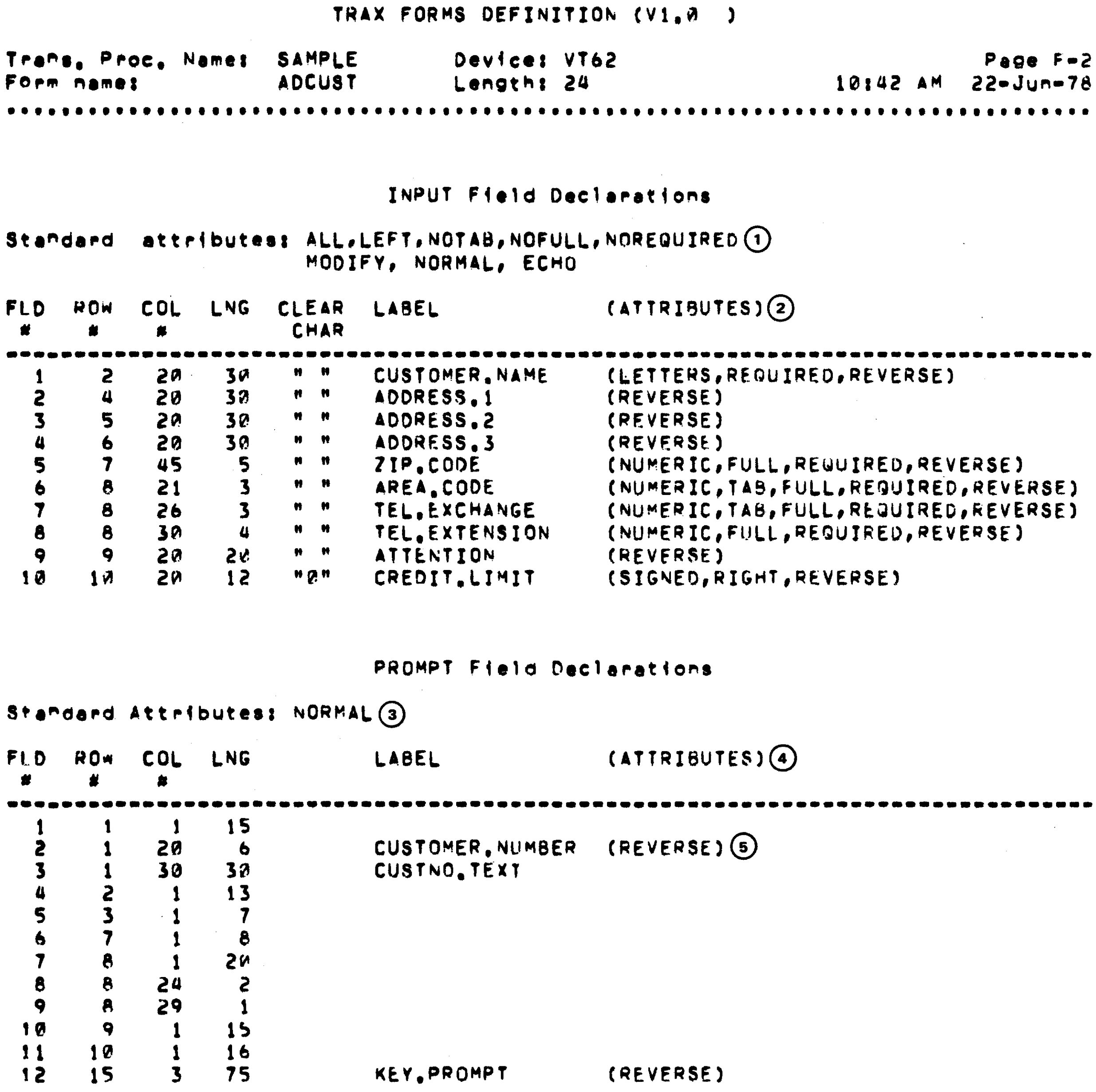
ROW # – is the row on the screen where the field is located. For Prompt fields, this is a row in the Form area.

COL # – is the column (1 is the leftmost column, 80 the rightmost) at which the field will begin.

LNG – is the length of the field in character positions.

LABEL – is the label, or name, assigned to the field. Only those Prompt fields referenced by WRITE clauses in REPLY statements need labels. Only the first 16 characters of the label are printing on the listing.

(ATTRIBUTES) – are the attributes specified for a field that are not listed as standard attributes at the top of the table.



```
                          TRAX FORMS DEFINITION (V1.0  )

Trans. Proc. Name:  SAMPLE        Device: VT62                            Page F-2
Form name:          ADCUST        Length: 24                   10:42 AM  22-Jun-78
..............................................................................


                             INPUT Field Declarations

Standard  attributes: ALL,LEFT,NOTAB,NOFULL,NOREQUIRED①
                      MODIFY, NORMAL, ECHO

FLD  ROW  COL  LNG  CLEAR  LABEL             (ATTRIBUTES)②
 #    #    #         CHAR
------------------------------------------------------------------------------
  1    2   20   30   " "   CUSTOMER.NAME     (LETTERS,REQUIRED,REVERSE)
  2    4   20   30   " "   ADDRESS.1         (REVERSE)
  3    5   20   30   " "   ADDRESS.2         (REVERSE)
  4    6   20   30   " "   ADDRESS.3         (REVERSE)
  5    7   45    5   " "   ZIP.CODE          (NUMERIC,FULL,REQUIRED,REVERSE)
  6    8   21    3   " "   AREA.CODE         (NUMERIC,TAB,FULL,REQUIRED,REVERSE)
  7    8   26    3   " "   TEL.EXCHANGE      (NUMERIC,TAB,FULL,REQUIRED,REVERSE)
  8    8   30    4   " "   TEL.EXTENSION     (NUMERIC,FULL,REQUIRED,REVERSE)
  9    9   20   20   " "   ATTENTION         (REVERSE)
 10   10   20   12   "0"   CREDIT.LIMIT      (SIGNED,RIGHT,REVERSE)


                            PROMPT Field Declarations

Standard Attributes: NORMAL③

FLD  ROW  COL  LNG         LABEL             (ATTRIBUTES)④
 #    #    #
------------------------------------------------------------------------------
  1    1    1   15
  2    1   20    6         CUSTOMER.NUMBER   (REVERSE)⑤
  3    1   30   30         CUSTNO.TEXT
  4    2    1   13
  5    3    1    7
  6    7    1    8
  7    8    1   20
  8    8   24    2
  9    8   29    1
 10    9    1   15
 11   10    1   16
 12   15    3   75         KEY.PROMPT        (REVERSE)
```

Figure 4-12 ANNOTATED Form Listing Page F-2

4.12.5 Summary of DISPLAY Field Declarations – Page F-3

1. The standard attributes are the set of default attributes assumed for DISPLAY fields by the ATL utility. If you specify a field with attributes other than those in the Standard attributes list, they will be shown in parentheses to the left of the field name in the DISPLAY field declarations table.
2. This line shows the table column headings. The columns contain the following information:

   FLD # – is a sequential number given to each DISPLAY field in the source listing. You cannot move the cursor to a DISPLAY field.

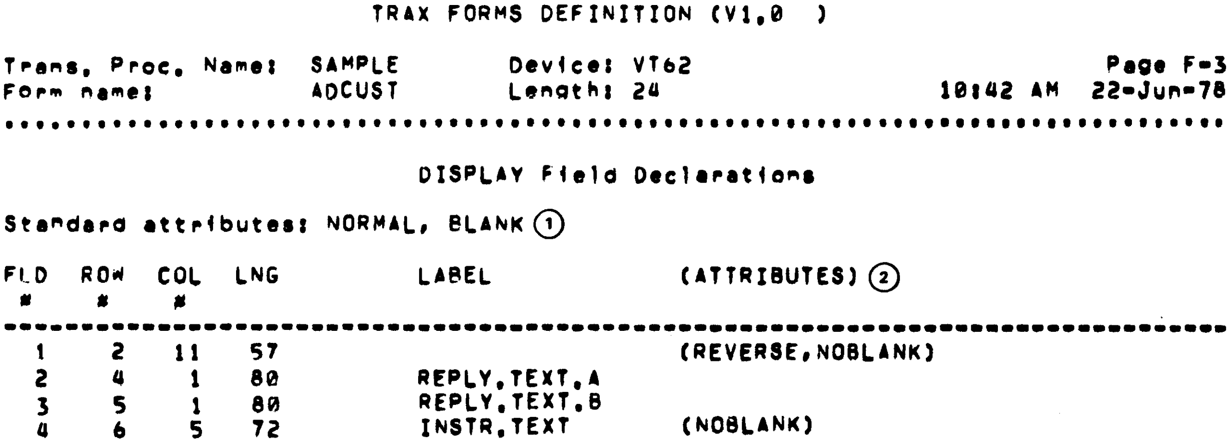
   ROW # – is the row on the screen where the field is located. For DISPLAY fields, this is a row in the Display area.

   COL # – is the column (1 is the leftmost column, 80 the rightmost) at which the field will begin.

   LNG – is the length of the field in character positions.

   LABEL – is the label, or name, assigned to the field. Only those DISPLAY fields referenced by WRITE clauses in REPLY statements need labels. Only the first 16 characters of the label are printed on the listing.

   (ATTRIBUTES) – are the attributes specified for a field that are not listed as standard attributes at the top of the table.



```
                              TRAX FORMS DEFINITION (V1.0  )

Trans. Proc. Name: SAMPLE        Device: VT62                              Page F-3
Form name:         ADCUST        Length: 24                       10:42 AM  22-Jun-78
.................................................................................

                              DISPLAY Field Declarations

Standard attributes: NORMAL, BLANK ①

FLD  ROW  COL  LNG           LABEL               (ATTRIBUTES) ②
 #    #    #
=================================================================================
  1    2   11   57                                (REVERSE,NOBLANK)
  2    4    1   80           REPLY.TEXT.A
  3    5    1   80           REPLY.TEXT.B
  4    6    5   72           INSTR.TEXT           (NOBLANK)
```

Figure 4-13 ANNOTATED Form Listing Page F-3

4.12.6 Summary of MENU Field Declarations –Form "SELECT" Page F-3

1. This line shows the table column headings. The columns contain the following information:

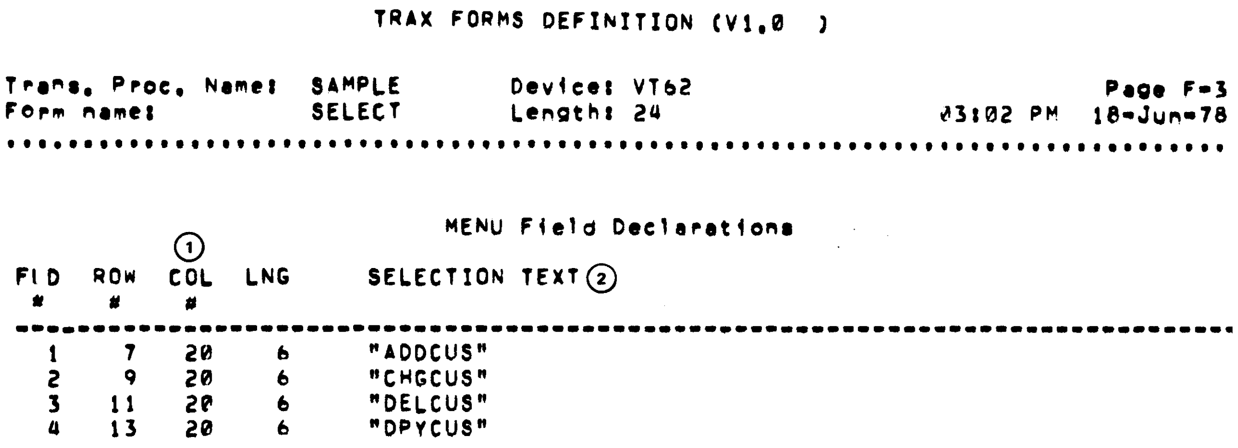
   FLD # – is a sequential number given to each MENU field in the source listing. You can move the cursor to a MENU field. Pressing the SELECT key causes the VT62 to highlight the entire selected MENU field in reverse video.

ROW # – is the row on the screen where the field is located. For MENU fields, this is a row in the DISPLAY area.

COL # – is the column (1 is the leftmost column, 80 the rightmost) at which the MENU field will begin.

LNG – is the length of the field in character positions.

2. The text string shown in this column are the strings that are transmitted as part of the MESSAGE statement VALUE = MENU clause, or as the SELECT clause MENU parameter in a transaction selection form. If you specify a MENU field with more than six-characters as part of a transaction selection form, only the first six characters are used by the transaction processor when it invokes the next transaction. This is because transaction names are limited to six characters in length.



```
                          TRAX FORMS DEFINITION (V1.0  )

Trans. Proc. Name: SAMPLE       Device: VT62                           Page F-3
Form name:         SELECT       Length: 24                 03:02 PM  18-Jun-78
.............................................................................

                              MENU Field Declarations
          ①
FLD  ROW  COL  LNG      SELECTION TEXT②
 #    #    #
------------------------------------------------------------------------------
  1    7   20    6      "ADDCUS"
  2    9   20    6      "CHGCUS"
  3   11   20    6      "DELCUS"
  4   13   20    6      "DPYCUS"
```

Figure 4-14 ANNOTATED Form Listing "SELECT" Page F-3

4.12.7 Summary of PRINT Field Declarations

–Form "PRTCUS" Page F-3

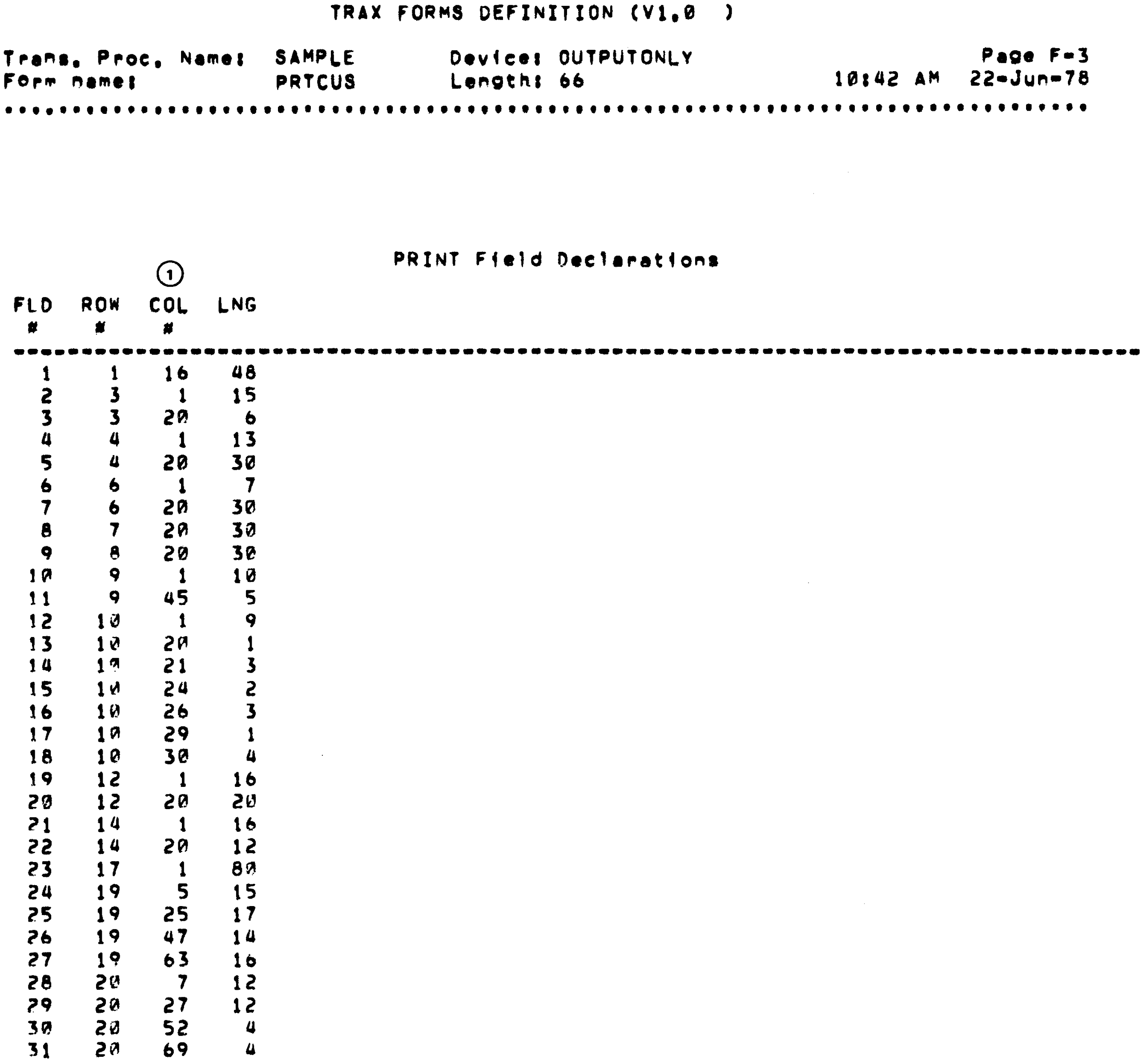
1. This line shows the table column headings. The columns contain the following information:

FLD # – is a sequential number given to each PRINT field in the source listing. This number is the order in which the field is printed, not the order in which it was specified.

