CA TPX™ Session Management

ACL/E Programming Guide

Release 5.3

CA Technologies Product References

This document references the following CA Technologies products:

- CA TPX™ Session Management (CA TPX)
- CA STX™ (CA STX)
- CA ACF2® Security (CA ACF2)
- CA Top Secret® Security (CA Top Secret)
- CA IDMS™ Database (CA IDMS Database)
- CA IDMS™/DC Database (CA IDMS/DC Database)
- CA 7®Job Management (CA 7)
- CA Remote Console™ (CA Remote)
- CA TCPaccess™ Telnet Server (CA TCPaccess Telnet Server)
- CA Vman™ (CA Vman)

Contact CA Technologies

Contact CA Support

For your convenience, CA Technologies provides one site where you can access the information that you need for your Home Office, Small Business, and Enterprise CA Technologies products. At http://ca.com/support, you can access the following resources:

- Online and telephone contact information for technical assistance and customer services
- Information about user communities and forums
- Product and documentation downloads
- CA Support policies and guidelines
- Other helpful resources appropriate for your product

Providing Feedback About Product Documentation

If you have comments or questions about CA Technologies product documentation, you can send a message to techpubs@ca.com.

To provide feedback about CA Technologies product documentation, complete our short customer survey which is available on the CA Support website at http://ca.com/docs.

Contents

Chapter 7: Working With Data on the Screen 75

Chapter 8: Defining and Manipulating Variables 85

Appendix B: Sample Programs 107

Appendix C: Actions and Commands 113

Appendix D: Command Summary 119

Chapter 1: ACL/E Overview

This section contains the following topics:

What Is ACL/E?

Automated Conversation Language/Extended (ACL/E) is a high-level programming language oriented to the 3270 keyboard. ACL/E automates routine interactions, such as application session signon and signoff, and provides powerful tools that enhance the way users see and use applications. For example, ACL/E works with the CA TPX panel manager to display alternate screen images. This capability serves a wide range of purposes, from supplying customized help panels for applications to re-engineering user interfaces for SAA compliance. ACL/E's ability to coordinate multiple application sessions provides state-of-the-art application integration.

What ACL/E Commands Can Do

ACL/E provides commands to:

- Perform arithmetic and logical operations
- Examine and manipulate the current screen image
- Interact with applications
- Issue CA TPX commands
- Define and manipulate variables
- Display an alternate screen image
- Integrate multiple sessions

If you include an ACL/E signon program in the user's application session definition, the program executes automatically when the user establishes the application session associated with it. Similarly, if you include a signoff program in the user's application session definition, the program executes when the session associated with it is terminated. ACL/E provides a large number of predefined variables that represent signon data, screen data, the terminal, and input from the user and the application. You can also define your own variables.

ACL/E programs can receive input from, and send output to, the terminal, the application, and other ACL/E programs. They can also obtain values from user ID and application session definitions.

Together, these features make ACL/E the ideal tool, not only for automating routine 3270 terminal interactions, but also for significantly expanding the session management capabilities of CA TPX.

CB0VSCRI Library

Sample ACL/E programs are provided in the CB0VSCRI data set. The function of each program is briefly summarized in the $ACLS member.

Prerequisites

You should be familiar with CA TPX and how to use it, as this guide does not explain topics outside of ACL/E. For more information about using CA TPX, see the *User Guide*.

Conventions Used In This Guide

In this guide, the command character is represented by its default value, which is a slash (/).

Program Function keys are shown as PF*n.* For instance, Program Function Key two is represented as PF2.

Chapter 2: Syntax and Semantics

This chapter explains the syntax and semantics of ACL/E statements, commands, and variables. It will help you understand the most basic concepts of ACL/E and prepare you for the tutorial in the next chapter.

This section contains the following topics:

Statements

An ACL/E program consists of a series of statements.

Purpose

Each statement specifies a particular action to be taken.

Components

The only required component of a statement is a command. In addition, a statement can have one or two operands, a label, and a comment.

Commands

The following is an example of a statement that consists only of a command:

```
STOP
```

The STOP command terminates a program's execution. A command must be preceded by a space. If it is not, ACL/E will consider it a label.

Operands

Operands are used to make a command more specific. An example of a statement with one operand is the DOWN command, which moves the cursor down the screen for the number of lines specified by the operand. For example, the statement that moves the cursor down the screen five lines is:

```
DOWN 5
```

The command and operand are separated by at least one space. In a statement with only one operand, the operand is terminated by at least one space.

Many commands have two operands. For example, a statement that adds an integer to an accumulator has two operands. The following ADD statement adds 5 to the value of the accumulator A1:

```
ADD A1,5
```

Another example is a statement that compares a variable and a string such as "&USERID" and "USER1":

```
COMPARE '&USERID','USER1'
```

Another is a statement that causes the program to branch to another statement if a certain condition code exists. A statement such as the one below may appear after the statement comparing the value of &USERID with the string USER1. The statement says to branch to the statement labeled OKAY if it is true that the value of &USERID is the string USER1:

```
BRANCH EQ,OKAY
```

In a statement that contains two operands, the command and first operand are separated by at least one space, and the first and second operands are separated by a comma only. An error message will appear if there is a space before or after the comma.

Labels

Labels are used to identify statements that the program branches to if a condition code is set. A label must begin in position 1 and can be up to eight characters long. It must be terminated by at least one space. An example statement containing a label is:

```
OKAY    STOP
```

Comments

A comment can appear within a statement or on a line by itself. If a comment appears within a statement, it must be preceded by at least one blank. If it appears on a line by itself, it must have an asterisk in the first position (to prevent it from being treated as a label). In the following example, the sentence "End program execution" is a comment:

```
STOP      End program execution
```

"End program execution" is also a comment in this example:

```
*       End program execution
        STOP
```

Note: For a list of possible actions and the ACL/E commands you use to perform them, see the appendix Actions and Commands (see page 113). For a list of ACL/E commands and the actions you perform with them, see the appendix Command Summary (see page 119).

Strings

ACL/E operands are strings or constructions that contain strings. A string is a set of characters. The length of a string is determined by the number of characters that comprise it.

Simple Strings

A set of characters that begins with a non-null character and contains no commas, spaces, or quotes is a simple string. Simple strings do not need to be bounded by quotes. A simple string that is not bounded by quotes must be terminated by a space or a comma. The length of a simple string is the number of characters prior to the terminating space or comma. Enclosing a simple string in quotes does not change its length or value. The following examples show two ways to write the same simple string:

```
strings
'strings'
```

In each case, the length of the string is 7.

Complex Strings

Character strings that contain commas, spaces, or quotes that are to be treated as data are complex strings. Complex strings are bounded by single quotes. An example of a complex string is:

```
'This is a complex string.'
```

The length of the string includes the blanks and the period, but not the single quotes, so the length of the complex string shown above is 25.

Within quotes, a single quote or apostrophe is represented as data by two single quotes (not a double quote). For example:

```
'Jane''s book on the Hollywood musical'
'Scott Joplin''s New Rag'
```

The two single quotes count as one when determining the length of the string. So the first string above has a length of 36 and the second has a length of 22.

Numeric Strings

A numeric string contains only the characters 0 through 9. A numeric string cannot contain leading spaces.

Variable Strings

A variable string contains one or more variables that are resolved at execution time. For example, the variable string "&USERID" in the statement below is replaced by the user's actual user ID when the program executes:

```
KEY 'Hello, &USERID'
```

Accumulators

ACL/E has four accumulators, called A1, A2, A3, and A4. Accumulators can be used in place of any numeric. They can be set, incremented, decremented, and compared.

Condition Codes

ACL/E statements such as COMPARE, SEARCH, SRCHWAIT, and others return a condition code, which is then used by a BRANCH statement to determine what the program should do next.