ROW # – is the row on the screen where the field is located. For PRINT fields, this is the distance in lines from the top of the form.

COL # – is the column (1 is the leftmost column, 132 the rightmost) at which the field will begin.

LNG – is the length of the field in character positions.



```
                              TRAX FORMS DEFINITION (V1.0  )

Trans. Proc. Name:  SAMPLE        Device: OUTPUTONLY                      Page F-3
Form name:          PRTCUS        Length: 66                   10:42 AM  22-Jun-78
................................................................................



                                   PRINT Field Declarations
               ①
 FLD   ROW   COL   LNG
  #     #     #
----------------------------------------------------------------------------------
   1     1    16    48
   2     3     1    15
   3     3    20     6
   4     4     1    13
   5     4    20    30
   6     6     1     7
   7     6    20    30
   8     7    20    30
   9     8    20    30
  10     9     1    10
  11     9    45     5
  12    10     1     9
  13    10    20     1
  14    10    21     3
  15    10    24     2
  16    10    26     3
  17    10    29     1
  18    10    30     4
  19    12     1    16
  20    12    20    20
  21    14     1    16
  22    14    20    12
  23    17     1    80
  24    19     5    15
  25    19    25    17
  26    19    47    14
  27    19    63    16
  28    20     7    12
  29    20    27    12
  30    20    52     4
  31    20    69     4
```

Figure 4-15 ANNOTATED Form Listing "PRTCUS" Page F-3

4.12.8 Exchange Message Layout – Page F-4

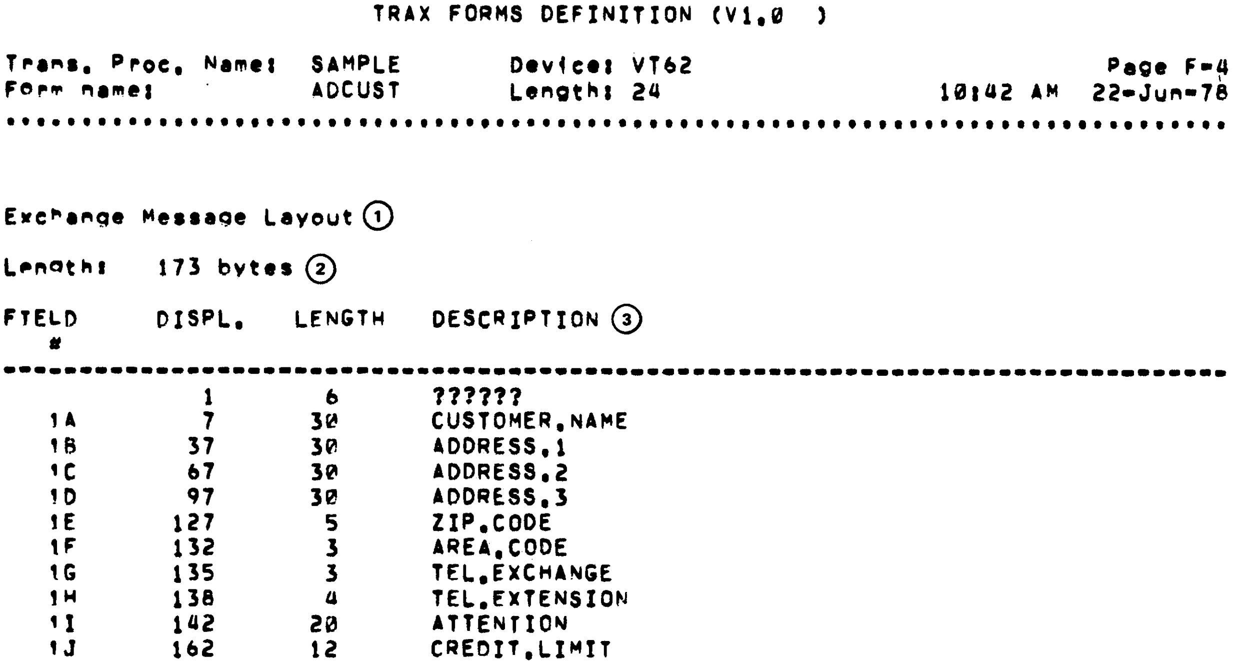
1. This table describes the exchange message constructed when the user presses an enabled data entry function key.
2. The number of characters in the exchange message.
3. The column headings for the exchange message table data.

FIELD # identifies the MESSAGE statement and VALUE clause argument where the field was referenced. The FIELD consists of a number and a letter. The number refers to the particular MESSAGE statement where the field was referenced. The letter refers to the particular VALUE argument within the MESSAGE statement.

DISPL. is the character position in the exchange message where the field is inserted into the message.

LENGTH is the number of characters in the field.

DESCRIPTION identifies the data placed into the exchange message at the position specified in the DISPL. column. This column shows field labels, literal text, keycap values, and other legal arguments to a MESSAGE statement VALUE clause.



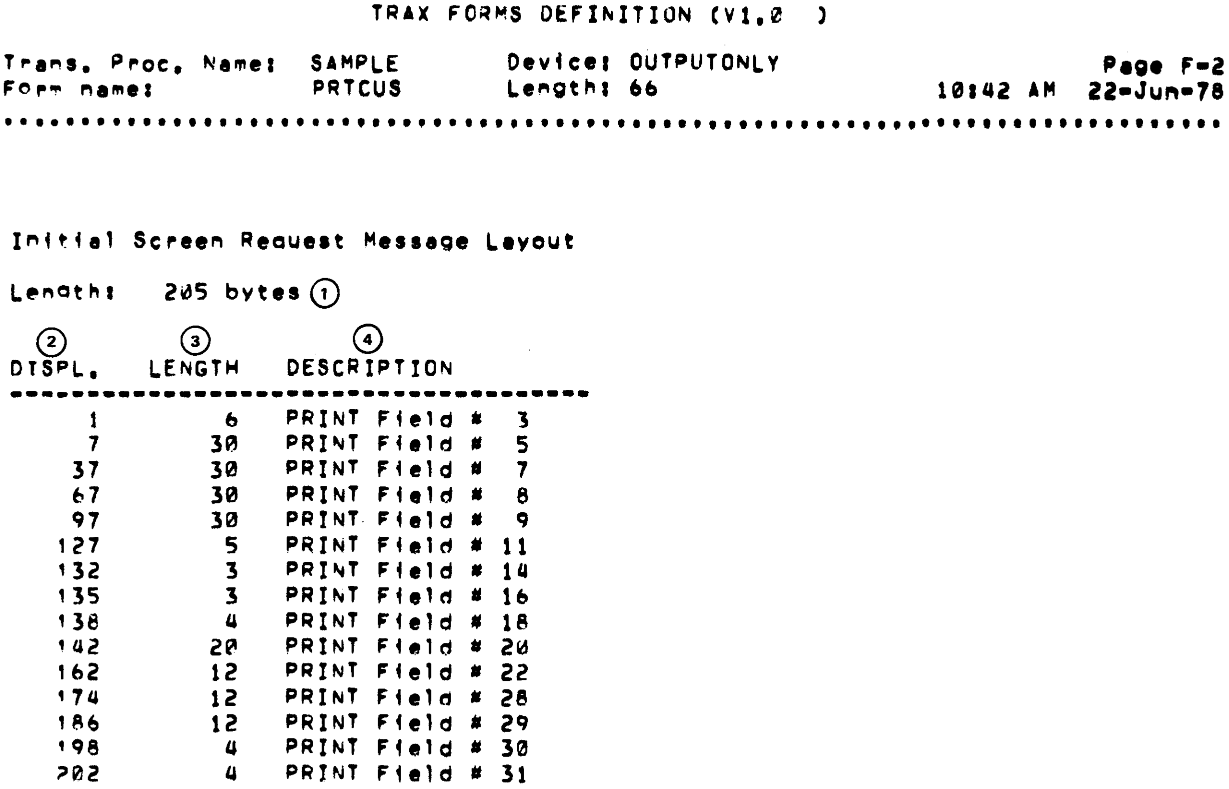
```
                              TRAX FORMS DEFINITION (V1.0  )

Trans. Proc. Name:   SAMPLE          Device: VT62                               Page F-4
Form name:           ADCUST          Length: 24                      10:42 AM  22-Jun-78
•••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••••


Exchange Message Layout ①

Length:     173 bytes ②

FIELD       DISPL.     LENGTH     DESCRIPTION ③
  #
-----------------------------------------------------------------------------------------
                1         6       ??????
  1A            7        30       CUSTOMER.NAME
  1B           37        30       ADDRESS.1
  1C           67        30       ADDRESS.2
  1D           97        30       ADDRESS.3
  1E          127         5       ZIP.CODE
  1F          132         3       AREA.CODE
  1G          135         3       TEL.EXCHANGE
  1H          138         4       TEL.EXTENSION
  1I          142        20       ATTENTION
  1J          162        12       CREDIT.LIMIT
```

Figure 4-16 ANNOTATED Form Listing Page F-4

4.12.9 Initial Screen Request Layout

–Form PRTCUS Page F-2

1. This is the total length of data expected from the requesting response message. If the requesting response message is shorter than this length, unpredictable results occur.
2. DISPL. refers to the character position in the requesting response message where the data specified in the REQUEST function begins.
3. LENGTH refers to the length parameter specified in the request function. If this value is greater than the length of the receiving form field, the receiving field is automatically extended to that length. If the length is less than that specified for the receiving field, the field is always filled beginning with the leftmost character. No change is made to character positions in that field beyond the length specified in the REQUEST function.
4. DESCRIPTION refers to the field number or label where the REQUEST function places the data from the requesting response message.



```
                         TRAX FORMS DEFINITION (V1.0  )

Trans. Proc. Name:  SAMPLE       Device: OUTPUTONLY                       Page F-2
Form name:          PRTCUS       Length: 66                  10:42 AM  22-Jun-78
.................................................................................


Initial Screen Request Message Layout

Length:    205 bytes ①

  ②         ③          ④
DISPL.    LENGTH    DESCRIPTION
-------------------------------------
     1         6    PRINT Field #  3
     7        30    PRINT Field #  5
    37        30    PRINT Field #  7
    67        30    PRINT Field #  8
    97        30    PRINT Field #  9
   127         5    PRINT Field # 11
   132         3    PRINT Field # 14
   135         3    PRINT Field # 16
   138         4    PRINT Field # 18
   142        20    PRINT Field # 20
   162        12    PRINT Field # 22
   174        12    PRINT Field # 28
   186        12    PRINT Field # 29
   198         4    PRINT Field # 30
   202         4    PRINT Field # 31
```

Figure 4-17 ANNOTATED Form Listing "PRTCUS" Page F-2

4.12.10 Reply Message Layout – Page F-5

1. This table describes the format of the response message expected from a TST when it activates a REPLY. Two tables are shown, Reply #1, and Reply #2.
2. The number of characters expected in the requesting response message. For Reply #1, this value is 6.

3. The column headings for the reply message table data.

   FIELD # is the number of the REPLY statement WRITE clause that first references this field.

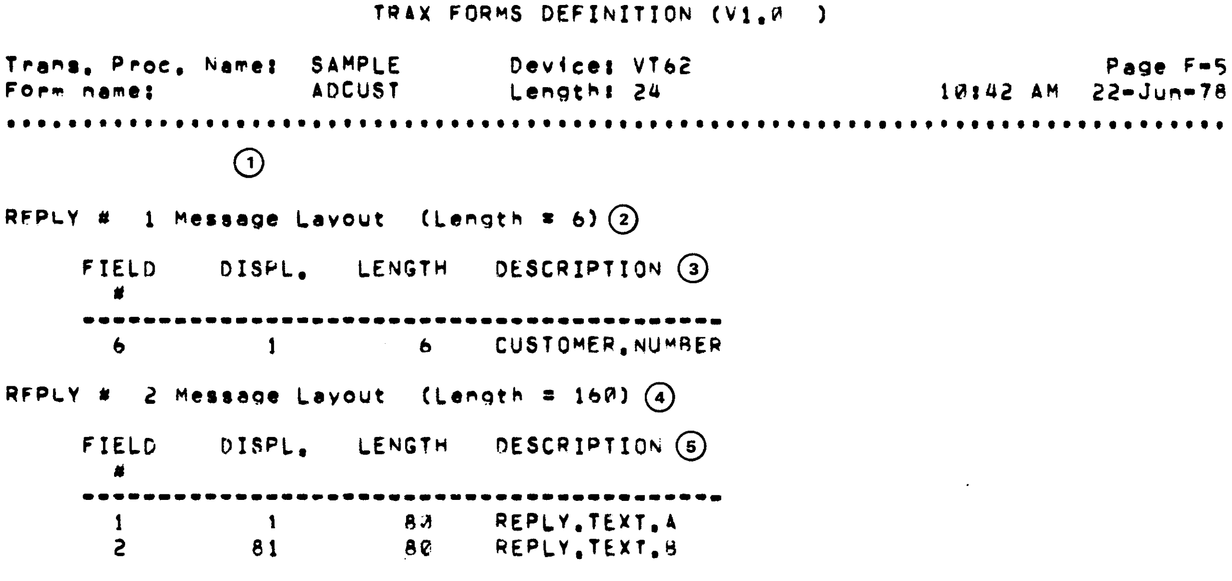
   DISPL. is the number of character positions past the beginning of the response message where the field contents are located. In REPLY #1, the field contents start in the first character position of the response message.

   LENGTH is the number of characters in the field. Six characters are required for the customer number.

   DESCRIPTION identifies the destination of the field data extracted from the response message. In this example, the field label CUSTOMER.NUMBER indicates that the data goes into that field.

   The second table shows the format of the response message that a TST sends when it activates reply #2.

4. The length of this response message is 160 characters.
5. The response message fills two 80-character screen lines. The first 80 characters of the message go into line A, and the remaining characters go into line B.



```
                          TRAX FORMS DEFINITION (V1.Ø  )

Trans. Proc. Name:  SAMPLE       Device: VT62                        Page F-5
Form name:          ADCUST       Length: 24                 1Ø:42 AM  22-Jun-78
................................................................................

                ①

REPLY #  1 Message Layout  (Length = 6) ②

    FIELD    DISPL.    LENGTH   DESCRIPTION ③
      #
    ------------------------------------------
      6        1          6     CUSTOMER.NUMBER

REPLY #  2 Message Layout  (Length = 16Ø) ④

    FIELD    DISPL.    LENGTH   DESCRIPTION ⑤
      #
    ------------------------------------------
      1        1         8Ø     REPLY.TEXT.A
      2       81         8Ø     REPLY.TEXT.B
```

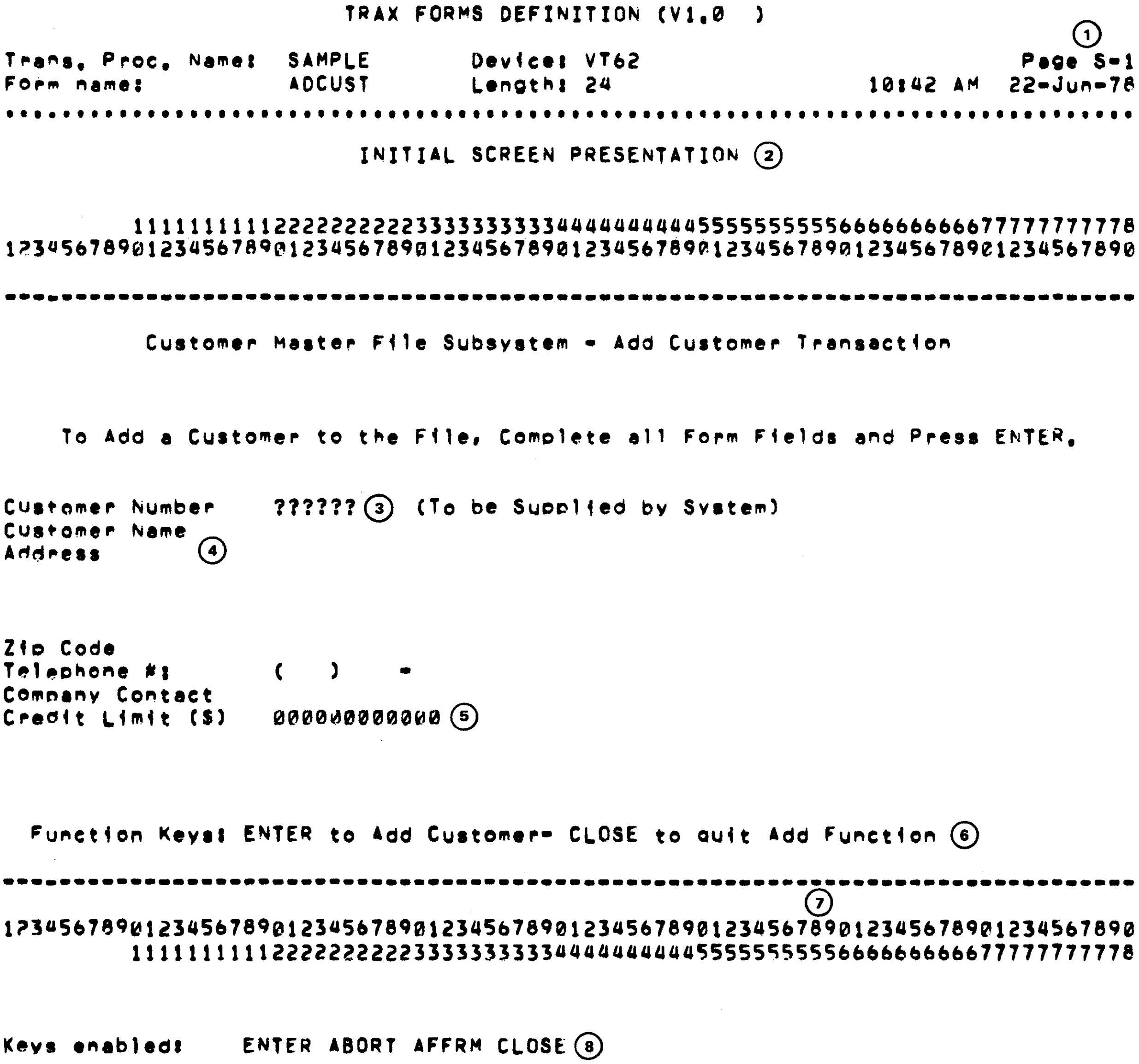
Figure 4-18 ANNOTATED Form Listing Page F-5

4.12.11 Screen Display Format Listings

The “S” in the page number indicates that this is a screen display listing produced by the ATL utility. These listings are produced when you specify ALL or FORMS in response to the List prompt in the ATL utility dialog.

4.12.11.1 Initial Screen Display – Page S-1

1. Page S-1 shows the form as it first appears on the terminal screen, that is, before it has been modified by the user or by any defined replies.
2. The question marks in this field were specified in the VALUE clause of the PROMPT statement that defined the field.
3. Since a blank was specified as the clear character for the name, address, zip code, telephone, and contact fields, no characters illustrate the Input fields. For debugging purposes, it is suggested that some displayable character, such as a period be specified as a clear character. The period is repeated for the number of characters specified in the field definition. Once you are satisfied with the form layout, simply edit the source file to change the FORM statement CLEAR clause to a blank.
4. This field had a clear character of "0" specified. This allows you to insure that all characters in the field are numeric, and that leading zeros are placed in the exchange message.
5. A prompt field tells you about the function keys that can be pressed. The text of this message is changed by reply 1.
6. A series of numbers rules off the columns of the form for debugging and design purposes.
7. This line tells you which function keys are enabled when the form is first displayed. This corresponds to item (13) on page F-1.

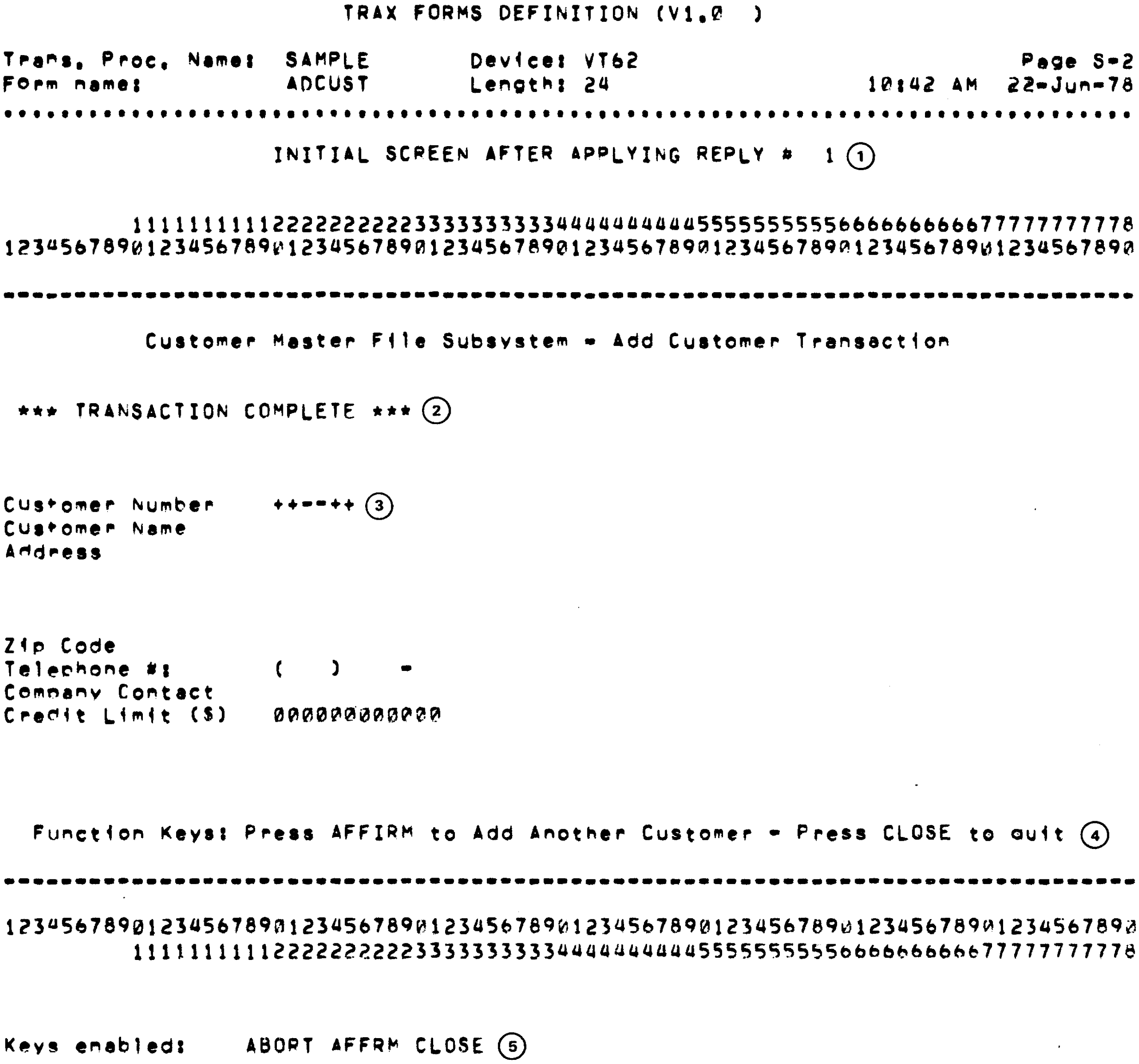


```
                            TRAX FORMS DEFINITION (V1.0  )
                                                                              ①
Trans. Proc. Name:  SAMPLE        Device: VT62                          Page S-1
Form name:          ADCUST        Length: 24                   10:42 AM  22-Jun-78
................................................................................
                             INITIAL SCREEN PRESENTATION ②

         11111111112222222222333333333344444444445555555555666666666677777777778
12345678901234567890123456789012345678901234567890123456789012345678901234567890

--------------------------------------------------------------------------------
          Customer Master File Subsystem - Add Customer Transaction


     To Add a Customer to the File, Complete all Form Fields and Press ENTER.


Customer Number     ?????? ③ (To be Supplied by System)
Customer Name
Address          ④



Zip Code
Telephone #:        (   )    -
Company Contact
Credit Limit ($)    000000000000 ⑤



  Function Keys: ENTER to Add Customer- CLOSE to quit Add Function ⑥

--------------------------------------------------------------------------------
                                                                ⑦
12345678901234567890123456789012345678901234567890123456789012345678901234567890
         11111111112222222222333333333344444444445555555555666666666677777777778


Keys enabled:     ENTER ABORT AFFRM CLOSE ⑧
```

Figure 4-19 ANNOTATED Screen Listing Page S-1

4.12.11.2 Reply Screen Display #1

1. This listing shows the form structure after Reply #1 has modified it.
2. The text string "★★★TRANSACTION COMPLETE★★★" has been written into the BLANK DISPLAY field REPLY.TEXT.A.
3. The symbol "++ – –++" represents the six characters of data that are extracted from the response message that activated the reply. This is the customer number assigned and returned to the form by the TST.
4. The function key prompt text has been overwritten by new text.
5. These function keys are enabled at the time reply 1 modifies the form.

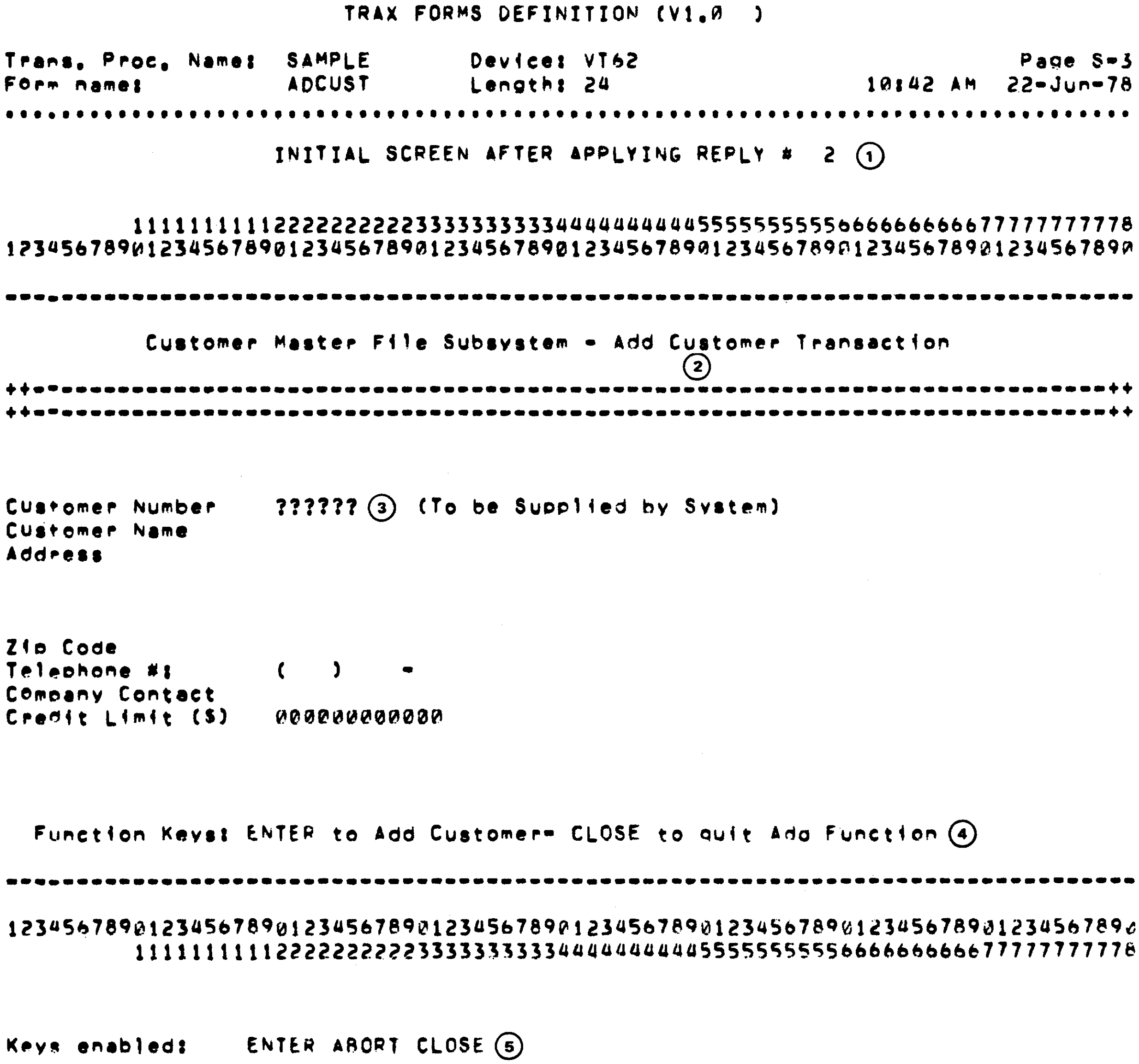


```
                         TRAX FORMS DEFINITION (V1.0  )

Trans. Proc. Name:  SAMPLE       Device: VT62                          Page S-2
Form name:          ADCUST       Length: 24                  10:42 AM  22-Jun-78
................................................................................
                   INITIAL SCREEN AFTER APPLYING REPLY #  1 ①

         1111111111222222222233333333334444444444555555555566666666667777777778
12345678901234567890123456789012345678901234567890123456789012345678901234567890

--------------------------------------------------------------------------------
          Customer Master File Subsystem - Add Customer Transaction

 *** TRANSACTION COMPLETE *** ②


Customer Number     ++--++ ③
Customer Name
Address



Zip Code
Telephone #:        (   )    -
Company Contact
Credit Limit ($)    000000000000



  Function Keys: Press AFFIRM to Add Another Customer - Press CLOSE to quit ④
--------------------------------------------------------------------------------

12345678901234567890123456789012345678901234567890123456789012345678901234567890
         1111111111222222222233333333334444444444555555555566666666667777777778


Keys enabled:     ABORT AFFRM CLOSE ⑤
```

Figure 4-20 ANNOTATED Screen Listing Page S-2

4.12.11.3 Reply Screen Display #2

1. This listing shows the form structure after Reply #2 has modified it.
2. The two error text fields, REPLY.TEXT.A and REPLY.TEXT.B have been filled with text from the response message which activated the reply.
3. These question marks remain from the initial display of the form – reply #2 does not alter this field.
4. The function key text is similarly unchanged.
5. These function keys are enabled after reply #2 has been displayed on the screen.



```
                          TRAX FORMS DEFINITION (V1.0  )

Trans. Proc. Name:  SAMPLE        Device: VT62                         Page S-3
Form name:          ADCUST        Length: 24                 10:42 AM  22-Jun-78
................................................................................
                     INITIAL SCREEN AFTER APPLYING REPLY #  2 ①

         111111111122222222223333333333444444444455555555556666666666777777777778
12345678901234567890123456789012345678901234567890123456789012345678901234567890

--------------------------------------------------------------------------------

          Customer Master File Subsystem - Add Customer Transaction
                                                ②
++----------------------------------------------------------------------------++
++----------------------------------------------------------------------------++


Customer Number     ?????? ③ (To be Supplied by System)
Customer Name
Address



Zip Code
Telephone #:        (   )    -
Company Contact
Credit Limit ($)    000000000000



  Function Keys: ENTER to Add Customer- CLOSE to quit Add Function ④

--------------------------------------------------------------------------------

12345678901234567890123456789012345678901234567890123456789012345678901234567890
         111111111122222222223333333333444444444455555555556666666666777777777778


Keys enabled:    ENTER ABORT CLOSE ⑤
```

Figure 4-21 ANNOTATED Screen Listing Page S-3

CHAPTER 5

DETAILED ATL STATEMENT DESCRIPTIONS

The ATL statements are described in alphabetical order. All valid clauses and parameters are presented for each statement.

DEFAULT

5.1 THE DEFAULT STATEMENT

STATEMENT: DEFAULT

USE: You use the DEFAULT statement to specify default settings for field attributes, declare clear characters, and to enable or disable function keys.

USAGE NOTES: Once you specify an attribute or clear character, or enable/disable a function key in a DEFAULT statement, that specification remains in effect for the rest of the form definition, or until another DEFAULT statement explicitly alters the earlier declaration.

ATL syntax requires that the DEFAULT statement be followed by at least one clause.