ACL/E uses the following condition codes:

EQ

Indicates Equal or found

NE

Indicates Not equal to or not found

LT

Indicates Less than

GT

Indicates Greater than

LE

Indicates Less than or equal to

GE

Indicates Greater than or equal to

Example

The following statements compare the accumulator A1 with the integer 5. If the values are equal, a condition code of EQ is set, causing the BRANCH statement to branch to the statement labeled MORE:

```
COMPARE A1,5
BRANCH EQ,MORE
```

Chapter 3: Tutorial

This tutorial will help you learn about ACL/E through realistic programming samples. The tutorial covers the most commonly used ACL/E commands and variables.

This section contains the following topics:

Introduction

In this chapter you will learn how to create, store, start, and stop sample programs (sometimes called "scripts") that introduce essential ACL/E components. As you analyze these sample programs, you also can do exercises to modify and expand these programs. In this way, you can learn the basic concepts of ACL/E programming without concerning yourself with details and exceptions. At the same time, you can refer to other places in the guide where topics are explained in more detail.

Although the tutorial programs do not illustrate ACL/E's full power, they provide a firm foundation in its syntax and semantics as well as in the major applications of ACL/E programs.

After you are familiar with these fundamentals, you will be able to master the more advanced material presented later in this guide. And you will be ready to put ACL/E to work in your own scenarios.

To make signon and signoff programs execute automatically, you add them to users' application session definitions. For more information, see the chapter Working With CA TPX (see page 57).

Unless instructed otherwise, you should run the sample programs described in this guide from a CA TPX Notepad session.

Create, Store, Start, and Stop ACL/E Programs

You can create your ACL/E programs with any text editor, but you must store them where CA TPX expects to find them. In z/OS, your programs must be stored as members of a library referenced in the ACLLIB DD statement in your CA TPX startup procedure. In VM, your programs must be saved as the program name, with the file type TPXACL and mode C. Store the programs you create through this tutorial in your user ACL/E library.

Start ACL/E Program

To start an ACL/E program manually, you issue the ACL/E start command. To do this, type the CA TPX command character, an S, a blank, and the name of the ACL/E program, and then press the command key. If, for example, your command character is the slash (/) and your command key is PF12, you would start an ACL/E program called ACLHELP in the current session by typing the following command and then pressing PF12:

```
/S ACLHELP
```

Note: All examples in this tutorial use the slash (/) as the command character.

To start an ACL/E program in another session, you include that session's session ID in the command. For example, if you are in the TSOA session and you want to start the ACL/E program ACLHELP in the CICSA session, you would type the following command and then press the command key:

```
/S CICSA ACLHELP
```

Stop ACL/E Program

Usually, an ACL/E program will reach a STOP command and terminate. However, if a program "hangs" or goes into an endless loop, you can stop it by typing the following command and then pressing the command key:

```
/V
```

When using a non-SNA terminal (or emulator) and the keyboard is locked, you must press the Reset key to unlock the keyboard so that you can type the command. When using an SNA terminal (or emulator) and the keyboard is locked, you must press the Attn key, which takes you back to the menu from where you can issue the STOP command.

A Program That Says Hello

The ACL/E program shown below introduces you to programming with ACL/E. This program types the words "Hello, ACL/E" at your terminal.

```
KEY 'Hello, ACL/E'
STOP
```

Explanation

All ACL/E programs are made up of one or more statements containing commands specifying the operations to be performed. In addition, a statement can have one or two operands. In the program above, STOP is a statement consisting of a command only. The KEY statement contains the KEY command and a single operand. A command can begin in any position except position 1, which is reserved for labels.

When a statement includes an operand, the command and the operand must be separated by at least one space. Statements with two operands are used in A Program That Repeats a String (see page 28) in this chapter.

KEY Statement

The following KEY statement types the string of characters "Hello, ACL/E" on your screen:

```
KEY 'Hello, ACL/E'
```

The single quotes identify the operand "Hello, ACL/E" as a character string. A sequence of characters surrounded by single quotes is a character string. For a description of other types of strings, see the chapter Syntax and Semantics (see page 17).

STOP Command

The following STOP command terminates the program:

```
STOP
```

ACL/E programs execute sequentially, beginning with the first statement and going through each statement in turn until they encounter a command or condition requiring them to stop or branch to a specified statement.

Exercise

Create this program using any text editor. Call it HELLO1. Execute it by issuing the following command from the first line of a blank CA TPX Notepad screen (not the command line):

```
/S HELLO1
```

As a variation on this exercise, consider what changes you would need to make so that the program displays the words "Good Morning" instead of "Hello, ACL/E".

A Program That Greets You by User ID

A variation on the hello program is one that greets you by your user ID and types the current date and time. For example, if your user ID is USER1, the date is March 5, and the time is 11:16, the program types the following message:

```
Hello, USER1
            Today is 03/05/90
                             The time is 11:16:00
```

Sample Program

The program that produces this message is shown below:

```
*** This program greets the user by user ID ***
                        DOWN 3
                        KEY 'Hello, &USERID'
                        DOWN 1
                        KEY 'Today is &ZDATE'
                        DOWN 1
                        KEY 'The time is &ZTIME'
                        STOP
```

Note: Enter the /S command for this program from the command line of a CA TPX Notepad session.

Explanation

This example shows how variables are used in ACL/E programs and introduces the DOWN statement. The example also shows you how to add comment text to your ACL/E programs.

```
*** This program greets the...***
```

The first line in your program is a comment, which in this case briefly describes what the program does. When you execute the program, ACL/E ignores comments. Any text beginning with an asterisk in position 1 is a comment.

DOWN Statement

The following DOWN statement says to move the cursor down one line:

```
DOWN 1
```

The sample program shown above contains three of these statements. Without these statements, the program would attempt to type everything on one line,and the output would be similar to this:

```
Hello, USER1Today is 03/02/90
```

The system would display the following error message:

```
-KEY-OPERATION ENCOUNTERED OUT OF RANGE CONDITION
```

The error message indicates that the ACL/E program attempted to type into an area that does not accept input.

KEY Command

The KEY command types a string of characters on your screen just as it did inthe program that types “Hello” (as described inA Program That Says Hello inthis chapter). But the following KEY statements use variables that represent your user ID and the current date and time.

```
KEY 'Hello, &USERID'
KEY 'Today is &ZDATE'
KEY 'The time is &ZTIME'
```

&USERID, &ZDATE, and &ZTIME are variable names. ACL/E treats a group of characters preceded by an ampersand (&) and followed by a blank or other delimiter as a variable name. When you execute this program, ACL/E replaces the &ZDATE and &ZTIME with the current date and time. For the variable&USERID, ACL/E substitutes the user ID you entered to sign on to CA TPX.

The variables used in this program are examples of predefined variables available in ACL/E. You also can define your own variables. For more information about variables, see the chapter Defining and Manipulating Variables (see page 85).

Exercise

Create the program and store it as HELLO2; then execute it in the same manner as you did the program HELLO1.

A Program That Repeats a String

Here is another variation, a program that greets you by user ID ten times. The output begins on the line below the command line and is similar to this:

```
HELLO, USER1
HELLO, USER1
HELLO, USER1
HELLO, USER1
HELLO, USER1
HELLO, USER1
HELLO, USER1
HELLO, USER1
HELLO, USER1
HELLO, USER1
```

Sample Program

The program that produces this output is shown below:

```
*** This program says "hello" 10 times ***
*
             OPTION MAXI,200         maximum instructions
             CLEAR                   press the 'CLEAR' key
             SET A1,0                zero loop counter
             SET A2,10               set max loop value
MORE         NL 1                    go to next input line
             KEY 'Hello, &USERID'    say "hello."
             ADD A1,1                increment counter
             COMPARE A1,A2           test A1 and A2
             BRANCH LT,MORE          repeat til A1 = 10
             STOP
```

Explanation

In this program, statements with two operands appear for the first time. This program also introduces inline comments. An inline comment is a string of non-reserved text that is preceded by at least one space after a required command and its operands. Non-reserved text is text that does not represent an ACL/E command. In this program, the inline comments are spaced to line up at the right of the program statements.