FORMAT:

```
DEFAULT                                                                  ┐
┌                                                 ┌                  ┐   │
│               ⎛ ⎧ ANY          ⎫ ⎞              │                  │   │
│               ⎜ ⎪ ALPHANUMERIC ⎪ ⎟              │                  │   │
│ ATTRIBUTES =  ⎜ ⎨ LETTERS      ⎬ ⎟              │  ⎧          ⎫    │   │
│               ⎜ ⎪ NUMERIC      ⎪ ⎟              │  ⎪   ═══    ⎪    │   │
│               ⎜ ⎩ SIGNED       ⎭ ⎟              │  ⎪          ⎪    │   │
│               ⎜                  ⎟              │  ⎪          ⎪    │   │
│               ⎜ { LEFT  }        ⎟              │  ⎪   ═══    ⎪    │   │
│               ⎜ { RIGHT }        ⎟              │  ⎪          ⎪    │   │
│               ⎜                  ⎟              │  ⎪          ⎪    │   │
│               ⎜ { NOTAB }        ⎟              │  ⎪          ⎪    │   │
│               ⎜ { TAB   }        ⎟              │  ⎪          ⎪    │   │
│               ⎜                  ⎟              │  ⎪          ⎪    │   │
│               ⎜ { NOFULL }       ⎟            , │  ⎨   • • •  ⎬    │   │
│               ⎜ { FULL   }       ⎟              │  ⎪          ⎪    │   │
│               ⎜                  ⎟              │  ⎪          ⎪    │   │
│               ⎜ { NOREQUIRED }   ⎟              │  ⎪   ═══    ⎪    │   │
│               ⎜ { REQUIRED   }   ⎟              │  ⎪          ⎪    │   │
│               ⎜                  ⎟              │  ⎪          ⎪    │   │
│               ⎜ { MODIFY   }     ⎟              │  ⎪   ═══    ⎪    │   │
│               ⎜ { NOMODIFY }     ⎟              │  ⎩          ⎭    │   │
│               ⎜                  ⎟              │                  │   │
│               ⎜ { NORMAL  }      ⎟              │                  │   │
│               ⎜ { REVERSE }      ⎟              │                  │   │
│               ⎜                  ⎟              │                  │   │
│               ⎜ { BLANK   }      ⎟              │                  │   │
└               ⎝ { NOBLANK }      ⎠              └                  ┘   │
[ ENABLE = keyname ]                                                     ┘

[ DISABLE = keyname ]

[ CLEAR = "character"]
```

Clauses and Parameters:

Clause	Parameter	Description
ATTRIBUTES=		The ATTRIBUTES clause allows you to specify the way that Input fields may be completed by the terminal user, or the way that Input, Prompt, and Display fields are displayed on the terminal screen. You can specify more than one attribute in the same ATTRIBUTES clause.
	ANY	An Input field can contain any displayable ASCII character in the range OCTAL 040 to 176.
	LETTERS	An Input field can contain only the letters A through Z, a through z, and space.
	NUMERIC	An Input field can contain only the numbers 0 through 9. Spaces are not permitted.
	ALPHANUMERIC	An Input field can contain any character that is a number, a letter, or a space.
	SIGNED	An Input field can contain signed numeric data: The numbers 0 through 9, the sign characters + and —, field punctuation characters (, and .). Spaces are not permitted. An error is displayed on the terminal's error line if the terminal user attempts to enter a character that is not allowed by the character representation attribute specifield for that Input field.
	LEFT	Each character typed into an Input field appears one position to the right of the preceding character.
	RIGHT	Each character typed into an Input field appears in the right-most position of the field. All preceding characters are moved one position to the left. This attribute is very useful for entering numeric data onto a form.
	NOTAB	To advance to another Input field, the terminal user must press the NEXT FIELD or FORWD FIELD key after typing data into an Input field.
	TAB	After the user has completely filled the current field, the cursor moves automatically to the next Input field

Clause	Parameter	Description
	FULL	The user must type enough characters to completely fill the Input field. If the user presses the NEXT FIELD or FORWD FIELD key before completely filling a field specified with this attribute, the VT62 error message is displayed on the terminal error line. The user may skip past a field specified with the FULL attribute by pressing the FORWARD FIELD key.
	NOFULL	Input fields needs not be filled completely.
	REQUIRED	The user must enter data into this field before going on to the next field.
	NOREQUIRED	The user may skip past this field without entering data.
	MODIFY	The terminal user may change the contents of an Input field specified with this attribute.
	NOMODIFY	The terminal user cannot change the contents or enter data in an Input field specified with this attribute. Furthermore, the user cannot position the cursor in this field.
	NORMAL	The Input, Prompt or Display field appears as white characters on a dark background.
	REVERSE	The Input, Prompt, or Display field appears as dark characters on a white background. Spaces appear as white squares. You can assist the terminal user by defining Input fields with the REVERSE attribute. The field is then clearly defined on the screen display.
	BLANK	This attribute can be specified for Display fields only. A Display field with this attribute is not displayed on the initial screen representation of the form. The field appears only on the reply screen displays defined for a form. REPLYs that do not specify text for a Display field with the BLANK attribute cause that field to be blanked.
	NOBLANK	A Display field defined as NOBLANK is written by the terminal as part of the initial form. It is not blanked for a REPLY but a REPLY may overwrite it.

Clause	Parameter	Description
ENABLE=		Allows you to enable user and system function keys. The ATL utility enables the ENTER and ABORT keys by default. To enable other function keys, you must specify the key name in an ENABLE clause. If you place the ENABLE clause in a DEFAULT statement that precedes the FORM statement, the key you specify remains enabled for the initial screen display and all reply screen displays associated with that form, unless you specifically override the default specification in a FORM or REPLY statement. You can specify only one key name in an ENABLE clause. To enable several function keys, you must use several clauses.
	keyname	You can specify the following function key names as valid parameters in an ENABLE clause. ENTER CLOSE AFFIRM STOPREPEAT KEY0 KEYDOT KEY1 KEY2 KEY3
DISABLE=		Allows you to disable user and system function keys. The ATL utility enables the ENTER and ABORT keys by default. You cannot disable the ABORT key. You may disable the ENTER key. To disable other function keys, you must specify the key name in a DISABLE clause. If you place the DISABLE clause in a DEFAULT statement that precedes the FORM statement, the key you specify remains disabled for the initial screen display and all reply screen displays associated with that form, unless you override the disabling in a FORM or REPLY statement.

Clause	Parameter	Description
		You can specify only one key name in a DISABLE clause. To disable several function keys, you must use several clauses.
	keyname	You can specify the following function key names as valid parameters in a disable clause. ENTER CLOSE AFFIRM STOPREPEAT KEY0 KEYDOT KEY1 KEY2 KEY3
CLEAR=		Allows you to specify the character that is displayed in empty Input field character positions. Any Input field character position that is not filled in by the terminal (using the VALUE clause, or a REPLY statement WRITE clause) or completed by the user, is filled using the character specified in this clause.
	"character"	You can specify any displayable ASCII character as the parameter to the CLEAR clause. The null character (ASCII 000) is assumed as the default. The character you specify as the parameter must be enclosed in single or double quotation marks.

EXAMPLES:

The following example shows a default statement that requires all input fields to be completed, enables the CLOSE and AFFIRM system function keys, and specifies a period (.) as the default clear character.

```
DEFAULT                              !To SET Defaults for whole form
     ATTRIBUTES = REQUIRED           !All fields must have data
     ENABLE = CLOSE                  !Enable system function keys
     ENABLE = AFFIRM
     CLEAR = " . "                   !Set period as default clear character
```

5.2 THE DISPLAY STATEMENT

STATEMENT: DISPLAY

USE: You use the DISPLAY statement to define Display fields.

Display fields are written on the screen by the terminal as part of the initial form display or as part of replies. They provide a means for communicating with the terminal user, giving instructions and error information.

USAGE NOTES: Display fields cannot be returned in the exchange message when a user key is pressed.

Display fields cannot be accessed by the terminal user. That is, the cursor cannot be positioned in a Display field.

Display fields possessing the BLANK attribute (see below) are very useful for displaying text received as part of a REPLY.

ATL syntax requires that the DISPLAY statement be followed by at least one clause.

You must specify the row and column parameters as part of the DISPLAY statement.

The length of a Display field may be defined explicitly in a LENGTH clause, or implicitly in a VALUE clause. If both clauses are present, the value ATL assigns to the field length is the greater of the two length definitions.

FORMAT:
DISPLAY = row, column

```
┌                                                                                    ┐
│VALUE =  ⎧ "string"                      ⎫ ┌ ⎧ "string"                      ⎫     ┐│
│         ⎪ FILL ("character", count)     ⎪ │ ⎪ FILL ("character", count)     ⎪     ││
│         ⎪ DATE                          ⎪ │ ⎪ DATE                          ⎪     ││
│         ⎨ TIME                          ⎬ │,⎨ TIME                          ⎬ ••• ││
│         ⎪ TRANSACTION                   ⎪ │ ⎪ TRANSACTION                   ⎪     ││
│         ⎪ NAME                          ⎪ │ ⎪ NAME                          ⎪     ││
│         ⎪ STATION                       ⎪ │ ⎪ STATION                       ⎪     ││
└         ⎩ REQUEST (position, count)     ⎭ └ ⎩ REQUEST (position, count)     ⎭     ┘┘

[ LENGTH = count ]

┌ ATTRIBUTES =   ⎧ ⎧ NORMAL  ⎫ ⎫  ┌   ⎧ = ⎫ ┐                                       ┐
│                ⎪ ⎩ REVERSE ⎭ ⎪  │ , ⎨   ⎬ │                                       │
│                ⎨             ⎬  │   ⎩ = ⎭ │                                       │
│                ⎪ ⎧ BLANK   ⎫ ⎪  │         │                                       │
└                ⎩ ⎩ NOBLANK ⎭ ⎭  └         ┘                                       ┘

[ LABEL = label-name ]
```

Clauses and Parameters:

Clause	Parameter	Description
	row	Specifies the screen row where the first character of the Display field is located. The row number is always calculated relative to the first line of the form. The value you specify may not be greater than the value specified in the SPLIT clause of the FORM statement.
	,column	Specifies the character position on the row where the field begins. You may not specify a character position that overlaps another Display or Menu field. Row and column parameters can also be specified using the ATL “dot” constructs.
VALUE=		You use the VALUE clause to specify the initial contents of a Display field that is defined with the NOBLANK attribute. In a DISPLAY statement that specifies a field as having the BLANK attribute, you may use the VALUE clause to specify the data displayed in that field as part of a REPLY. In this case, 1. The DISPLAY field label must be specified in a REPLY statement WRITE clause. 2. The REPLY statement WRITE clause that references the Display field cannot contain other parameters. 3. The DISPLAY statement VALUE clause cannot contain the REQUEST function in its parameter list. You can specify more than one value for a VALUE clause by supplying several parameters. You can also specify a valid parameter more than once as part of a VALUE clause.
	“string”	A string of characters enclosed by quotation marks. The enclosed string is displayed on the screen exactly as typed, without the quotation marks.
	FILL (“character”,count)	The character you specify in the first parameter is written into the Display field the number of times specified by the second parameter.

Clause Parameter Description

DATE

The current date is written into the Display field in the format: DD-MMM-YY

TIME

The current time of day is written into the Display field in the format: HH:MM:SS

TRANSACTION

The system-determined transaction instance number is written into the Display field as ten ASCII characters. If you specify this parameter on a form used as the first form of an exchange, the field is filled with ten characters. (A transaction instance number is assigned after the exchange message from the first form is received at the terminal station.)

NAME

The 6-character name of the transaction that is currently being run from the application terminal is written into the Display field.

STATION

The 6-character terminal station name is written into the Display field.

REQUEST (position,count)

The REQUEST function allows you to obtain information contained in the response message that requested this form.

When you specify a Display field with a VALUE clause containing the REQUEST function, the field is filled with text from the requesting response message, beginning at the character position specified by the first parameter, and continuing for the number of characters specified by the second parameter.

You may also use the dot constructs to specify the character position parameter. A . by itself refers to the next available position in the requesting response message. The values .+n and .–n specify right (+) and left (–) offsets from the current position in the response message

The first character position in the requesting response message can be referenced by specifying 1 as the first parameter of the REQUEST function.

LENGTH=

Use this clause to explicitly define the length of a Display field.

Clause	Parameter	Description
	count	A numeric value from 1 to 1920. If the count parameter causes the field to overlap another field on the form a syntax error is flagged by the ATL utility. If a field extends past the physical right margin, that field is "wrapped around" to continue in the first column of the next line.
ATTRIBUTES =		The ATTRIBUTES clause allows you to specify when and how Display fields appear on the terminal screen.
		You can specify more than one attribute in the same ATTRIBUTES clause.
	NORMAL	The Display field appears as white characters on a dark background.
	REVERSE	The Display field appears as dark characters on a white background. Spaces appear as white squares.
	BLANK	A Display field with this attribute is not displayed on the initial screen representation of the form. The field may appear only on the reply screen displays defined for a form. REPLYs that do not specify text for a Display field with the BLANK attribute cause that field to be blanked.
	NOBLANK	A Display field defined as NOBLANK is written by the terminal as part of the initial form. It is not blanked for a REPLY but a REPLY may overwrite it.
LABEL =		Use the LABEL clause to specify a label for a Display field. Any field that you reference in another ATL statement (in this case, REPLY) must have a label.
	label-name	A label can be from 1 to 30 characters long. The first character must be a letter and the remaining characters are limited to: A through Z, a through z, 0 through 9, period (.), hyphen (-), underline (_).

Clause Parameter Description

In addition, you can use the percent symbol (%) or the dollar sign ($) as the last character in a label.

The label character set allows you to use the same data item names in both forms and TSTs. However, if the label-name in the LABEL clause is the same as (or part of) an ATL statement, clause or parameter keyword, the ATL utility issues a Fatal error message.

You can avoid this possibility by always including a special character (.), (-), (_), (%), or ($) in a label name. These characters are never used in an ATL reserved keyword. See Table 2-1 for the list of ATL reserved words.

EXAMPLES:

Display statements can be used to display data on the initial form and all replies, or on replies only. In the examples shown here, the first display statement appears on all versions of the form:

```
DISPLAY = 2,11                          !Indent 1 line and 11 spaces
          VALUE = "Customer Master File Subsystem - ",
                  "Add Customer Transaction"
          ATTRIBUTES = REVERSE,         !Highlight in reverse video
                       NOBLANK          !Don't Erase during Replies
```

The second display statement simply defines an 80-character error line, for use during replies. Since the ATTRIBUTES clause is missing, this display field has the BLANK attribute assumed.

```
DISPLAY = 4,1                           !Skip 1 Line
          LABEL = REPLY.TEXT.A          !Label for 1st REPLY Line
          LENGTH = 80                   !Line covers Full Screen Width
```

END

5.3 THE END STATEMENT

STATEMENT:	END
USE:	You use the END statement to define the end of the form definition.
USAGE NOTES:	This statement must be specified as the last item in an ATL source statement file. The ATL utility does not process any text beyond this statement.

FORMAT:

END

5.4 THE FORM STATEMENT

STATEMENT: FORM

USE: You use the FORM statement to specify dimensions and attributes required as part of the form definition. You can use the FORM statement to:

1. Divide the screen into Display and Form areas.
2. Specify the height and width of the form for hard copy devices.
3. Change default function key text values.
4. Cause the terminal bell to sound when displaying an initial screen.
5. Specify that the form is used to select a transaction.

USAGE NOTES: ATL syntax requires that the FORM statement be followed by at least one clause.

The ATL utility assumes the following default conditions exist:

1. The form being defined contains a Form area only, i.e. there is no Display area.
2. All function keys, except the ABORT and ENTER keys, are disabled.
3. This form is not being used to select the next transaction.
4. Default text is associated with the user function keys.
5. The Bell will not ring when an initial form display occurs.

You can override these assumptions by using the FORM statement and its corresponding clauses.

FORMAT:
FORM

```
/ [ SPLIT = length ]                                                          \
|                                                                             |
| [ ENABLE = keyname ]                                                        |
|                                                                             |
| [ DISABLE = keyname ]                                                       |
|                                                                             |
| [ KEYCAP = keyname, "text-string" ]                                         |
|                                                                             |
< [ SELECT =  { MENU              }      ,reply-1     { ,reply-2       } ]    >
| [           { input-field-label }                   { ,NOAUTHORIZE   } ]    |
|                                                                             |
| [ LENGTH =  { line-count }  ]                                               |
| [           { FEED       }  ]                                               |
|                                                                             |
| [ BELL   [= periods] ]                                                      |
|                                                                             |
\ [ WIDTH = form-width ]                                                      /
```

Clauses and Parameters:

Clause	Parameter	Description
SPLIT =		The SPLIT clause permits you to separate the form into a Display area and a Form area. This clause is valid only for forms defined for use on interactive terminals. The Display area is always the top part of the form. The Form area is the remainder of the screen.
	line-count	The line-count specifies the length of the Display area of the form. Acceptable values for line-count are the integers from 0 (the default) to the maximum length of the form. A line-count value of 0 creates a form with no Display area. A line-count value equal to the maximum form length creates a Form with no Form area. For the VT62 DECSCOPE, the maximum length of the form is 23.
ENABLE =		Allows you to enable user and system function keys at the time of the initial screen display. The ATL utility enables the ENTER and ABORT keys by default. To enable other function keys, you must specify the key name in an ENABLE clause. If you place the ENABLE clause in a DEFAULT statement before the FORM statement, the key you specify remains enabled for the initial screen display and all reply screen displays associated with that form, unless specifically overridden in a DISABLE clause as part of a FORM or REPLY statement. Enabling a key in FORM statement affects only the initial display of the form. You can specify only one key name in an ENABLE clause. To enable several function keys, you must use several clauses.
	keyname	You can specify the following function key names as valid parameters in an ENABLE clause. ENTER CLOSE AFFIRM

Clause Parameter Description

STOPREPEAT
KEY0
KEYDOT
KEY1
KEY2
KEY3

DISABLE =

Allows you to disable user and system function keys at the time of the initial screen display.

The ATL utility enables the ENTER and ABORT keys by default. You cannot disable the ABORT key. You may disable the ENTER key.

To disable other function keys you must specify the key name in a disable clause.

If you place the DISABLE clause in a DEFAULT statement that precedes the FORM statement, the key you specify remains disabled for the initial screen display and all reply screen displays associated with that form, unless specifically overridden by a DISABLE clause in a FORM or REPLY statement. If you disable a key in the FORM statement, that key is affected only for the initial screen display.

You can specify only one key name in a disable clause. To disable several function keys, you must use several clauses.

keyname

You can specify the following function key names as valid parameters in a disable clause.

ENTER
CLOSE
AFFIRM
STOPREPEAT
KEY0
KEYDOT
KEY1
KEY2
KEY3

KEYCAP =

You use the KEYCAP clause to define the text string associated with a specified user function key. This clause may be specified only when the form is defined

Clause — Parameter — Description

for use on interactive terminals. When you press a user function key, the text defined for that key is entered into the exchange message at the position specified by the VALUE=KEY clause in the MESSAGE statement. (See section 3.1.1.5 for a discussion of this feature.) If the KEYCAP clause is not present in the FORM statement, the ATL utility assumes the following default key texts.

KEY	DEFAULT KEY TEXT
ENTER	"ENTER "
KEY0	"KEY00 "
KEY1	"KEY01 "
KEY2	"KEY02 "
KEY3	"KEY03 "
KEYDOT	"KYDOT "

keyname — The name of the user function key whose text is changed by this KEYCAP clause.

,"text" — The text string associated with the user function key specified in the keyname parameter.

NOTE

> The total number of characters associated with the six user function keys is limited to 214. To determine if you have reached this limit, add the lengths of all of the text strings specified in the KEYCAP clause, then add six characters for each key retaining its default value.

SELECT = — You use the SELECT clause to tell the ATL utility that this form definition is used to select a transaction. You can specify this clause only on forms defined for use on interactive terminals.

MENU — To select a transaction from a list of transaction names presented as Menu fields on the terminal screen (and called a menu), you must specify the keyword MENU as the first parameter in the SELECT clause.

input-field-label — If you want to require the user to type the transaction identifier, then the first parameter in the SELECT

Clause	Parameter	Description
		clause must be the label of the Input field where the user types the transaction name.
	reply1,	You must specify the number of the reply screen displayed if the requested transaction does not exist.
	reply2	You must specify the number of the reply screen displayed when the terminal finds that the user is not authorized for this transaction type.
	NOAUTHORIZE	You can skip the authorization check by specifying NOAUTHORIZE as the third parameter.
LENGTH =		The LENGTH clause specifies a form's line-count for a hard-copy device. You can specify this clause only for forms used at output-only terminals.
	line-count	Specifies how long the form is. For example, for an LA180 using 11-inch paper (6 lines = 1 inch) specify LENGTH=66. If a 46-line form is displayed on that paper, the LENGTH clause causes 20 line feeds to be appended to the form, moving the paper to the top of the next form.
	FEED	If the output device you are using has a hardware form feed, you can specify this parameter. The terminal executes a hardware form feed when the end of a form display is reached.
		If the LENGTH clause is omitted, neither form feed nor line feed characters are appended to the form.
BELL		You use the BELL clause to specify that the terminal bell is to sound at the time the initial screen is displayed.
	[=periods]	The periods parameter is optional. If you do not specify a period parameter, the default is one period. Specifying a number of periods determines how long the bell is to ring (155 milliseconds/period for the VT62). For example, specifying BELL=7 causes the bell on a VT62 terminal to sound for approximately one second when an initial form is displayed.
WIDTH =		The WIDTH clause specifies the number of characters that can be printed across the hard-copy device print

Clause	Parameter	Description
		area. You can specify this clause only for an output-only form.
	character-count	May be any value less than or equal to the physical width of the device print line. If the specified width exceeds the physical width of the device a fatal error results. If you specify that a Print field is to extend beyond the right margin, specified by the WIDTH clause, a fatal error results. If you specify that a print field is to extend beyond the hardware-defined right margin, a warning results and the excess field characters are "folded" over to the next print line, beginning in column 1.

EXAMPLES:
The form statement has a number of different uses. In the simplest case, when you define a form for use on the VT62, you can simply specify the size of the Display area (using the ATL SPLIT clause), and how many times the bell will ring. For example:

```
FORM                                       !Form Statement
     SPLIT = 8                             !8 Line DISPLAY at Top of Form
     BELL = 2                              !Ring Bell Twice for Initial Form
```

When you specify the SELECT clause in a form statement, you are defining a transaction selection form. Depending on the first parameter of the SELECT clause, you specify that the form is a menu-type selection form, or an input-type selection form. In the first form statement shown here, the field label TRANS-NAME has been specified in the SELECT clause. The data entered into this field is used to select and display the next transaction type.

```
FORM
     SPLIT = 20                            !20 lines Display area
     SELECT = TRANS-NAME,1,2               !Input field label and reply numbers
```

If the keyword MENU is specified as the first parameter in the SELECT clause, then the contents of the menu field selected by the user will govern the next transaction displayed on the terminal. As before, the reply number for an invalid transaction is specified. In this example, the user authorization check is skipped.

```
FORM
     SPLIT = 23                            !Form has only Display area
     SELECT = MENU,1,NOAUTHORIZE           !Menu selection specified
```

A FORM statement used to define a Report form used at an output only station allows you to specify the dimensions of the form. In the following example, note the use of the LENGTH and WIDTH clauses.

```
FORM
    LENGTH = 22                             !Form has 22 lines (3 forms per page)
    WIDTH = 132                             !132-character form width
    BELL = 3                                !Bell rings for ½ sec to alert user
```

INPUT

5.5 THE INPUT STATEMENT

STATEMENT: INPUT

USE:

You use the INPUT statement to define Input fields in the Form area. Input fields are transmitted to the terminal station when a user function key is pressed. The user can move the cursor to an Input field. Input fields are the only fields on a form where the user can enter or change data. The terminal may initialize Input fields and a REPLY may overwrite them.

USAGE NOTES:

The total number of Input fields may not exceed 127. Furthermore, the combined total of Input and MENU fields in a single form definition may not exceed 128.

You must specify the row and column parameters as part of the INPUT statement.

You must also specify at least one clause following the INPUT statement.

To simplify the coding of forms, you can specify row and column values as offsets from the previous field through the use of the ATL "dot" constructs.

The length of an INPUT field may be defined either explicitly in a LENGTH clause, or implicitly by a VALUE clause. If both clauses are present, ATL assigns the greater of the two lengths to the field.

FORMAT:

```
INPUT = row, column
  ⎧ ⎡                 ⎛ ⎧ ANY          ⎫ ⎞ ⎡   ⎧ ___ ⎫ ⎤ ⎤ ⎫
  ⎪ ⎢                 ⎜ ⎪ ALPHANUMERIC ⎪ ⎟ ⎢   ⎪ ___ ⎪ ⎥ ⎥ ⎪
  ⎪ ⎢ ATTRIBUTES =    ⎜ ⎨ LETTERS      ⎬ ⎟ ⎢   ⎪ ___ ⎪ ⎥ ⎥ ⎪
  ⎪ ⎢                 ⎜ ⎪ NUMERIC      ⎪ ⎟ ⎢   ⎪ ___ ⎪ ⎥ ⎥ ⎪
  ⎪ ⎢                 ⎜ ⎩ SIGNED       ⎭ ⎟ ⎢   ⎪ ___ ⎪ ⎥ ⎥ ⎪
  ⎪ ⎢                 ⎜ { LEFT  }        ⎟ ⎢ , ⎨ ••• ⎬ ⎥ ⎥ ⎪
  ⎪ ⎢                 ⎜ { RIGHT }        ⎟ ⎢   ⎪ ___ ⎪ ⎥ ⎥ ⎪
  ⎪ ⎢                 ⎜ { NOTAB }        ⎟ ⎢   ⎪ ___ ⎪ ⎥ ⎥ ⎪
  ⎪ ⎢                 ⎨ { TAB   }        ⎬ ⎢   ⎪ ___ ⎪ ⎥ ⎥ ⎪
  ⎪ ⎢                 ⎜ { NOFULL }       ⎟ ⎢   ⎪ ___ ⎪ ⎥ ⎥ ⎪
  ⎪ ⎢                 ⎜ { FULL   }       ⎟ ⎢   ⎩     ⎭ ⎥ ⎥ ⎪
  ⎨ ⎢                 ⎜ { NOREQUIRED }   ⎟ ⎢           ⎥ ⎥ ⎬
  ⎪ ⎢                 ⎜ { REQUIRED   }   ⎟ ⎢           ⎥ ⎥ ⎪
  ⎪ ⎢                 ⎜ { MODIFY   }     ⎟ ⎢           ⎥ ⎥ ⎪
  ⎪ ⎢                 ⎜ { NOMODIFY }     ⎟ ⎢           ⎥ ⎥ ⎪
  ⎪ ⎢                 ⎜ { NORMAL  }      ⎟ ⎢           ⎥ ⎥ ⎪
  ⎪ ⎢                 ⎜ { REVERSE }      ⎟ ⎢           ⎥ ⎥ ⎪
  ⎪ ⎣                 ⎝ { NOECHO }       ⎠ ⎣           ⎦ ⎦ ⎪
  ⎪ ⎡ VALUE =  ⎧ "string"                   ⎫ ⎡   ⎧ "string"                   ⎫      ⎤ ⎤ ⎪
  ⎪ ⎢          ⎪ FILL ("character", count)  ⎪ ⎢   ⎪ FILL ("character", count)  ⎪      ⎥ ⎥ ⎪
  ⎪ ⎢          ⎪ DATE                       ⎪ ⎢   ⎪ DATE                       ⎪      ⎥ ⎥ ⎪
  ⎪ ⎢          ⎨ TIME                       ⎬ ⎢ , ⎨ TIME                       ⎬ •••• ⎥ ⎥ ⎪
  ⎪ ⎢          ⎪ TRANSACTION                ⎪ ⎢   ⎪ TRANSACTION                ⎪      ⎥ ⎥ ⎪
  ⎪ ⎢          ⎪ NAME                       ⎪ ⎢   ⎪ NAME                       ⎪      ⎥ ⎥ ⎪
  ⎪ ⎢          ⎪ STATION                    ⎪ ⎢   ⎪ STATION                    ⎪      ⎥ ⎥ ⎪
  ⎪ ⎣          ⎩ REQUEST (position, count)  ⎭ ⎣   ⎩ REQUEST (position, count)  ⎭      ⎦ ⎦ ⎪
  ⎪  [LENGTH = count]                                                                      ⎪
  ⎪  [CLEAR = "character"]                                                                 ⎪
  ⎩  [LABEL = label-name]                                                                  ⎭
```

Clauses and Parameters:

Clause	Parameter	Description
	row	Specifies the screen row where the first character of the Input field is located. The row number is always calculated relative to the first line of the form area. The value you specify may not be greater than the difference between the value in the SPLIT clause of the FORM statement and the number of available lines, 23 in the case of the VT62.
	,column	Specifies the character position in the row where the field begins. You may not specify a character position that overlaps another Input or Prompt field.
VALUE =		You use the VALUE clause to specify the initial contents of an Input field. You can specify more than one value for a VALUE clause by supplying several parameters. You can also specify a valid parameter more than once as part of a VALUE clause.
	"string"	A string of characters enclosed by quotation marks. The enclosed string is displayed on the screen exactly as typed, without the quotation marks.
	FILL ("character",count)	The character you specify in the first parameter is written into the Input field the number of times specified by the second parameter.
	DATE	The current date is written into the Input field in the format: DD-MMM-YY
	TIME	The current time of day is written into the Input field in the format: HH:MM:SS
	TRANSACTION	The system-determined transaction instance number is written into the Input field as ten ASCII characters.
	NAME	The 6-character name of the transaction that is currently being run from the application terminal is written into the Input field.
	STATION	The 6-character terminal station name is written into the Input field.

Clause — Parameter — Description

REQUEST (position,count)

The REQUEST function allows you to obtain information contained in the response message that requested the current form.

When you specify an Input field with a VALUE clause containing the REQUEST function, the field is filled with text from the requesting response message. This text begins at the character position specified by the first parameter, and continues for the number of characters specified by the second parameter.

You also may use the dot constructs to specify the character position parameter. A . by itself refers to the next available position in the requesting response message. The values .+n and .–n specify right (+) and left (–) offsets from the current position in the response message.

The first character position in the requesting response message can be referenced be specifying 1 as the first parameter of the REQUEST function.

Characters are moved from the requesting message to the field in left-to-right order, regardless of the field justification attribute of the receiving field.

ATTRIBUTES =

The ATTRIBUTES clause allows you to specify the way that fields may be completed by the terminal user, or the way that Input fields are displayed on the terminal screen.

You can specify more than one attribute in the same ATTRIBUTES clause.

ANY

An Input field can contain any displayable ASCII character in the range OCTAL 040 to 176.

LETTERS

An Input field can contain only the letters A through Z, a through z, and space.

NUMERIC

An Input field can contain only the numbers 0 through 9. Spaces are not permitted.

ALPHANUMERIC

An Input field can contain any character that is a number, a letter, or a space.

Clause	Parameter	Description
	SIGNED	An Input field can contain signed numeric data: The numbers 0 through 9, the sign characters + and –, field punctuation characters (, and .). Spaces are not permitted. An error is displayed on the terminal's error line if the terminal user attempts to enter a character that is not allowed by the character representation attribute specified for that Input field.
	LEFT	Each character typed into an Input field appears one position to the right of the preceding character.
	RIGHT	Each character typed into an Input field appears in the right-most position of the field. All preceding characters are moved one position to the left. This attribute is very useful for entering numeric data onto a form.
	NOTAB	To advance to another Input field, the terminal user must press the NEXT FIELD or FORWD FIELD key after typing data into an Input field.
	TAB	After the user has completely filled the current field, the cursor moves automatically to the next Input field.
	FULL	The user must type enough characters to completely fill the Input field. If the user presses the NEXT FIELD or FORWD FIELD key before completely filling a field specified with this attribute, a VT62 error is noted and displayed on the terminal error line. The user may skip past a field specified with the FULL attribute by pressing the FORWARD FIELD key.
	NOFULL	Input fields need not be filled completely.
	REQUIRED	The user must enter data into this field before going on to the next field.
	NOREQUIRED	The user may skip this field without entering data.
	MODIFY	The terminal user may change the contents of an Input field specified with this attribute.
	NOMODIFY	The terminal user cannot change the contents or enter data in an Input field specified with this attribute.

Clause	Parameter	Description
		Furthermore, the user cannot position the cursor in this field.
	NORMAL	The Input field appears as white characters on a dark background.
	REVERSE	The Input field appears as dark characters on a white background. Spaces appear as white squares. You can assist the terminal user by defining Input fields with the REVERSE attribute. The field is then clearly defined on the screen display.
	NOECHO	The terminal suppresses the display of any characters the user types into an Input field. NOECHO can be specified for only one field on a form. The field specified with the NOECHO attribute is limited in length to 40 characters.
LENGTH =		Use this clause to explicitly define the length of an Input field.
	count	A numeric value from 1 to 1920. If the count parameter causes the field to overlap another field on the form, a syntax error is flagged by the ATL utility. If a field extends past the physical right margin, that field is "wrapped around" to continue in the first column of the next line.
CLEAR =		Allows you to specify the character that is displayed in empty Input field character positions.
		Any Input field character position that is not filled in by the system (using the VALUE clause, or a REPLY statement WRITE clause) or completed by the user, is filled using the character specified in this clause.
	"character"	You can specify any displayable ASCII character as the parameter to the CLEAR clause. The null character (ASCII 000) is assumed as the default. The character you specify as the parameter must be enclosed in single or double quotation marks.
LABEL =		Use the LABEL clause to specify a label for an Input field. Any field that you reference in another ATL statement (MESSAGE, REPLY, etc.) must have a label.

Clause Parameter Description

label-name

A label can be from 1 to 30 characters long. The first character must be a letter and the remaining characters are limited to:

A through Z,
a through z,
0 through 9,
period (.),
hyphen (-),
underline (_).

In addition, you can use the percent symbol (%) or the dollar sign ($) as the last character in a label.

The label character set allows you to use the same data item names in both forms and TSTs. However, if the label-name is the same as (or part of) an ATL statement, clause or parameter keyword, the ATL utility issues a Fatal error message.