This program also introduces the OPTION, CLEAR, SET, NL, ADD, COMPARE, and BRANCH commands.

OPTION Statement

The following OPTION statement at the beginning of the sample program sets the maximum number of instructions that the program will execute:

```
OPTION MAXI,200
```

This program is set to execute no more than 200 instructions. By specifying a maximum number of instructions, you can prevent a program from going into an uncontrolled loop.

If the program tries to execute more than the maximum number of statements, the following message is displayed:

```
SCRIPT EXCEEDED INSTRUCTION LIMIT
```

The term "script" in this message refers to the ACL/E program.

CLEAR Command

The following CLEAR command simulates pressing the Clear key and clears the terminal screen:

```
CLEAR
```

To make the program repeat the string "Hello, USER1" ten times, you could repeat a KEY statement ten times. But an easier way to repeat the string is to use a loop.

SET Command

The following SET commands assign values to a variable:

```
SET A1,0
SET A2,10
```

The variables A1 and A2 are two of four accumulators that are predefined by ACL/E. (For a more detailed discussion of accumulators, see the chapter Using Arithmetic and Logic (see page 49).) In this program, A1 and A2 control the loop. The first SET statement sets the value of the loop counter A1 to zero (0), while the second sets the maximum loop value A2 to ten (10).

NL Statement

The following NL statement does the same thing as pressing the ¿ (Enter) key on your keyboard:

```
NL 1
```

The NL statement moves the cursor down one line and positions it in the line's first input field. If you had used the DOWN statement, as in HELLO2, the program would display each line immediately below the last character in the previous line. (To see HELLO2, see A Program That Greets You by User ID (see page 26)).

This would result in the following error message

```
-KEY-OPERATION ENCOUNTERED AN OUT-OF-
RANGE CONDITION
```

The error message indicates that the output would not fit in the input field.

KEY Statement

The following KEY statement types the string "Hello USER1" on the terminal screen:

```
KEY 'Hello, &USERID'
```

Because this statement is contained in a loop that executes ten times, the statement needs to be specified only once.

ADD and COMPARE Statements

The following ADD statement increments the accumulator A1 each time the loop is executed. The following COMPARE statement then compares the values of the accumulators A1 and A2.

```
ADD A1,1
COMPARE A1,A2
```

When a COMPARE statement compares two values, it produces a condition code representing the relationship between them. In this program, a condition code of LT (less than) indicates that A1 is still less than A2. The condition code is then examined by the BRANCH statement.

BRANCH Statement

The following BRANCH statement uses the condition code to determine what action the program should take next.

```
BRANCH LT,MORE
```

In this program, the LT,MORE part of the BRANCH statement specifies that as long as the condition code is LT (A1 is less than A2), the program returns to the statement labeled MORE, which is the first line in the loop. Each time the program executes the loop, it moves the cursor down one line, types the message, increments A1 by one, and then compares A1 to A2 again.

The program continues to execute the loop until A1 is equal to A2. When A1 is equal to A2, the COMPARE statement sets a condition code of EQ. The branch statement only branches to the MORE label if A1 is less than A2, so the program proceeds to the STOP command and terminates.

EQ and LT are two of the seven condition codes used by ACL/E. For a more detailed discussion of condition codes, see the chapter Using Arithmetic and Logic (see page 49).

Exercise

Create the program and store it as HELLO3; then execute it in the same manner as you did the program HELLO1.

A Program for Signing on to TSO

ACL/E programs are most commonly used to automatically sign CA TPX users on to their application sessions. A program that signs you on to TSO is shown below. This program automates the steps you would take during a manual signon, such as checking to make sure you are at the TSO user ID prompt, typing your user ID and password or password phrase, and pressing the Enter key.

```
*** This program logs a user on to TSO ***
**
             OPTION TERM,ON          Display input and output.
             WINP                    Wait for TSO to speak.
**
USERID       SEARCH 'ENTER USERID'   Look for prompt
             BRANCH NE,AGAIN         If no prompt, go to AGAIN.
             KEY '&USERID'           Enter user ID...
             ENTER                   ...and send it to TSO.
             KEY '&PSWD' or '&PWPH'  Enter password.or password.phrase.
             ENTER                   ...and send it to TSO.
             BRANCH ANY,DONE         Go to DONE no matter what.
 **
AGAIN        ENTER                   Try again.
             BRANCH ANY,USERID       Go to USERID.
 **
DONE         STOP                    All done.
```

Explanation

This program introduces the ACL/E commands WINP, SEARCH, and ENTER. It also shows you some new ways to use the KEY and BRANCH commands. The following is an explanation of each line in the program.

OPTION Statement

The following OPTION statement tells the program to display the data on the screen as it is entered:

```
OPTION TERM,ON
```

The TSO signon screen will be displayed and you will be able to see the program enter your user ID and password.

WINP Command

The following WINP command tells the ACL/E program to wait for input from an application:

```
WINP
```

This program illustrates ACL/E's screen examination capability. In this program, WINP tells the program to halt until TSO displays information on the terminal screen.

SEARCH Statement

The following SEARCH statement is used to search for a string on the application screen:

```
SEARCH 'ENTER USERID'
```

In this program, the SEARCH command looks for the string "ENTER USERID" on the terminal screen. Like the COMPARE statement, the SEARCH statement sets a condition code, which the BRANCH statement uses to determine what to do next. In this program, the SEARCH statement sets the condition code EQ (for equal) if the screen contains the string. If the screen does not contain the string, the SEARCH statement sets a condition code of NE (for not equal). The BRANCH statement on the next line acts according to the condition set by the SEARCH statement.

BRANCH Statement

After the SEARCH statement sets a condition code, the following BRANCH statement determines what the program should do next:

```
BRANCH NE,AGAIN
```

If the condition code is NE, the branch statement tells the program to go to the line labeled AGAIN, which simulates pressing the Enter key, to attempt to display the TSO signon screen (a discussion of the ENTER command follows). If the condition code is EQ, the program proceeds to the next statement.

KEY and ENTER Commands

The following KEY and ENTER commands work together to enter your user ID and password or password phrase in the same way you would if you were manually signing on to TSO:

```
KEY '&USERID'
ENTER
KEY '&PSWD' or '&PWPH'
ENTER
```

The first KEY statement types your user ID on the screen. The ENTER statement on the next line simulates pressing the Enter key from the keyboard. The next two lines enter your password or password phrase in the same way. If you run this program, you will notice that the user ID is displayed on the screen while the password or password phrase is not, just as if you were entering the data manually.

BRANCH Statement

So far you have seen two examples of how the BRANCH statement can be used to redirect the flow of an ACL/E program. For a sample of a program that compares two accumulators to determine whether branching should occur, see A Program That Repeats a String. In the first BRANCH statement in this program, a string is compared with data on the screen to determine whether branching should occur.

The ANY value allows the BRANCH statements to tell the program to go to the specified label no matter what is going on at that point. In the following BRANCH statement, the label name is DONE. Branch statements with a value of ANY are called unconditional BRANCH statements. Unconditional BRANCH statements are analogous to GOTO statements in other programming languages in that they are used chiefly to go around other statements.

```
BRANCH ANY,DONE
```

Exercise

Follow this procedure to write and test your TSO logon program:

1. Activate your TSO session to test the program.
2. Identify the program (TSOLOG1) as the startup program for your TSO session (for instructions on setting up a startup program, see "Setting Up Automatic Startup and Termination Programs").
3. Sign off CA TPX and then sign on again. The changes you made to your user options in step 2 will not take affect until you sign off and sign on again.
4. Activate your TSO session to test the program.

Notice that if the program, as currently written, fails to find the search string, it tries again and again, because nothing stops it from returning to the USERID label when it does not find the string "ENTER USERID" on the screen. To prevent a possible endless loop, add a loop counter to stop the program when it fails to find the target string after three attempts. Use the counter in HELLO3 as a model. To see HELLO3, see A Program That Repeats a String.

Expand the program so that it takes you to the ISPF main menu after signing you on to TSO.

A Program That Displays System Information

The following program introduces ACL/E's ability to display alternate screen images. With this feature, you can do everything from providing customized help screens to completely re-engineering the user interface. The program displays the alternate screens that provide news about two computer systems. Users select news screens from a menu.

```
***           This program displays custom screens
**            providing information about Systems 1 and 2.
*****************************************************
* Menu Screen
*****************************************************
MNU           DISPLAY MNU001           Display menu
              INPUT                    Wait for user selection
              COMPARE '&TAID','PF1'    If PF1...
              BRANCH EQ,SYS1           ...go to SYS1 news
              COMPARE '&TAID','PF2'    If PF2...
              BRANCH EQ,SYS2           ...go to SYS2 news
              COMPARE '&TAID','PF3'    If PF3...
              BRANCH EQ,QUIT           ...terminate program
*****************************************************
* SYS1 Information Screen
*****************************************************
SYS1          DISPLAY SYS001           Display SYS1 news
              INPUT                    Wait for user selection
              COMPARE '&TAID','PF3'    If PF3...
              BRANCH EQ,MNU            ...return to menu
              BRANCH ANY,SYS1          If not PF3,
*                                      redisplay SYS1 news
*****************************************************
* SYS2 Information Screen
*****************************************************
SYS2          DISPLAY SYS002           Display SYS2 news
              INPUT                    Wait for user selection
              COMPARE '&TAID','PF3'    If PF3 pressed...
              BRANCH EQ,MNU            ...return to menu
              BRANCH ANY,SYS2          If not PF3,
                                       redisplay SYS2 news
*****************************************************
QUIT          STOP
```

Explanation

This program introduces the DISPLAY and INPUT commands as well as the &TAID variable.

DISPLAY Statement

The following DISPLAY statement tells ACL/E to display an alternate screen, and the operand in this statement identifies the panel:

```
DISPLAY MNU001
```

In z/OS, this value is the data set member of a library referenced in the PANELS DD statement in your CA TPX startup procedure. In VM, this is the file name with file type PANELS and mode E. The DISPLAY statement in this program tells ACL/E to display the screen contained in the data set member MNU001, which is shown here:

```
                          System News Menu
MNU001
     PF1 System 1 News
     PF2 System 2 News
===>
Press a PF key to display news
```

This menu tells users to specify whether they want System 1 or System 2 news by pressing PF1 or PF2, respectively. If a user presses PF1, a screen containing System 1 news is displayed. If the user presses PF2, a screen containing System 2 news is displayed.

INPUT Command

The following INPUT command tells the program to wait for the user to press an action key such as PF1, PF2, PF3, etc.:

```
INPUT
```

COMPARE and BRANCH Statements

The variable &TAID in the following COMPARE statements represents the key pressed by the user; that is, the last AID (attention identifier) byte received from the terminal in response to the INPUT command. If the user presses PF1, for example, &TAID is given a value of PF1.

```
COMPARE '&TAID','PF1'
BRANCH EQ,SYS1
```

The COMPARE statement compares the value of &TAID with "PF1" and sets a condition code that is used by the BRANCH statement. If the condition code is EQ (indicating that the user pressed the PF1 key) the program goes to the statement labeled SYS1, which displays the System 1 News screen. If the condition code is NE (indicating that the user did not press PF1), the program goes to the next statement, which is another COMPARE statement.

The following COMPARE statement compares the value of &TAID with "PF2" and sets a condition code that is used by the BRANCH statement:

```
COMPARE '&TAID','PF2'
BRANCH EQ,SYS2
```

If the condition code is EQ (indicating that the user pressed the PF2 key) the program goes to the statement labeled SYS2, which displays the System 2 News screen. If the condition code is NE (indicating that the user did not press PF2), the program goes to the next statement, which checks to see if the user pressed PF3. If the user did press PF3, the BRANCH statement terminates the ACL/E program.

The program uses the same logic to determine whether it should return to the menu from a news screen. If the user presses PF3 while viewing a news screen, the program returns to the statement labeled MNU that displays the menu screen (MNU001).

Create Alternate Screens

You can create panels for alternate screens using any text editor, and you must store them where CA TPX expects to find them. In z/OS, store your panel definitions as a data set member in a library referenced by the PANELS DD statement in your CA TPX startup procedure. In VM, store the panel definitions as the panel name with file type PANELS and mode E.

Exercise

Create the program and store it as SYSINFO1. Create the panels shown here in the specified file. Explain what you would need to do to display information about other systems at your site. Indicate what you would need to add to create a hierarchy of menus—for example, to display a System 1 and a System 2 menu, each listing three system-specific information screens.

Note: If the user presses a key other than PF1, PF2, or PF3, the program as it is written will display the System 1 News screen since that is the next statement in the program. Change the program so that it displays an error message if the user does not press PF1 or PF2.

The System News Menu

The following program shows the panel definition for the System News Menu (data set member MNU001):

```
)ATTR
  % TYPE(text) INTENSE(HI) SKIP(ON) COLOR(white)
  + TYPE(text) INTENSE(LOW) SKIP(ON) COLOR(green)
  _ TYPE(input) INTENSE(LOW) COLOR(white)
)BODY
+         System News Menu
&zpanel
       %PF1 +System 1 News
       %PF2 +System 2 News
    + ===>_acmd
%Press a PF key to display news
)INIT
)END
```

The System1 News Panel

The following program shows the panel for the System 1 screen (data set member SYS001):

```
)ATTR
  % TYPE(text) INTENSE(HI) SKIP(ON) COLOR(white)
  + TYPE(text) INTENSE(LOW) SKIP(ON) COLOR(green)
  _ TYPE(input) INTENSE(LOW) COLOR(white)
)BODY
+        System 1 News
&zpanel
System 1 will be down for maintenance from 6:00 p.m
Friday, May 25 to 6:00 a.m. Saturday.
         %PF3 +Return to Menu
     + ===>_acmd
)INIT
)END
```

The System2 News Panel

The following program shows the panel for the System 2 screen (data set member SYS002):

```
)ATTR
  % TYPE(text) INTENSE(HI) SKIP(ON) COLOR(white)
  + TYPE(text) INTENSE(LOW) SKIP(ON) COLOR(green)
  _ TYPE(input) INTENSE(LOW) COLOR(white)
)BODY
+         System 2 News
&zpanel
Please delete data sets you do not need.
        %PF3 +Return to Menu
     + ===>_acmd
)INIT
)END
```

Programs That Copy a Screen Image to a Data Set

The following two programs introduce ACL/E's ability to integrate multiple sessions. The programs let you copy a session screen image into a TSO data set member. The first program (SCREEN1) copies the screen data into an array, then schedules the second program (SCREEN2), which copies the stored data into a data set member.

Note: If you plan to create these programs and use them on your own system, make sure that you are using a model 2, 3, or 4 terminal. These programs will not execute correctly on a model 5 terminal, because the display is 132 columns wide.

Copy the Screen Data into an Array

Program SCREEN1, shown below, copies the screen data into an array. This program assumes that the session T2 is active.

```
                OPTION      FLOW,OFF
                OPTION      TERM,ON
                OPTION      MAXI,9999
                SET         A1,1
                SET         A2,1
                UDEFINE     LINECT,3
                SET         LINECT,&SCRNROWS
                UDEFINE     DATA(50),80
*
ULOOP           SET         DATA(&A1),&SCREEN(&A2,80)
                ADD         A1,1
                ADD         A2,80
                COMPARE     A1,&SCRNROWS
                BRANCH      LE,ULOOP
                ACLPGM      T2,SCREEN2
                STOP
```

Explanation of SCREEN1 Program

This program determines the number of lines on the screen and then goes through the data line by line, copying each line into a variable array. As soon as the number of lines read into the array is equal to the number of lines on the screen, the program schedules the second program, which copies the data in the array to a data set member.