You can avoid this possibility by always including a special character (.), (-), (_), (%), or ($) in a label name. These characters are never used in an ATL reserved keyword. See Table 2-1 for the list of ATL reserved words.

EXAMPLES:
The INPUT statements define the only user-accessible ATL fields. A wide range of ATTRIBUTES are available for use with the INPUT statements, as well as the LABEL clause to assign names to fields, and the LENGTH clause to explicitly specify field length. In addition, INPUT fields can be given an initial value through the VALUE clause. The following examples give a brief overview of the different ways used to specify INPUT fields.

The first INPUT field example is used to specify a zip code field on a form. The field is specified as a required field that must be full. Hence five characters must always be typed into this field.

```
INPUT = .,45                          !Skip to column 45 of current line
        LABEL = ZIP.CODE              !Label field for use by MESSAGE
        LENGTH = 5                    !Zip Code is 5 digits in U.S.A.
        ATTRIB = REVERSE,             !Highlight in Reverse Video
                 NUMERIC,             !Restrict Character Set
                 FULL,                !Must type all 5 digits
                 REQUIRED             !Field must be completed
```

In the second example of an INPUT statement, the Input field is designed to contain a signed numeric value. The designer wanted data entered in this field for every customer, and the data was to be right-justified, and blank spaces were to be zero-filled to assist the TST when it processed the amount. The resulting INPUT statement to do all this was:

```
INPUT = .,20                                !Column 30 of This Line
        LABEL = CREDIT.LIMIT                !Label for Message
        LENGTH = 12                         !Allows 12 spaces for amount
        ATTRIB = REVERSE,SIGNED,            !Highlight, allow commas and periods
                 RIGHT, REQUIRED            !Right-justify for easier typing
                                            !Require input in this field
        CLEAR = "0"                         !Fill in with left zeros
```

In the last example, the NOECHO attribute is used to protect the contents of the ACCOUNT-NUMBER field from being viewed by others.

```
INPUT = .,.+10                              !Start 10 spaces past prompt
        LENGTH = 9                          !Define field length
        LABEL = ACCOUNT-NUMBER              !Label field name
        ATTRIBUTE = REQUIRED,
                    NOECHO,
                    NUMERIC                 !Protect account number data
```

MENU

5.6 THE MENU STATEMENT

STATEMENT: MENU

USE: The MENU statement is used to describe Menu fields that are found in the Display area of a form. A group of Menu fields forms a Menu. The user can select one field from a Menu, using the SELECT key and transmit that field to the terminal station by pressing a user function key.

USAGE NOTES: The cursor may be moved to a Menu field by using the cursor control keys on the terminal keyboard.

The contents of a Menu field can be selected or deselected by pressing the SELECT/DESELECT key.

Selected Menu fields are displayed in REVERSE video.

The terminal user may select only one Menu field on a form. If more than one Menu field is selected, and the user presses a user function key, the terminal bell will ring, and an error message will be written to the terminal error line.

The MENU fields are initialized as part of the initial form display. The terminal user cannot enter data into a menu field and menu fields are not altered by a reply.

The total number of MENU fields on a single form may not exceed 127. Furthermore, the total number of Input and MENU fields on a form may not exceed 128.

You must specify row and column parameters as part of the MENU statement.

ATL syntax requires that the MENU statement be followed by at least one VALUE clause.

The length of a MENU field is implicitly defined by the VALUE clause.

FORMAT:
MENU = row, column

$$\text{VALUE} = \left\{\begin{array}{l}\text{“string”}\\ \text{FILL (“character”, count)}\\ \text{REQUEST (position, count)}\end{array}\right\}\left[\,,\left\{\begin{array}{l}\text{“string”}\\ \text{FILL (“character”, count)}\\ \text{REQUEST (position, count)}\end{array}\right\},\ldots\right]$$

Clauses and Parameters:

Clause Parameter Description

row

The value you supply for row tells the terminal the screen row where the field you are defining is written. The row number is defined relative to the start of the form.

,column

The value you supply for the column parameter specifies the character position on the row where the field begins.

The following MENU statement specifies a field that appears on the first row of the screen, beginning in column 20.

```
MENU=1,20
```

To simplify the coding of forms, you may specify row and column values as offsets from the previous field through the use of the ATL "dot" constructs.

VALUE =

You use the VALUE clause to specify the initial contents of a Menu field.

You can specify more than one value for a VALUE clause by supplying several parameters. You can also specify a valid parameter more than once as part of a VALUE clause.

"string"

A string of characters enclosed by quotation marks. The enclosed string is displayed on the screen exactly as typed, without the quotation marks.

FILL("character",count)

The character you specify in the first parameter is written into the Menu field the number of times specified by the second parameter.

REQUEST (position,count)

The REQUEST function allows you to obtain information contained in the response message that requested the current form.

When you specify a Menu field with a VALUE clause containing the REQUEST function, the field is filled with text from the requesting response message. The filling operation begins at the character position specified by the first parameter, and continuing for the number of characters specified by the second parameter.

Clause	Parameter	Description
		You may also use the dot constructs to specify the character position parameter. A . by itself refers to the next available position in the requesting response message. The values .+n and .—n specify right (+) and left (—) offsets from the current position in the response message.
		The first character position in the requesting response message can be referenced by specifying 1 as the first parameter of the REQUEST function.

EXAMPLES:

The MENU statement is most commonly specified with a character string value. For example:

```
MENU =  .+1,20                          !Next Line, Starting at column 20
        VALUE = "ADDCUS"                !Name of Add Customer Transaction
```

A MENU statement can also be initialized using the REQUEST function. The following example shows a field initialized by a requesting response message.

```
MENU =  1,20
        VALUE = REQUEST (1,5)           !Fill with 1st five characters
                                        !of response message.
```

5.7 THE MESSAGE STATEMENT

STATEMENT: MESSAGE

USE: To construct an exchange message, you must specify one or more MESSAGE statements in the form definition. The MESSAGE statement defines the structure of the exchange message.

USAGE NOTES: ATL syntax requires that the MESSAGE statement be followed by at least one VALUE clause.

You must specify a position value as part of the MESSAGE statement.

Multiple MESSAGE statements may be issued, and the first character of a later MESSAGE need not be contiguous to the last character of the preceding message. However, if you do not explicitly specify the content of a character position, that position may possibly contain spurious ASCII values.

The total length of the exchange message corresponds to the number of character positions specified by all MESSAGE statements in the form definition. If a longer length is desired, an arbitrary character can be placed in the message.

FORMAT:
MESSAGE = position

```
VALUE =  { "string"                    }   [ { "string"                    }      ]
         { FILL ("character", count)   }   [ { FILL ("character", count)   }      ]
         { DATE                        }   [ { DATE                        }      ]
         { TIME                        }   [ { TIME                        }      ]
         { TRANSACTION                 }   [ { TRANSACTION                 }      ]
         { NAME                        } , [ { NAME                        } ,... ]
         { STATION                     }   [ { STATION                     }      ]
         { REQUEST (position, count)   }   [ { REQUEST (position, count)   }      ]
         { MENU                        }   [ { MENU                        }      ]
         { label-name                  }   [ { label-name                  }      ]
         { KEY                         }   [ { KEY                         }      ]
```

Clauses and Parameters:

Clause	Parameter	Description
	position	The value you specify for the position parameter refers to the character position in the exchange message where the terminal station is directed to begin placing the data specified in the VALUE clauses.
VALUE =		The VALUE clause allows you to specify the contents of the exchange message. You may use more than one parameter in a value clause, and you may use the same parameter more than once in the same VALUE clause.
	"string"	The string you specify is placed in the exchange message.
	FILL ("character",count)	The character specified by the first parameter is placed in the exchange message the number of times specified in the second parameter.
	DATE	The current date is placed in the exchange message in the format: DD-MMM-YY.
	TIME	The current clock time is placed in the exchange message in the format: HH:MM:SS.
	TRANSACTION	The system-determined transaction instance number is placed in the exchange message as ten ASCII characters. The transaction instance number can be retrieved through the use of the ATL VALUE clause and TRANSACTION parameter in the exchange message. This is the only way the transaction instance number can be retrieved by the TST.
	NAME	The 6-character name of the current transaction invoked at the application terminal is placed in the exchange message.
	STATION	The 6-character terminal station ID is placed in the exchange message.
	MENU	The contents of the selected MENU item are placed in the exchange message. The length assumed for this parameter is equal to the length of the longest defined menu field. The null character (OCTAL 000) is used

Clause	Parameter	Description
		to pad any unfilled character positions in the exchange message.
	label-name	The contents of the Input field referenced by that label parameter are placed in the exchange message.
	KEY	When a user function key is pressed, the text string associated with that user function key is placed in the exchange message. (See Section 3.1.1.5, "Exchange messages" for a discussion of this feature.) The length of the field to be defined is the length of the longest defined key text.

EXAMPLES:

The message statement specifies the format used to construct the exchange message. In the first example, the MESSAGE statement starts in position 7. The first six locations are filled with ASCII 000. This is done to allow alignment with the fields in the customer record.

```
MESSAGE = 7                                   !Start filling in 7th character position
                                              !To reserve space for Customer _#.
VALUE =                                       !Value clause defines filling order
          CUSTOMER.NAME,
          ADDRESS.1,
          ADDRESS.2,
          ADDRESS.3,
          ZIP.CODE,
          AREA.CODE, TEL.EXCHANGE, TEL.EXTENSION,
          ATTENTION,
          CREDIT.LIMIT
```

A second example of the MESSAGE statement shows how the VALUE = KEY clause is used to transmit the function key text string to the system.

```
MESSAGE = 1
          VALUE = KEY                         !Place Key text string in Exch. Msg.
```

The MESSAGE statement can also specify that the contents of a selected MENU field are placed in the exchange message:

```
MESSAGE = 1
          VALUE = MENU                        !Place selected menu item in Exch. Msg.
```

PRINT

5.8 THE PRINT STATEMENT

STATEMENT: PRINT

USE: Use the PRINT statement to specify the contents of a field on a form designed for use on a hard-copy device.

USAGE NOTES: ATL Syntax requires that the PRINT statement be followed by at least one VALUE clause.

The initial character position of the field to be printed is specified by the row and column parameters.

You may specify PRINT statements in any order.

The ATL utility builds the form definition record by inserting PRINT statements in the order specified by the values supplied for the row and column parameters.

To simplify the coding of forms, you may specify row and column values as relative offsets from the previous field through the use of the ATL "dot" constructs.

The length of a PRINT field is implicitly defined by the VALUE clause.

FORMAT:

```
PRINT = row, column

    VALUE =  { "string"                  }  [   { "string"                  }      ]
             { FILL ("character, count)  }  [   { FILL ("character", count) }      ]
             { DATE                      }  [   { DATE                      }      ]
             { TIME                      }  [ , { TIME                      } ,... ]
             { TRANSACTION               }  [   { TRANSACTION               }      ]
             { NAME                      }  [   { NAME                      }      ]
             { STATION                   }  [   { STATION                   }      ]
             { REQUEST (position, count) }  [   { REQUEST (position, count) }      ]
```

Clauses and Parameters:

Clause	Parameter	Description
	row	The value you supply for row corresponds to the terminal line where the field you are defining is printed. The row number is defined relative to line 1, which is the first printed line of the form.

Clause	Parameter	Description
	,column	The value you supply for column corresponds to the character position on the row where the field begins.
		To simplify coding of forms, you may specify row and column values using the ATL "dot" constructs.
VALUE =		The VALUE clause allows you to specify the contents of a PRINT field. You may specify more than one parameter to the same value clause. You may also specify the same parameter more than once in the same value clause.
	"string"	A string of characters enclosed by quotation marks. This might be used to supply column headings on a hard-copy report.
	FILL ("character",count)	The character specified by the first parameter is written in the PRINT field the number of times specified in the second parameter.
	DATE	The current date is written to the PRINT field in the format: DD-MMM-YY
	TIME	The current clock time is written to the PRINT field in the format: HH:MM:SS
	TRANSACTION	The system-determined transaction instance number is written to the PRINT field as ten ASCII characters.
	NAME	The 6-character name of the current transaction is written to the PRINT field.
	STATION	The 6-character terminal station ID of the initiating terminal is written to the PRINT field.
	REQUEST (position,count)	The REQUEST function allows you to obtain information contained in the REPORT message that requested this form. A PRINT field specified with a REQUEST function in the VALUE clause is filled with text from the requesting REPORT message. The filling operation begins at the character position specified by the first parameter, and continues for the number of characters specified by the second parameter.
		You may also use the dot constructs to specify the character position parameter. The first character position

Clause	Parameter	Description
		in the requesting response message can be referenced by specifying 1 as the first parameter of the REQUEST function.

EXAMPLES:

The following PRINT statement demonstrates the use of the REQUEST function. Six characters from the requesting response message are placed into the PRINT field that begins at column 20 of the first printer line.

```
PRINT = 1,20                          !Position Print Field
        VALUE =                       !Fill this field field with
                REQUEST (1,6)         !First Six Chars of Message
```

5.9 THE PROMPT STATEMENT

STATEMENT: PROMPT

USE: The PROMPT statement allows you to specify PROMPT fields in the FORM area of the terminal screen. Prompt fields are most commonly used to display user instructions.

USAGE NOTES: The cursor cannot be positioned in a Prompt field and the terminal user cannot change its contents. Prompt fields are written by the terminal station and may be overwritten as part of a REPLY. They are not transmitted to the terminal station when a user function key is pressed.

ATL Syntax requires that you specify at least one clause following each PROMPT Statement.

You must specify row and column parameters following the PROMPT keyword.

The length of a Prompt field can be defined either explicitly in a length clause or implicitly by the VALUE clause. When both clauses are present, ATL assigns the greater of the two lengths to the field.

FORMAT:

```
PROMPT = row, column
/ [ VALUE = { "string"                        } [ , { "string"                        } ... ] ] \
|           { FILL ("character, count)        }     { FILL ("character", count)       }         |
|           { DATE                            }     { DATE                            }         |
|           { TIME                            }     { TIME                            }         |
|           { TRANSACTION                     }     { TRANSACTION                     }         |
|           { NAME                            }     { NAME                            }         |
|           { STATION                         }     { STATION                         }         |
|           { REQUEST (position, count)       }     { REQUEST (position, count)       }         |
|                                                                                                |
|   [LENGTH = count]                                                                             |
|                                                                                                |
|   [ATTRIBUTE = NORMAL ]                                                                        |
|   [            REVERSE]                                                                        |
|                                                                                                |
\   [LABEL = label-name]                                                                         /
```

Clauses and Parameters:

Clause	Parameter	Description
	row	The value you supply for the row parameter tells the terminal the screen row where the field you are defining is written. The row number is always calculated relative to the first line of the Form area.
	,column	The value you supply for the column parameter specifies the character position on the row where the field begins.
		To simplify the coding of forms, you may specify row and column values as relative offsets from the previous field through the use of the ATL "dot" constructs.
VALUE =		You use the VALUE clause to specify the initial contents of a PROMPT field.
		You can specify more than one value for a VALUE clause by supplying several parameters. You can also specify a valid parameter more than once as part of a VALUE clause.
	"string"	A string of characters enclosed by quotation marks. The enclosed string is displayed on the screen exactly as typed, without the quotation marks.
	FILL ("character",count)	The character you specify in the first parameter is written into the Prompt field the number of times specified by the second parameter.
	DATE	The current date is written into the Prompt field in the format: DD-MMM-YY
	TIME	The current time of day is written into the Prompt field in the format: HH:MM:SS
	TRANSACTION	The system-determined transaction instance number is written into the Prompt field as ten ASCII characters.
	NAME	The 6-character name of the transaction that is currently being run from the application terminal is written into the Prompt field.
	STATION	The 6-character terminal station name is written into the Prompt field.

Clause	Parameter	Description
	REQUEST (position,count)	The REQUEST function allows you to obtain information contained in the response message that requested the current form. When you specify a Prompt field with a value clause containing the REQUEST function, the field is filled with text from the requesting response message, beginning at the character position specified by the first parameter, and continuing for the number of characters specified by the second parameter. You may also use the dot constructs to specify the character position parameter. A . by itself refers to the next available position in the requesting response message. The values .+n and .–n specify right (+) and left (–) offsets from the current position in the response message. The first character position in the requesting response message can be referenced by specifying 1 as the first parameter of the REQUEST function.
LENGTH =		Use this clause to explicitly define the length of a Prompt field.
	count	A numeric value from 1 to 1920. If the count parameter causes the field to overlap another field on the form, a syntax error is flagged by the ATL utility. If a field extends past the physical right margin, that field is "wrapped around" to continue in the first column of the next line.
ATTRIBUTES =		The ATTRIBUTES clause allows you to specify the way that Prompt fields are displayed on the terminal screen.
	NORMAL	The Prompt field appears as white characters on a dark background.
	REVERSE	The Prompt field appears as dark characters on a white background. Spaces appear as white squares.
LABEL =		Use the LABEL clause to specify a label for a Prompt field. A Prompt field must be labelled if you intend to reference it in an ATL REPLY statement.