OPTION Statement

The program SCREEN1 begins with the following OPTION statements:

```
OPTION FLOW,OFF
OPTION TERM,ON
OPTION MAXI,9999
```

- OPTION FLOW,OFF says not to record the statements executed in the CA TPX log.
- OPTION TERM,ON says to display data as it is entered. This allows you to watch the screen data being copied into the data set member.
- OPTION MAXI,9999 says to set the maximum number of instructions executed to 9999.

SET Statement

To count the number of lines, you use accumulators A1 and A2. These accumulators are initially set to 1 with the following SET statements:

```
SET A1,1
SET A2,1
```

UDEFINE Command

UDEFINE is one of three commands used for defining variables. Variables defined with the UDEFINE command are known as persistent variables because they remain in effect for all of a user's sessions until the user terminates all sessions and signs off CA TPX. The other types of user-defined variables are transient variables and global variables. Transient variables remain in effect until the ACL/E program in which they are defined terminates. The command VDEFINE is used to define transient variables. Global variables can be used by any ACL/E program running in this CA TPX, and remain in effect until they are deleted. The command GDEFINE is used to define global variables. For more information on defining variables, see Defining and Manipulating Variables (see page 85).

In this program, the following UDEFINE command is used to define a variable named LINECT with a length of three (3) characters. This variable records the number of lines on the screen being copied.

```
UDEFINE LINECT,3
SET LINECT,&SCRNROWS
```

SET Statement

The following SET statement sets the value of the LINECT variable to the total number of lines on the screen:

```
SET LINECT,&SCRNROWS
```

&SCRNROWS is an ACL/E variable which represents the total number of rows (or lines) on the screen. For example, if a screen has 24 lines, the value of &SCRNROWS is 24.

Two other ACL/E variables are &SCRNCOLS, which represents the number of columns on the screen, and &SCRNSIZE, which represents the total number of characters on the screen (absolute screen size) in bytes. For example, if a screen has 80 columns and 24 rows, the value of &SCRNSIZE is 1920.

UDEFINE Statement

The following UDEFINE statement defines an array variable named DATA:

```
UDEFINE DATA(50),80
```

An array is a variable that can contain many different values, each of which can be stored and retrieved separately. This array has been defined as having 50 elements, meaning that it can contain 50 different values. Each element can contain up to 80 characters. In the variable name, the elements are specified by number. For example, DATA(1) refers to the value in the first element, DATA(2) refers to the value in the second element, etc.

DATA will contain the screen data. The definition of DATA assumes that the screen being copied consists of no more than 50 lines, and that the lines contain no more than 80 characters.

SET Statement

The remainder of the program is a loop that copies lines of data from the screen into &DATA until the last line of the screen is reached.

The following command copies 80 characters of screen data (one row) into an element of DATA. As long as the value of A1 is less than or equal to the number of screen rows, the program will copy rows into the array. For example, assume that the screen contains 24 lines. When the program is completed, the first 24 elements of DATA will contain the screen data. Each element will contain 80 characters.

```
ULOOP       SET DATA(&A1),&SCREEN(&A2,80)
```

Substring Notation

The (&A2,80) part of the following statement is referred to as substring notation. This means that it represents only one part of the total value of the variable &SCREEN. &SCREEN is an ACL/E variable that represents all the data on the screen and &A2 is the value of the accumulator A2. So &SCREEN(&A2,80) represents the 80 characters of screen data starting at position A2.

```
SET DATA(&A1),&SCREEN(&A2,80)
```

In this program, each time the loop is executed, A1 is incremented by 1 and A2 is incremented by 80. So, the first time the loop is executed (A1=1 and A2=1), DATA(1) is assigned a value equal to the 80 characters of screen data starting with position 1 on the screen. The second time the loop is executed (A1=2 and A2=81), DATA (2) is assigned a value equal to the 80 characters of screen data starting with position 81 on the screen, and so on.

ADD Statement

After copying the screen data into an array element, the program uses the following ADD statements to increment A1 by 1 and A2 by 80:

```
ADD A1,1
ADD A2,80
```

These increments prepare the program to copy the next line of the screen into the next array element.

COMPARE and BRANCH Statements

The program uses the following COMPARE and BRANCH statements to make sure that all the lines of the screen have not been copied yet before executing the loop again:

```
COMPARE A1,&SCRNROWS
BRANCH LE,ULOOP
```

The COMPARE statement compares the value of A1 to the total number of rows on the screen (&SCRNROWS). As long as A1 is less than or equal to (LE) the number of rows on the screen, the program will execute the loop again to copy another line of the screen.

ACLPGM Command

After all the lines of the screen are copied, the program schedules the second program with the following ACLPGM command:

```
ACLPGM T2,SCREEN2
```

SCREEN2 is the name of the second program and T2 is the session ID for TSO. (TSO can have a different session ID on your system.) The session specified in the ACLPGM statement (T2 in this case) must be active when the command is executed.

Copy the Data into a Data Set Member

Program SCREEN2, shown below, is scheduled by the first program, SCREEN1. This program is executed in a TSO ISPF session and copies the data from the array created in the first program into a data set member.

```
        OPTION      FLOW,OFF
        OPTION      TERM,ON
        OPTION      MAXI,9999
ELOOP   COMPARE     &SCREEN(31,14),'PRIMARY OPTION'
        BRANCH      EQ,EDIT
        PF          3
        BRANCH      ANY,ELOOP        KEEP BACKING UP
*
EDIT    KEY         '2'              SELECT EDIT
        ENTER                        ENTER
        HOME                         PRESS THE HOME KEY
        TABF        8                TAB TO 'OTHER DSN'
        KEY         '''TPX.USER.PANELS(SCREEN)'''
        ENTER                        GO INTO EDIT
        KEY        'CAPS OFF'        GO TO CAPS OFF
        ENTER
        NL          1                TAB TO FIRST LINE
        KEY         'I'              INSERT A LINE (letter "i")
        ENTER
        TABF        1                PLACE THE CURSOR IN THE FIRST
        TABB        1                POSITION OF THE LINE (AFTER INSERT)
        SET         A1,1             SET COUNTER
*
*EVERYTHING IS SET UP. NOW LOOP THROUGH THE LINES.
*
KLOOP1  KEY         &DATA(&A1)(1,70)) KEY ONE LINE
        ADD         A1,1              NEXT LINE
        COMPARE     A1,&LINECT        ARE WE DONE?
        BRANCH      GT,STOP           YES
        ENTER                         MAKES A NEW LINE
        TABF        1                 PLACE THE CURSOR IN THE FIRST
        TABB        1                 POSITION OF THE LINE (AFTER INSERT)
        BRANCH      ANY,KLOOP1        CONTINUE
*
STOP    UDELETE     LINECT            DELETE LINE COUNT
        UDELETE     DATA              DELETE ARRAY            STOP
```

Explanation of SCREEN2 Program

This program copies the screen data from the array created in the first program to a TSO data set member.

COMPARE and BRANCH Statements

The following COMPARE statement checks to see if you are at the ISPF Primary Option screen by comparing the 14 characters starting at position 31 with the string "PRIMARY OPTION". If you are at the ISPF Option screen, the first BRANCH statement directs the program to the line labeled EDIT, which prepares a TSO data set member to receive the data from the array. If you are not at the ISPF Primary Option screen, the PF statement simulates the PF3 key to back out to the option screen. The second BRANCH statement then directs the program to execute the COMPARE statement again.

```
COMPARE &SCREEN(31,14),'PRIMARY OPTION'
BRANCH EQ,EDIT
PF 3
BRANCH ANY,ELOOP
```

The SEARCH command could have been used here to produce the same result. The difference between the SEARCH and COMPARE statements is that the COMPARE statement will determine if the string is in a particular place on the screen and the SEARCH statement simply determines whether the string is displayed somewhere on the screen.

KEY, ENTER, HOME, and TABF Statements

The following statements open the data set member into which the screen data will be copied:

```
KEY '2'
ENTER
HOME
TABF 8
```

The KEY and ENTER statements type the numeral 2 on the command line of the ISPF Option screen to select the edit option and simulate the Enter key.