Clause Parameter Description

label-name

A label can be from 1 to 30 characters long. The first character must be a letter and the remaining characters are limited to:

A through Z,
a through z,
0 through 9,
period (.),
hyphen (-),
underline (_).

In addition, you can use the percent symbol (%) or the dollar sign ($) as the last character in a label.

The label character set allows you to use the same data item names in both forms and TSTs. However, if the label-name is the same as (or part of) an ATL statement, clause or parameter keyword, the ATL utility issues a Fatal error message.

You can avoid this possibility by always including a special character (.), (-), (_), (%), or ($) in a label name. These characters are never used in an ATL reserved keyword. See Table 2-1 for the list of ATL reserved words.

EXAMPLES:

A PROMPT field is most often used to give instructions or information in the form area of a field. In the following example, three PROMPT fields are shown. The first field identifies the data contained on the line. The second field is displayed as six (?) characters, and labelled so that it can be referenced by a later reply. The third field tells the user that the preceding field contents will be assigned by the application TST.

```
PROMPT =  1,1                          !Skip a line, move to column 1
        VALUE = “Customer Number”      !Label the empty field

PROMPT =  .,.+4                        !Field starts 4 spaces past prompt
        LABEL = CUSTOMER.NUMBER        !Assign LABEL to allow filling by REPLY
        VALUE = “??????”               !Fill with ? - System assigns value
        ATTRIB = REVERSE               !Highlight in REVERSE video

PROMPT =  .,30                         !Skip to Column 30 on this line
        LENGTH = 30                    !Define Length for blanking out later
        VALUE = “(To be Supplied by System)”
        LABEL = CUSTNO.TEXT            !Label for Blanking out Later
```

Note that even though these three Prompt fields are contiguous, each field is defined separately. This is to allow the fields to be referenced individually by a REPLY statement.

5.10 THE REPEAT AND REND STATEMENTS

STATEMENT: REPEAT and REND

USE: The REPEAT and REND statements allow you to reduce coding effort when defining many similar fields in a single form definition. When you specify a block of ATL statements delimited by the REPEAT and REND statements, the ATL utility repeats that block of statements the number of times specified in the parameter to the REPEAT statement.

USAGE NOTES: Any statement may occur in a REPEAT block except another REPEAT statement or the END statement.

There is no limit on the number of Repeat blocks you may specify in a form definition.

The Repeat block begins with the first statement that follows a WITH clause, and continues until the REND statement is encountered.

You must specify at least one WITH clause when you are using the REPEAT statement. You may specify up to 36 dummy parameters.

Dummy parameters may be specified anywhere in an ATL statement or clause.

If the WITH clause parameter that follows the equal (=) sign is enclosed in quotation marks (" ", or ' '), that parameter is assumed to be a character parameter.

FORMAT:

```
REPEAT = number

        ( [WITH #n = start [,increment]] )
        < [WITH #n = "character" ]       >
REND    ( • • •                          )
```

Clauses and Parameters:

Clause	Parameter	Description
	number	The number of times you want the ATL utility to repeat the statements contained in the Repeat block. Any dummy parameters in those statements are

Clause	Parameter	Description
		replaced by integers or characters according to the parameters specified in the WITH clauses that follow the REPEAT statement.
WITH		Allows you to set up dummy parameters inside a Repeat block. The dummy parameter specified in the WITH clause and the Repeat block is modified according to the parameter values of the WITH clause. Two types of dummy parameters are allowed: integer, and character.
	#n=	The dummy parameter must be a character in the range 0 through 9 and A through Z.
	start	The value initially assigned to the dummy integer parameter. This parameter may take any numeric value.
	[,increment]	The optional increment to be applied to the dummy integer parameter. If no increment is specified, 1 is assumed by default. On each pass through a Repeat block, the ATL utility ups the value of the dummy integer parameter by the value specified for an increment.
	"character"	The first time the ATL utility processes the statements in the Repeat block, it replaces the dummy character parameter with the value specified in this parameter as a quoted character. Subsequent passes through the block cause the starting value to be incremented in the ascending sequence: 0 through 9, then A through Z. For example, if you specify "A" as the initial character parameter in a WITH clause, the first set of statements in that repeat block will have the letter "A" substituted by the ATL utility. The second set of statements created from the Repeat block will have the letter "B". The substitution sequence continues until the REPEAT statement count is exhausted. Any attempt to increment beyond Z results in an error.

EXAMPLES:
This example demonstrates how to use a Repeat block to minimize the amount of coding required to specify an ATL form definition. Four identical input fields must be coded, with their initial

values taken from a requesting response message. Two dummy variables are used. #1 is used to specify the position in the requesting message where the data to fill the field is located. The dummy parameter #A is used to assign different field label names to the Input fields being created.

```
REPEAT = 4
          WITH #1 = 1,3              !#1 IS AN INTEGER; BUMP BY 3 EACH TIME
          WITH #A ="A"               !#A IS A CHARACTER

INPUT =   .+1,10
          VALUE =REQUEST(#1,3)
          LABEL=FIELD.#A$

REND      REND
```

The 7-line Repeat block shown above is equivalent to the twelve lines of code shown below.

```
INPUT =   .+1,10
          VALUE=REQUEST(1,3)
          LABEL=FIELD.A$
INPUT =   .+1,10
          VALUE=REQUEST(4,3)
          LABEL=FIELD.B$
INPUT =   .+1,10
          VALUE=REQUEST(7,3)
          LABEL=FIELD.C$
INPUT =   .+1,10
          VALUE=REQUEST(10,3)
          LABEL=FIELD.D$
```

REPLY

5.11 THE REPLY STATEMENT

STATEMENT: REPLY

USE: The REPLY statement allows you to specify modifications to the current form displayed at the application terminal. These modifications are applied when the terminal station receives a Reply-type response message from a TST. Replies may be used to indicate successful completion of processing or error conditions, or to prompt for additional or corrected user input.

USAGE NOTES: ATL syntax requires that you specify at least one clause following a REPLY statement.

Display fields are handled in a special way. If the field is not being filled as part of a Reply, and if you specified that field with the BLANK attribute, then the field is blank-filled. This allows you to erase Display fields from previous replies.

FORMAT:
REPLY = number

```
/   WRITE = field-label [ / "string"                       \ ] [   / "string"                       \ ]
|                       [ | FILL ("character", count)      | ] [   | FILL ("character", count)      | ]
|                       [ | DATE                           | ] [   | DATE                           | ]
|                       [ | TIME                           | ] [ , | TIME                           | ]
|                       [ < TRANSACTION                    > ] [   < TRANSACTION                    >,...]
|                       [ | NAME                           | ] [   | NAME                           | ]
|                       [ | STATION                        | ] [   | STATION                        | ]
<                       [ \ REQUEST (position, count)      / ] [   \ REQUEST (position, count)      / ]
|
|  [CURSOR = field-label]
|
|  [ENABLE = keyname]
|
|  [DISABLE = keyname]
|
\  [BELL   [= periods ]]
```

Clauses and Parameters:

Clause	Parameter	Description
	number	The number parameter identifies a set of screen modifications. When a TST sends a REPLY message with a reply number specified as a parameter, then the set of screen modifications having the same reply number are

Clause	Parameter	Description
		displayed. You may specify several REPLY statements with the same number parameter. The screen that is ultimately displayed consists of the combination of all REPLY statements having the same number parameter. This highest REPLY number that you can assign is 64.
WRITE =		The WRITE clause allows you to specify modifications to be made to a previously defined field. The WRITE clause parameters identify a field, and describe how it is filled when the reply screen is displayed. The string you specify in a WRITE clause must have a length less than or equal to that of the referenced field. If the string has a length less than the field, the remainder of the field is unaltered. However, in the case of an Input field, the remainder of the field is filled with the CLEAR character specified for that field. Furthermore, in the case of a BLANK Display field, any portion of the field that is not explicitly written to by a reply is blanked by the terminal. If you specify a WRITE clause with only a label, and do not specify the field contents, the ATL utility fills the field with the VALUE clauses in the original field definition. In this case, the original field definition must have at least one VALUE clause and that VALUE clause must not contain the REQUEST ( ) parameter.
	label-name	The label-name parameter specifies the label-name of the field written into by the subsequent parameters in the WRITE clause. The field specified by this label can be an Input, Display or Prompt field but not a Menu field.
	"string"	A string of characters enclosed by quotation marks.
	FILL ("character",count)	The character specified in the first parameter is written into the field referenced by the label and the number of times specified in the second parameter.
	DATE	The current date is written to the specified field in the format: DD-MMM-YY
	TIME	The current clock time is written to the specified field in the format: HH:MM:SS

Clause	Parameter	Description
	TRANSACTION	The system-determined transaction instance number is written to the specified field as ten ASCII characters.
	NAME	The 6-character name of the current transaction is written to the specified field.
	STATION	The 6-character terminal station ID is written to the specified field.
	REQUEST (position,count)	The REQUEST function allows you to obtain information contained in the response message that requested the current reply screen. A field written using the REQUEST function in a WRITE clause is filled with text from the requesting response message beginning at the character position specified by the first parameter, and continuing for the number of characters specified by the second parameter. You may also use the dot constructs to specify the character position parameter. The first character position in the requesting response message can be referenced by specifying 1 as the first parameter of the REQUEST function.
CURSOR =		The cursor is always poisitioned on the first character of the first Input field after the display of a reply screen unless the CURSOR clause is present.
	label-name	The label of an Input field where the CURSOR is positioned after a reply screen has been displayed.
	MENU	The cursor is positioned on the first MENU field after a reply screen has been displayed.
ENABLE =		Allows you to enable user and system function keys after the terminal displays a reply screen. The ATL utility enables the ENTER and ABORT keys by default. To enable other function keys you must specify the key name in an ENABLE clause. You can specify only one key name in an ENABLE clause. To enable several function keys, you must use several clauses.

Clause Parameter Definition

keyname

You can specify the following function key names as valid parameters in an ENABLE clause:

ENTER
CLOSE
AFFIRM
STOPREPEAT
KEY0
KEYDOT
KEY1
KEY2
KEY3

DISABLE =

Allows you to disable user and system function keys after the terminal displays a reply screen.

The ATL utility enables the ENTER and ABORT keys by default. You cannot disable the ABORT key. You may disable the ENTER key.

To disable other function keys you must specify the key name in a disable clause.

You can specify only one key name in a disable clause. To disable several function keys, you must use several clauses.

keyname

You can specify the following function key names as valid parameters in a disable clause:

ENTER
CLOSE
AFFIRM
STOPREPEAT
KEY0
KEYDOT
KEY1
KEY2
KEY3

BELL

You use the BELL clause to specify that the terminal bell is to sound at the time the reply screen is displayed. This clause may be specified for any terminal equipped with an audible bell.

Clause	Parameter	Description
	[=periods]	The periods parameter is optional. If you do not specify a period parameter, the default is one period. Specifying a number of periods determines how long the bell is to ring (155 milliseconds/period for the VT62). For example, specifying BELL=3 causes the bell on a VT62 terminal to sound for 465 ms. when a reply screen is displayed.

EXAMPLES:

The following example demonstrates how to code a Display field that is displayed only at the time a REPLY is sent to the terminal:

```
DISPLAY = 1,1                              !Text goes in upper left corner
          VALUE = "Error in processing"
          ATTRIBUTES = BLANK               !Specify the field as blank
          LABEL = ERROR.LINE               !Label for use in Reply Message

REPLY =   1                                !Specify number for this reply screen
          WRITE = ERROR.LINE               !Use label to Write the reply line.
```

REPLY statements can write to Input, Display, Menu, and Prompt fields using the request clause. In the following example, a full set of modifications are made to the Add Customer Form. Notice how the 6-character customer number is written into a Prompt field using the ATL REQUEST function.

```
REPLY = 1
          DISABLE = ENTER
          !
          ! Write the success message in the Display area
          !
          WRITE = REPLY.TEXT.B," ★★★ TRANSACTION COMPLETE ★★★ "
          !
          ! Blank the original user instructions
          !
          WRITE = INSTR.TEXT,FILL(" ",72)
          !
          ! Write the new customer number over ??????
          !
          WRITE = CUSTOMER.NUMBER,REQUEST(1,6)
          !
          ! Blank out the words: (To Be Supplied by System)
          !
          WRITE = CUSTNO.TEXT,FILL(" ",30)
          !
          ! Blank out original function key instructions
          !
          WRITE = KEY.PROMPT,FILL(" ",75)
          !
          ! Write new function key instructions
          !
          WRITE = KEY.PROMPT,"Function Keys: Press AFFIRM to Add Another ",
                   "Customer–Press CLOSE to quit"
```

APPENDIX A
ATL UTILITY ERROR MESSAGES

A.1 WARNING MESSAGES

% ATTRIBUTE specified is not legal for this statement.
The ATTRIBUTE is ignored.

% ATTRIBUTE previously specified in this statement.
The subsequent ATTRIBUTE specification is ignored.

% BELL count out of range.
The assumed count is 1. The permitted range is (1 to 255).

% COL parameter outside of device width. Field will wrap around.

% CURSOR value multiply-defined.
The CURSOR is positioned at the last field specified.

% DISPLAY statement not legal for this device type.
Display fields are not allowed on form used by an output-only device.

% FEED value for LENGTH clause allowed only on form statement.
The LENGTH clause is ignored.

% FEED value for LENGTH clause not valid for this device type.
The LENGTH=FEED clause is not legal for VT62 forms.

% Illegal Character "x"–Assumed as SPACE

% INPUT field declaration required for SELECTion through INPUT field

% INPUT statement not legal for this device.
INPUT fields cannot be specified on forms for output-only devices.

% LENGTH value out of device range.
The statement is ignored.

% MENU field declaration required for MENU SELECTion.

% MENU statement not legal for this device type.

% MESSAGE statement not legal for this device type.

% MESSAGE statement references MENU VALUE, but no MENU field defined.

% NOECHO INPUT field longer than maximum of 40 characters.

% Non-positive MESSAGE position illegal, assumed to be 1.

% Non-positive COL parameter illegal.

% NULL string for KEYCAP.
The clause is ignored.

% Only one NOECHO INPUT field allowed
The form will not run on a VT62.

% Only one key value allowed in MESSAGE statement.

% Only one (1) SELECT clause permitted per form definition.
The last SELECT clause is used by the compiler.

% Previous field length exceeded (device) (page) width. Field will wrap around

% PRINT statement not legal for this device type.
A form used on a VT62 cannot have a PRINT statement.

% PROMPT statement not legal for this device type.

% REPLY # out of range (1-64).
The SELECT clause specifies an invalid REPLY #.

% REPLY statement not valid for this device type.

% ROW parameter outside of area, assumed to be 1.

% Should be single letter.
The FILL function requires a single-letter text string.

% Should use a single character string.
The parameter to the CLEAR clause must be a single-letter string.

% Some user function keys must be enabled if form contains MESSAGE statement.
The form will compile. However, it will not run properly.

% Some user function keys must be enabled for SELECT statement.
The form will compile, However, it will not run properly.

% SPLIT value changed previously.
SPLIT clause is ignored.

% System keys may not have KEYCAPs.
The KEYCAP clause is ignored.

% Too late to change SPLIT value.
The SPLIT clause is ignored.

% Too late to change WIDTH.
The WIDTH clause is ignored.

% Unable to determine MESSAGE length because of unspecified VALUE element.

% WIDTH clause legal only for OUTPUT-ONLY devices.
The WIDTH clause is ignored on a form used by a VT62.

% WIDTH value out of device range.
The WIDTH clause is ignored.

% WITH clause variable initial value not letter or digit. 0 assumed.

% WITH clause variable should be single letter.
All letters after the first letter in the variable are ignored.

A.2 FATAL ERROR MESSAGES

? COL parameter outside of defined device width.

? Copied WRITE VALUE clause is REQUEST type.
This error is caused by a REPLY statement WRITE clause that references a Display field initially specified with both the BLANK attribute, and a REQUEST function.

? DISPLAY/MENU field #mm overlaps DISPLAY/MENU field #nn.

? Field wraparound exceeds form area.

? Field wraparound exceeds display area.

? Field wraparound exceeds page length

? Forms Definition Record too big–aborting.
The forms definition record must not exceed 8191 bytes.

? Illegal REPLY statement number.
The REPLY number must be in the range (1-64).

? Illegal VALUE clause for this statement type.

? INPUT/PROMPT field #mm overlaps INPUT/PROMPT field #nn

? Integer out of range

? LABEL clause not legal for this statement.

? LENGTH value out of range.
LENGTH set to 1 for remainder of compilation.

? MESSAGE field #mm overlaps MESSAGE field #nn.

? Missing text string delimiter

? Only letter or digit can follow #.