The HOME command simulates pressing the HOME key to place the cursor in the first data entry field on the screen. In this case, the first data entry field is the command line.

The TABF statement tells the program to simulate pressing the ®| (forward TAB) key eight times, which places the cursor on the "Other Data Set Name" field.

KEY and ENTER Statements

The following KEY and ENTER statements enter the name of the data set member to which the screen data will be copied:

```
KEY '''TPX.USER.PANELS(SCREEN)'''
ENTER
```

The extra single quotes are necessary because the string itself contains single quotes. The outermost pair of quotes identifies the string to ACL/E, while the two inner pairs of quotes represent the single quotes that TSO expects in the name of the data set member.

KEY, ENTER, NL, TABF, and TABB Statements

The first KEY and ENTER statements shown below turn off the CAPS option in ISPF. The NL statement places the cursor in the command area of the first line in the input area.

The next KEY statement enters the letter "I" in the input area, and the ENTER command simulates the Enter key to make a blank line in the data set.

The TABF and TABB commands simulate pressing the ®| (forward tab) and |¬ (back tab) keys. In this program, the TABF statement moves the cursor forward to the next input field and the TABB statement moves the cursor back to the previous input field. This is done to ensure that the cursor is in the first space of the inserted line.

```
KEY 'CAPS OFF'
ENTER
NL 1
KEY 'I'
ENTER
TABF 1
TABB 1
```

SET Statement

The following SET statement assigns a value of 1 to the accumulator A1:

```
SET A1,1
```

The accumulator is used in this program to determine when all the array elements have been entered into the TSO data set.

KEY Statement

The following KEY statement copies the data from the array element to the current line in the data set member:

```
KEY &DATA(&A1)(1,70)
```

The first time this loop is executed, the program copies the data in the array element &DATA(1), and the second time the loop is executed, it copies the data in &DATA(2), and so on.

Because ISPF displays only 72 columns of text on one line, the program copies only 70 characters of the screen data from each array element. However, ISPF does allow you to scroll right and left with the PF keys, so think about what statements you would need to add to copy all 80 characters of the screen text.

COMPARE and BRANCH Statements

The following COMPARE statement compares the value of A1 to the total number of rows on the screen to determine if all the lines of the screen have been copied:

```
COMPARE A1,&LINECT
BRANCH GT,STOP
```

If A1 is greater than (GT) the number of rows on the screen, the BRANCH statement directs the program to the line labeled STOP, where the variable LINECT is deleted and the program ends.

ENTER, TABF, TABB, and BRANCH Statements

If the previous BRANCH statement did not redirect the program, the following ENTER command simulates pressing the Enter key to add a new line:

```
ENTER
TABF 1
TABB 1
BRANCH ANY,KLOOP1
```

The TABF and TABB commands simulate pressing the ®| (forward tab) and |¬ (back tab) keys. In this program, the TABF statement moves the cursor forward to the next input field and the TABB statement moves the cursor back to the previous input field. This is done to ensure that the cursor is in the first space of the inserted line.

The BRANCH statement directs the program to the statement labeled KLOOP1, which copies the next line of screen data.

Exercise

Create these two programs. Call the first program SCREEN1 and the second one SCREEN2. Before running them, change the TSO session ID in the ACLPGM statement in SCREEN1 from T2 to your TSO session ID, if necessary. Then, change the data set member reference in SCREEN2 to the name of your own data set. Now, run SCREEN1 from an online administration panel by issuing the following command in the ADMIN session:

```
/S SCREEN1
```

Chapter 4: Using Arithmetic and Logic

This chapter discusses the ACL/E commands and variables that are used for arithmetic and logical operations and for conditional actions.

This section contains the following topics:

Setting and Manipulating Accumulators

ACL/E provides the tools necessary to perform arithmetic and logical operations. This chapter describes accumulators and statements that can be used for a variety of purposes, including the execution of loops within an ACL/E program.

ACL/E has four accumulators (A1 through A4), which can be used to represent any numeric value. In an ACL/E program, you can set the value of an accumulator and then increment or decrement it.

Set Value of an Accumulator

To set the value of an accumulator, use the SET command. The format for a SET statement is as follows:

```
SET variable,value
```

where *variable* is the name of the accumulator or other variable and *value* is the value that you want to assign to the variable. For example, the following statement assigns a value of 5 to the accumulator A1:

```
SET A1,5
```

You can also set an accumulator equal to another accumulator. For example, if you have already set A1 to 5, you can set A2 to 5 using the following statement:

```
SET A2,A1
```

In the SET statement, the first operand (which must be a variable) is assigned the value specified in the second operand, whether the value is numeric or another variable.

Increment an Accumulator

To increment an accumulator, use the ADD command. The format for an ADD statement is:

```
ADD accumulator,value
```

where *accumulator* is one of ACL/E's accumulators and *value* is a numeric value to be added to the variable. For example, the following statement increments A3 by 5:

```
ADD A3,5
```

You can also increment an accumulator by the value of another accumulator. For example, the following statement increments the value of A3 by the value of A2:

```
ADD A3,A2
```

The accumulator that is being incremented must be in the first operand position, so the following statement increments A2 by the value of A3:

```
ADD A2,A3
```

Decrement an Accumulator

To decrement an accumulator, use the SUB command. The format for a SUB statement is:

```
SUB accumulator,value
```

where *accumulator* is one of ACL/E's accumulators and *value* is the numeric value to be subtracted from the variable. For example, the following statement decrements the accumulator A3 by 5:

```
SUB A3,5
```

You can also decrement an accumulator by the value of another accumulator. For example, the following statement decrements A3 by the value of A2:

```
SUB A3,A2
```

The accumulator that is being decremented must be in the first operand position, so the following statement decrements A2 by the value of A3:

```
SUB A2,A3
```

COMPARE Statement

The COMPARE command is used to determine the relationship between two operands. You can compare an accumulator with another accumulator, an integer with an accumulator, a variable with a string of characters, and so on. This section illustrates some of the most common uses for the COMPARE statement.

Format

The format for a COMPARE statement is:

```
COMPARE operand1,operand2
```

where *operand1* and *operand2* can be accumulators, integers, strings, screen data, or any other variables.

Compare Values of Two Accumulators

You can use the COMPARE command to compare the values of two accumulators. The following statement compares the value of the accumulator A1 with the value of the accumulator A2:

```
COMPARE A1,A2
```

The result of the comparison is represented by a condition code that indicates the relationship between A1 and A2. The condition code is then set and can be examined by the next statement in the program, a BRANCH statement that uses it to determine where to go next.

In the following program, ACL/E compares the values of accumulators A1 and A2. Because the values are equal, the program branches to the statement labeled EQUAL, which types the message "A1 and A2 are equal." If the values were not equal, the program would execute the statement immediately following the BRANCH statement, which types the message "A1 and A2 are not equal."

```
          SET A1,10
          SET A2,10
          COMPARE A1,A2
          BRANCH EQ,EQUAL
          KEY 'A1 and A2 are not equal.'
          STOP
EQUAL     KEY 'A1 and A2 are equal.'
          STOP
```

Condition Codes Produced by the COMPARE Command

The following condition codes apply to comparisons between accumulators, variables, and strings:

- EQ—equal to
- NE—not equal to
- GT—greater than
- LT—less than
- LE—less than or equal to
- GE—greater than or equal to

An Example of the COMPARE Command

The following statement compares the value of A1 to the value of A2:

```
COMPARE A1,A2
```

The condition code produced by the statement depends on the values held in A1 and A2:

- If the value of A1 is 5, and the value of A2 is 6, the condition code will be LT.
- If the value of A1 is 6, and the value of A2 is 5, the condition will be GT.
- If the value of A1 is 6 and the value of A2 is 6, the condition code will be EQ.