? Premature end to input–aborting

? Previous field length exceeded page width.

? PRINT field #mm overlaps PRINT field#nn.

? REPLY #nn CURSOR INPUT field can't be modified.

? REPLY #nn CURSOR clause references non-existent MENU field.

? REPLY #nn CURSOR field not INPUT field.

? REPLY #m WRITE #n has no VALUEs.

? REPLY #m, WRITE #n VALUEs too long for WRITE referenced field.

? REPLY #m WRITE #n does not reference INPUT, PROMPT, DISPLAY, or PRINT field.

? REPEAT variable #nn–exceeds "Z"

? SELECT INPUT field referenced by more than one statement.

? SELECT INPUT field can't be modified.

? SELECT REPLY #2 not defined.

? SELECT REPLY #1 not defined.

? SELECT field not INPUT type.

? SPLIT value out of device range.

? Syntax Error

? Tables too large
No room is left in the ATL utility's tables.

? Text string too long.
A text string in a MESSAGE statement must be less than 255 characters in length.

? Too many INPUT fields.
You have exceeded 127 INPUT fields on a single form.

? Too many INPUT and MENU fields.
You have exceeded 128 INPUT and MENU fields on a single form.

? Too many MENU fields
You have exceeded 127 MENU fields on a single form.

? Total length of KEYCAP strings exceeds maximum of 214 characters.

? Undefined LABEL: xxxxxx.

? VALUE clause field type no INPUT.

? Zero length MESSAGE declared.

A.3 ATL UTILITY DIALOG ERROR MESSAGES

% Alphanumeric or asterisks required
The response to this question has characters other than those in the character set A through Z, a through z, 0 through 9, and the asterisk.

% ATL source file "xxxxxx" does not exist or is LOCKed.
The source file you specified cannot be found, or is LOCKed by another task.

% Can't find forms definition record "xxxxxx".
You attempted to DELETE a forms definition record that doesn't exist.

% Can't find forms definition record "xxxxxx", changing to ADD command.
You attempted to REPLACE a form definition that didn't exist.

% Forms definition record "xxxxxx" already exists.
You attempted to ADD a form definition record, when one already exists with the same name.

% Illegal file name "xxxxxx"
The file name you specified does not conform to TRAX file specification rules.

% Input required
You must answer this question. The utility remains at that point in the dialog until acceptable input has been supplied. If you don't know what input is required, you can display the help text by typing a question mark, or you may exit by typing CTRL Z. You may then consult the appropriate documentation.

% Integer required
The number you specified must be an integer value.

% Invalid response

Your response was not acceptable to the utility. Consult the help text or the documentation to determine the legal responses for this question.

% Number exceeds maximum

The number you specified exceeds the maximum value permitted by the utility. Consult the help text or the documentation for this question.

% Numeric required

Your response must be numeric. Legal numeric characters are 0 through 9.

% Octal value required

File version numbers must be octal numbers. You have typed a digit (8 or 9) that is unacceptable as an octal number.

% Positive integer required

The number you specify must be a positive integer value.

% Response exceeds maximum length

Your response to this question exceeds the maximum length permitted by the utility. Consult the help text or the documentation for the permissible length of transaction processor component names.

% Response not unique

The utility could not determine your response because the keyword was not unique. Insure that you specify at least the number of characters to uniquely describe the keyword you are entering.

% Selected command requires transaction processor name–restarting dialog

Once you specify the COMPILE command, the transaction processor name is erased by the ATL utility. If you invoke any other command after the COMPILE command, The ATL utility displays this error message and returns to the question: "Transaction processor name <" ">?".

% Transaction processor "tpname" does not exist.

You specified a transaction processor name, but the utility couldn't find the file [1,300] tpname.FDF.

APPENDIX B
ATL LANGUAGE SUMMARY

B.1 CONVENTIONS USED TO DESCRIBE THE LANGUAGE

[ ] Special brackets indicating optional information that can be omitted from a statement or clause.

{ } Braces indicating that a choice of one or more parameters must be made from the set enclosed by the braces. You can also specify a parameter more than once as part of the same clause.

Lower-case letters Parameters described in lower-case letters indicate data that you must supply such as a label, number, or text-string.

UPPER-CASE LETTERS Parameters shown in upper-case are keywords. They must be specified and spelled as shown in the parameter list.

Bold Face Keywords The default parameter keyword values assumed for certain clauses (ATTRIBUTES, for example) are shown in bold face.

●●● Ellipsis indicate clauses and parameters can be repeatedly specified.

B.2 ATL STATEMENTS AND CLAUSES

```
DEFAULT
[                    ( { ANY          }  )    [     {     }     ] ]
[  ATTRIBUTES =      ( { ALPHANUMERIC }  )    [     {     }     ] ]
[                    ( { LETTERS      }  )    [     {     }     ] ]
[                    ( { NUMERIC      }  )    [     {     }     ] ]
[                    ( { SIGNED       }  )    [     {  ≡  }     ] ]
[                    (                   )    [     {     }     ] ]
[                    ( { LEFT  }         )    [     {     }     ] ]
[                    ( { RIGHT }         )    [     {     }     ] ]
[                    (                   )    [     {  ≡  }     ] ]
[                    ( { NOTAB }         )    [     {     }     ] ]
[                    ( { TAB   }         )    [     {     }     ] ]
[                    (                   )    [     {     }     ] ]
[                    ( { NOFULL }        )    [  ,  { ... }     ] ]
[                    ( { FULL   }        )    [     {     }     ] ]
[                    (                   )    [     {     }     ] ]
[                    ( { NOREQUIRED }    )    [     {  ≡  }     ] ]
[                    ( { REQUIRED   }    )    [     {     }     ] ]
[                    (                   )    [     {     }     ] ]
[                    ( { MODIFY   }      )    [     {     }     ] ]
[                    ( { NOMODIFY }      )    [     {  ≡  }     ] ]
[                    (                   )    [     {     }     ] ]
[                    ( { NORMAL  }       )    [     {     }     ] ]
[                    ( { REVERSE }       )    [     {     }     ] ]
[                    (                   )    [                 ] ]
[                    ( { BLANK   }       )    [                 ] ]
[                    ( { NOBLANK }       )    [                 ] ]

[ ENABLE = keyname ]

[ DISABLE = keyname ]

[ CLEAR = “character”]
```

```
DISPLAY = row, column
  [VALUE = { "string"                    } [ , { "string"                    } ] ... ]
  [        { FILL ("character", count)   } [   { FILL ("character", count)   } ]     ]
  [        { DATE                        } [   { DATE                        } ]     ]
  [        { TIME                        } [   { TIME                        } ]     ]
  [        { TRANSACTION                 } [   { TRANSACTION                 } ]     ]
  [        { NAME                        } [   { NAME                        } ]     ]
  [        { STATION                     } [   { STATION                     } ]     ]
  [        { REQUEST (position, count)   } [   { REQUEST (position, count)   } ]     ]

  [ LENGTH = count ]

  [ATTRIBUTES =  { { NORMAL  } }  [ , { = } ] ]
  [              { { REVERSE } }  [   {   } ] ]
  [              {             }  [   {   } ] ]
  [              { { BLANK   } }  [   { = } ] ]
  [              { { NOBLANK } }  [   {   } ] ]

  [ LABEL = label-name ]

END

FORM
{ [ SPLIT = length ]                                                              }
{ [ ENABLE = keyname ]                                                            }
{ [ DISABLE = keyname ]                                                           }
{ [ KEYCAP = keyname, "text-string" ]                                             }
{ [ SELECT = { MENU              }     ,reply-1     { ,reply-2       } ]          }
{ [          { input-field-label }                  { ,NOAUTHORIZE   } ]          }
{ [ LENGTH = { line-count } ]                                                     }
{ [          { FEED       } ]                                                     }
{ [ BELL   [= periods] ]                                                          }
{ [ WIDTH = form-width ]                                                          }
```

```
INPUT = row, column
 ⎛⎡                  ⎛ ⎧ ANY          ⎫ ⎞ ⎡                    ⎤ ⎤ ⎞
 ⎜⎢ATTRIBUTES =      ⎜ ⎨ ALPHANUMERIC ⎬ ⎟ ⎢   ⎧  ———  ⎫        ⎥ ⎥ ⎟
 ⎜⎢                  ⎜ ⎪ LETTERS      ⎪ ⎟ ⎢   ⎪  ———  ⎪        ⎥ ⎥ ⎟
 ⎜⎢                  ⎜ ⎪ NUMERIC      ⎪ ⎟ ⎢   ⎪  ———  ⎪        ⎥ ⎥ ⎟
 ⎜⎢                  ⎜ ⎩ SIGNED       ⎭ ⎟ ⎢   ⎪  ———  ⎪        ⎥ ⎥ ⎟
 ⎜⎢                  ⎜                  ⎟ ⎢   ⎪  ———  ⎪        ⎥ ⎥ ⎟
 ⎜⎢                  ⎜ { LEFT  }        ⎟ ⎢ , ⎨  • • • ⎬       ⎥ ⎥ ⎟
 ⎜⎢                  ⎜ { RIGHT }        ⎟ ⎢   ⎪  ———  ⎪        ⎥ ⎥ ⎟
 ⎜⎢                  ⎜                  ⎟ ⎢   ⎪  ———  ⎪        ⎥ ⎥ ⎟
 ⎜⎢                  ⎜ { NOTAB }        ⎟ ⎢   ⎪  ———  ⎪        ⎥ ⎥ ⎟
 ⎜⎢                  ⎜ { TAB   }        ⎟ ⎢   ⎩  ———  ⎭        ⎥ ⎥ ⎟
 ⎜⎢                  ⎜                  ⎟ ⎣                    ⎦ ⎦ ⎟
 ⎜⎢                  ⎜ { NOFULL }       ⎟                              ⎟
 ⎜⎢                  ⎜ { FULL   }       ⎟                              ⎟
 ⎜⎢                  ⎜                  ⎟                              ⎟
 ⎜⎢                  ⎜ { NOREQUIRED }   ⎟                              ⎟
 ⎜⎢                  ⎜ { REQUIRED   }   ⎟                              ⎟
 ⎜⎢                  ⎜                  ⎟                              ⎟
 ⎨⎢                  ⎜ { MODIFY   }     ⎟                              ⎬
 ⎜⎢                  ⎜ { NOMODIFY }     ⎟                              ⎟
 ⎜⎢                  ⎜                  ⎟                              ⎟
 ⎜⎢                  ⎜ { NORMAL  }      ⎟                              ⎟
 ⎜⎢                  ⎜ { REVERSE }      ⎟                              ⎟
 ⎜⎢                  ⎜                  ⎟                              ⎟
 ⎜⎣                  ⎝ { NOECHO }       ⎠                              ⎟
 ⎜⎡VALUE =   ⎧ "string"                   ⎫ ⎡ ⎧ "string"                   ⎫    ⎤⎤ ⎟
 ⎜⎢          ⎪ FILL ("character", count)  ⎪ ⎢ ⎪ FILL ("character", count)  ⎪    ⎥⎥ ⎟
 ⎜⎢          ⎪ DATE                       ⎪ ⎢ ⎪ DATE                       ⎪    ⎥⎥ ⎟
 ⎜⎢          ⎪ TIME                       ⎪ ⎢ ⎪ TIME                       ⎪    ⎥⎥ ⎟
 ⎜⎢          ⎨ TRANSACTION                ⎬ ⎢,⎨ TRANSACTION                ⎬,•••⎥⎥ ⎟
 ⎜⎢          ⎪ NAME                       ⎪ ⎢ ⎪ NAME                       ⎪    ⎥⎥ ⎟
 ⎜⎢          ⎪ STATION                    ⎪ ⎢ ⎪ STATION                    ⎪    ⎥⎥ ⎟
 ⎜⎣          ⎩ REQUEST (position, count)  ⎭ ⎣ ⎩ REQUEST (position, count)  ⎭    ⎦⎦ ⎟
 ⎜  [LENGTH = count]                                                               ⎟
 ⎜  [CLEAR = "character"]                                                          ⎟
 ⎝  [LABEL = label-name]                                                           ⎠

MENU = row, column

   VALUE =   ⎧ "string"                   ⎫ ⎡ ⎧ "string"                   ⎫    ⎤
             ⎨ FILL ("character", count)  ⎬ ⎢,⎨ FILL ("character", count)  ⎬,•••⎥
             ⎩ REQUEST (position, count)  ⎭ ⎣ ⎩ REQUEST (position, count)  ⎭    ⎦
```

```
MESSAGE = position

    VALUE =  { "string"                      } [ { "string"                      }      ]
             { FILL ("character", count)     } [ { FILL ("character", count)     }      ]
             { DATE                          } [ { DATE                          }      ]
             { TIME                          } [ { TIME                          }      ]
             { TRANSACTION                   } [ { TRANSACTION                   }      ]
             { NAME                          } [,{ NAME                          },•••  ]
             { STATION                       } [ { STATION                       }      ]
             { REQUEST (position, count)     } [ { REQUEST (position, count)     }      ]
             { MENU                          } [ { MENU                          }      ]
             { label-name                    } [ { label-name                    }      ]
             { KEY                           } [ { KEY                           }      ]

PRINT = row, column

    VALUE =  { "string"                      } [ { "string"                      }      ]
             { FILL ("character, count)      } [ { FILL ("character", count)     }      ]
             { DATE                          } [ { DATE                          }      ]
             { TIME                          } [,{ TIME                          },•••  ]
             { TRANSACTION                   } [ { TRANSACTION                   }      ]
             { NAME                          } [ { NAME                          }      ]
             { STATION                       } [ { STATION                       }      ]
             { REQUEST (position, count)     } [ { REQUEST (position, count)     }      ]

PROMPT = row, column

{ [VALUE =   { "string"                      } [ { "string"                      }      ]] }
{ [          { FILL ("character, count)      } [ { FILL ("character", count)     }      ]] }
{ [          { DATE                          } [ { DATE                          }      ]] }
{ [          { TIME                          } [,{ TIME                          },•••  ]] }
{ [          { TRANSACTION                   } [ { TRANSACTION                   }      ]] }
{ [          { NAME                          } [ { NAME                          }      ]] }
{ [          { STATION                       } [ { STATION                       }      ]] }
{ [          { REQUEST (position, count)     } [ { REQUEST (position, count)     }      ]] }
{                                                                                          }
{ [LENGTH = count]                                                                         }
{                                                                                          }
{ [ATTRIBUTE = NORMAL ]                                                                    }
{ [            REVERSE]                                                                    }
{                                                                                          }
{ [LABEL = label-name]                                                                     }

REPEAT = number

          { [WITH #n = start [,increment]] }
          {                                }
          { [WITH #n = "character" ]       }
          {                                }
REND      { • • •                          }
```

```
REPLY = number

⎧  WRITE = field-label ⎡⎧ "string"                     ⎫⎤ ⎡ ⎧ "string"                     ⎫    ⎤
⎪                      ⎢⎪ FILL ("character", count)    ⎪⎥ ⎢ ⎪ FILL ("character", count)    ⎪    ⎥
⎪                      ⎢⎪ DATE                         ⎪⎥ ⎢ ⎪ DATE                         ⎪    ⎥
⎪                      ⎢⎨ TIME                         ⎬⎥ ⎢ ⎨ TIME                         ⎬    ⎥
⎪                      ⎢⎪ TRANSACTION                  ⎪⎥ ⎢,⎪ TRANSACTION                  ⎪,...⎥
⎪                      ⎢⎪ NAME                         ⎪⎥ ⎢ ⎪ NAME                         ⎪    ⎥
⎪                      ⎢⎪ STATION                      ⎪⎥ ⎢ ⎪ STATION                      ⎪    ⎥
⎨                      ⎣⎩ REQUEST (position, count)    ⎭⎦ ⎣ ⎩ REQUEST (position, count)    ⎭    ⎦
⎪ [CURSOR = field-label]
⎪
⎪ [ENABLE = keyname]
⎪
⎪ [DISABLE = keyname]
⎪
⎩ [BELL    [= periods ]]
```

INDEX

Trax Application Terminal
Language Reference Manual
AA-D330A-TC

READER'S COMMENTS

NOTE: This form is for document comments only. DIGITAL will use comments submitted on this form at the company's discretion. If you require a written reply and are eligible to receive one under Software Performance Report (SPR) service, submit your comments on an SPR form.

Did you find this manual understandable, usable, and well-organized? Please make suggestions for improvement.

Did you find errors in this manual? If so, specify the error and the page number.

Please indicate the type of reader that you most nearly represent.

- □ Assembly language programmer
- □ Higher-level language programmer
- □ Occasional programmer (experienced)
- □ User with little programming experience
- □ Student programmer
- □ Other (please specify)____________________

Name____________________ Date__________

Organization____________________

Street____________________

City__________ State__________ Zip Code
or
Country__________

Please cut along this line.

-- **Fold Here** --

--- **Do Not Tear - Fold Here and Staple** ---