Compare Value of an Accumulator with an Integer

ACL/E can also compare the value of an accumulator with a positive integer. For example, you can rewrite the program that compares the values of accumulators A1 and A2 to compare the value of A1 and the integer 10. The following is an example of such a program:

```
          SET A1,10
          COMPARE A1,10
          BRANCH EQ,EQUAL
          KEY 'A1 does not equal 10.'
          STOP
EQUAL     KEY 'A1 equals 10.'
          STOP
```

DO NOT put single quotes around the accumulator. The COMPARE command determines whether the compare action is numeric or string by checking the syntax of the statement. Single quotes around the accumulator indicate that it should be processed as a string. The COMPARE command does not recognize that the string refers to an accumulator.

For example, if A1 has a value of 3 and A2 has a value of 21, the following COMPARE statement compares the numeric values of the A1 and A2 and sets a condition code of LT:

```
COMPARE A1,A2
```

However, the single quotes in the following COMPARE statement indicate that A1 and A2 are to be compared as strings. As a result, the statement sets a condition code of NE because "A1" is not identical to "A2":

```
COMPARE 'A1','A2'
```

Compare Two Strings

ACL/E can also compare two strings to determine whether they are identical. The following elementary example shows you how a statement comparing two strings looks:

```
COMPARE 'Hello','Goodbye'
```

The result of this comparison is represented by the condition code NE, which indicates that the two strings are not identical.

Compare a Variable and a String

Although you may not find a use for comparisons between two constant strings, you will probably find comparisons between variables and strings very useful. The following program shows a statement that determines whether the user ID entered by a user matches a valid user ID (in this case, USER1). If the user ID is valid, the program continues. If not, the program terminates.

```
          COMPARE '&USERID','USER1'
          BRANCH EQ,MORE
          STOP
MORE      KEY 'Hello.'
          STOP
```

Use Substring Notation to Compare a Variable and a String

You can also use substring notation in variable/string comparisons. For example, the following is a statement that uses substring notation to determine whether the three characters beginning at position 1 of a user ID are "DEV":

```
COMPARE '&USERID(1,3)','DEV'
```

The statement says to begin with the first character position in the user ID and compare the first three characters with the target string "DEV."

Compare Screen Data with a String

In many programs, you will want to know whether the current screen is the one the program will be interacting with. For example, before a program that signs a user on to an application enters the user ID and password, it must determine that the current screen is the application signon screen. To do this, it searches the screen for data that identifies the signon screen.

The command for searching the screen is SEARCH. For example, the following statement determines whether the current screen contains the string "ENTER USERID":

```
SEARCH 'ENTER USERID'
```

The result of the search is represented by a condition code that indicates whether the string is present. The next statement, which is a BRANCH statement, uses this condition code to determine where to go next.

The following program examines the current screen to determine if it contains the string "ENTER USERID." If the string is present, the program branches to the statement labeled TSO, which enters the user ID, then goes to the next statement, which enters the password. If the string is not present, the program executes the statement that immediately follows the BRANCH statement. In this case, the next statement is the STOP statement, which terminates the program.

```
          SEARCH 'ENTER USERID'
          BRANCH EQ,TSO
          STOP
TSO       KEY '&USERID'
          ENTER
          KEY '&PASSWORD'
          ENTER
          STOP
```

Only two condition codes are valid SEARCH results:

EQ

Inidicates that the string is present on the current screen

NE

Inidcates that the string is not present on the current screen.

Search a String for Another String

You can search a string for another string with the SCAN command. For example, the following statement determines if the string contained in the variable &USERID contains the string ADMIN.

```
SCAN '&USERID','ADMIN'
```

Only two condition codes are valid SCAN results:

EQ

Indicates that the string is present in the string that was searched. The variable &LOC is set to the position of the string found within the string that was search.

NE

Indicates that the string is not present in the string that was searched.

The variable &LOC expresses the position as the number of characters from the beginning of the string being searched. For example the following SCAN statement sets &LOC to 3:

```
SCAN 'ENTERDATA','TER'
```

Chapter 5: Working With CA TPX

This chapter shows you how to make your ACL/E programs issue commands, assign automatic signon and signoff programs for applications, and use the message log.

This section contains the following topics:

Issue Commands

An ACL/E program can issue any command or sequence of commands so that you can return to the menu, jump to another session, start a new session, or lock the screen to force a user to re-enter the password.

A COMMAND statement is used to issue a command from an ACL/E program, such as jumping to the next session or displaying the menu. You can use COMMAND statements to terminate sessions, sign users off, and log terminals off. The following sections discuss alternative ACL/E commands you can use to perform these functions.

The format for a COMMAND statement is:

```
COMMAND 'command string'
```

where '*command string*' is the command you want to execute. If the command string contains spaces or punctuation, the string must be enclosed in single quotes. If the command string does not contain spaces or punctuation, the single quotes are optional.

Examples

Here are some examples of how you might use a COMMAND statement in an ACL/E program.

The following statement displays the menu:

```
COMMAND 'W'
```

This statement is used to "jump" to the next session:

```
COMMAND 'J'
```

This statement switches to the session with the session ID of T2:

```
COMMAND 'T2'
```

This statement locks the terminal:

```
COMMAND 'L'
```

This statement activates the TPXOPER session:

```
COMMAND 'A TPXOPER'
```

Note: The COMMAND statement schedules the command to execute. As soon as the command is scheduled to execute, the program continues with the next statement. Execution continues in the session in which the program was started, even after a COMMAND 'J' statement, which simulates the JUMP key.

Terminate a Session

Because session termination is one of the most common uses of ACL/E programs, ACL/E provides the following command to terminate a session:

```
TERMSESS
```

You can also use the following COMMAND statement to terminate a session:

```
COMMAND 'I'
```

This command will activate any signoff program for the session.

Sign a User Off CA TPX

ACL/E provides the following command to sign a user off:

```
SIGNOFF
```

You can also use the following COMMAND statement to sign a user off:

```
COMMAND 'F'
```

You cannot always use these two statements interchangeably. For example, if you have specified in a user's definition (in the "Inactivate On" field) that CA TPX terminate the user's sessions when the /F command is issued, you would not want to use the SIGNOFF command to sign that user off. If you did, the user's sessions would remain active since the /F command was not issued. In this case, you would use COMMAND 'F', which simulates the /F command and inactivates the sessions as specified in the user's definition.

Log a User's Terminal off CA TPX

ACL/E provides the following command to log off a user's terminal:

```
LOGOFF
```

You can also use the following COMMAND statement to log off a user's terminal:

```
COMMAND 'K'
```

You cannot always use these two statements interchangeably. For example, if you have specified in a user's definition (in the "Inactivate On" field) that CA TPX terminate the user's sessions when the /K command is issued, you would not want to use the LOGOFF command to log a user's terminal off CA TPX. If you did, the user's sessions would remain active since the /K command was not issued. In this case, you would use COMMAND 'K', which simulates the /K command and inactivates the sessions as specified in the user's definition.

Execute an ACL/E Program from an ACL/E Program

You can schedule another ACL/E program to execute from an ACL/E program. The scheduled program can execute in the current session or in another active session. When the scheduled program executes in the current session, the main program is suspended until the scheduled program completes execution. Variables from the main program are available to the scheduled program.

ACLPGM Command

The ACLPGM command schedules another ACL/E program. For example, the following statement calls a program named NEWPRG1 to execute in the current session:

```
ACLPGM NEWPRG1
```

To make the scheduled program execute in an active session other than the current session, use this format:

```
ACLPGM sessID,progname
```

where *sessID* is the ID for an active session in which you want to execute the program and *progname* is the name of the ACL/E program. For example, this statement schedules a program named NEWPRG1 to execute in the session named T2:

```
ACLPGM T2,NEWPRG1
```

Note: The ACLPGM command does not suspend execution of the main program when a program is scheduled to run in another session. For information about suspending execution of an ACL/E program while the scheduled program is executing, see Waiting for Notification from Scheduled Programs.

Set Up Automatic Startup and Termination Programs

A startup ACL/E program can take control automatically when a session is activated. The startup program can sign the user on to the application session, and perform application setup, such as taking the user to a particular screen. You must identify the program as the startup program by specifying its name in the "Startup ACL" field of the user's application session definition.

Bypass the Startup Program from the TPX Menu

The user can bypass the startup ACL/E program by activating the session using the /G command from the TPX Menu.

Termination Programs

Similarly, a termination ACL/E program can take control when a session is inactivated with the /I command or the TPXOPER CANCEL command. The termination program will sign the user off the session. You must identify the program as the termination program by specifying its name in the "TermACL" field of the user's application session definition.

Record Executed Commands in the Message Log

You can record the sequence of an ACL/E program's executed commands in the message log. In z/OS, your message log is identified by the LOG DD statement in your startup procedure. In VM, the message log is identified by the LOG FILEDEF statement in your startup exec.

Record Program Execution

To record all of the program's execution, include the following statement at the beginning of the program:

```
OPTION FLOW,ON
```

Turn Off Recording

To turn recording off, use the following form of the statement:

```
OPTION FLOW,OFF
```

The default value for this command is OFF.

Note: You can start or stop recording to the message log at any point in the program.

Write a Message to the Log

ACL/E programs can write messages to the message log at any time during execution. In z/OS, your message log is identified by the LOG DD statement in your startup procedure. In VM, the message log is identified by the LOG FILEDEF statement in your startup exec.

Specify Message Text

Messages written to the Message Log are helpful in debugging ACL/E programs. To specify the message text (up to 60 characters long), issue the MSG command. For example, the message "USER1 linked to CICSA" will be written to the message log each time the user starts a CICS session if you include the following statement in a CICS startup program:

```
MSG '&USERID linked to CICSA.'
```

Display Application Data at the Terminal

You can specify whether application data should be displayed on the terminal screen by using the OPTION command with the TERM operand. If you want to show the user what the ACL/E program and the application are typing on the screen, include the following statement in your ACL/E program:

```
OPTION TERM,ON
```

Stop Displaying Data

To stop showing the interactions between the ACL/E program and the application, include the following statement in your ACL/E program:

```
OPTION TERM,OFF
```

The default value for this command is OFF.

Note: You can start or stop displaying data on the terminal screen at any point in the program.

Redirect an Idle ACL/E Program

You may want an ACL/E program to branch to a statement with a specified label if no activity occurs for a specified number of seconds. To branch to the statement labeled RESUME after 60 seconds of inactivity, use the following statement:

```
OPTION TIME=60,RESUME
```

If the time out is executed, it is not reset.

Remove Time Restriction

You can remove the time restriction with the following statement:

```
OPTION TIME,OFF
```

Prevent Uncontrolled Loops

You can guard against uncontrolled looping in an ACL/E program by specifying the maximum number of instructions the program is allowed to execute. When the maximum is reached, the program will terminate. The range of values is 0 to 64,000. Use 0 to specify no limit. The default value is 50.

Examples

To terminate a program after 100 instructions are executed, use the following statement:

```
OPTION MAXI,100
```

To indicate that there is no limit on the number of instructions that can be executed, use the following statement:

```
OPTION MAXI,0
```

Set a Runaway Limit

Uncontrolled loops are also prevented by the runaway limit specified in CA TPX Administration. The runaway limit is an additional safeguard against uncontrolled loops, and cannot be disabled or changed by an ACL/E statement such as OPTION MAXI.

Certain ACL/E commands relinquish control of the program to the user, terminal, or application. Examples of commands that relinquish control are ENTER, PF, PA, INPUT, SEND, PAUSE, WAIT, and SUSPEND. Other ACL/E commands, such as ADD, KEY, TABF, and VDEFINE, do not relinquish control because they do not require any action from the user, terminal, or application.

An OPTION MAXI sets the total number of ACL/E statements that CA TPX will process before terminating the ACL/E program. A counter is updated each time a CA TPX executes a statement. When the counter reaches the specified value, the program terminates. The counter is updated regardless of whether the statement relinquishes control of the program.

The runaway limit, however, is the number of consecutive non-relinquishing statements that CA TPX will process before terminating the program. The counter for the runaway statement limit is updated each time a non-relinquishing ACL/E statement executes, and is reset to zero whenever an ACL/E statement that relinquishes control executes. If the value of this counter reaches the runaway limit, this product terminates the ACL/E program and writes the following message to the Log (if OPTION FLOW,ON is specified):

```
RUNAWAY CONDITION OCCURRED
```

Make an ACL/E Program Memory Resident

When an ACL/E program terminates, it is removed from active memory. You can improve performance of an ACL/E program that is used frequently by keeping it resident in active memory so that it does not have to be loaded from the library each time it is used. To do this, include the following statement:

```
OPTION RESIDENT,ON
```

Reload a Program

If you change an ACL/E program while it is memory resident, the change will not take effect until the program is reloaded.

To reload an ACL/E program from an operator's session, type the following statement:

```
RELOAD ACL=progname
```

To reload an ACL/E program from a system operator's console, type the following statement:

```
F TPX,RELOAD ACL=progname
```

where *progname* is the name of the ACL/E program you want to reload.

Specify ACL/E Information in Administration

If an ACL/E program is to be executed when a user starts or terminates a session, you must identify it in either a global or user-level application session definition. In addition, you can provide values for ACL/E variables in user-level application session definitions.

Specify Startup and Termination ACL/E Programs

You specify the name of an application's default startup and termination ACL/E programs in the "Start ACLPGM" and "Termination ACLPGM" fields of the Application Characteristics Table entry.

At the user and profile levels, you specify the names of an application session's startup ACL/E and termination ACL/E in the "Startup ACL" and "Term ACL" fields of the user's application session definition.

Specify ACL/E Variable Values

An ACL/E program can obtain values from a user's application session definition. The variables &SID and &APPL get their values from the "Session ID" and "Applid" fields. If the user's session user ID and password are different from the ones used for signing on to CA TPX, you specify them in the "ACL Userid" and "ACL Password" fields, respectively.

The ACL/E variables &P1 through &P8 get their values from the "Parm 1" through "Parm 8" fields in session options for a user.

Chapter 6: Interacting With Applications

This chapter discusses the ACL/E commands and variables you can use to make an ACL/E program interact with an application session in the same way that a user would.

This section contains the following topics:

Identify Application Sessions

ACL/E provides a variety of commands and variables that allow ACL/E programs to interact with an application session. You can use these commands and variables to identify the session by application ID or session ID, to send information to the application, and to wait for information from the session. Other commands described in this chapter allow ACL/E programs to wait for input from the user.

ACL/E provides variables that represent the VTAM name (application ID) or the session ID of the session in which an ACL/E program is running.

VTAM Name Variable

The variable that represents the VTAM name of the session in which the program is running is:

```
&APPL
```

Session ID Variable

The variable that represents the session ID of the session in which the program is running is:

```
&SID
```

Describe Application Sessions

ACL/E provides variables that can describe an application session. In these variables, you specify the name of the application or session. When variable substitution takes place, the variable returns a string of information about the application or session.

Session Variable

The variable that represents information about a session is:

`&SESS_INF(`*`sessionID`*`)`

This variable returns the following information about the session:

- If the session is active
- If the session is the current session
- If the session is not a valid session ID
- The application running in the session
- The name of the ACL/E program running in the session (if any)

Application Variable

The variable that represents information about an application is:

`&APPL_INF(`*`applicationID`*`)`

This variable returns the following information about the application:

- If the application is active
- If the application is in a quiesced state
- If the application is not defined in the Application Control Table (ACT)
- If the application is an internal application

Send Data to the Application

ACL/E programs can send data directly to an application by simulating action keys. ACL/E programs can simulate action keys such as Enter, PA*n*, PF*n*, and CLEAR. When an action key statement is executed, program execution is suspended until ACL/E receives a response from the application.

Clear the Screen

To simulate the pressing of the Clear key, which clears the screen, use:

```
CLEAR
```

Transmit Screen Data with Enter Key

To simulate the pressing of the Enter key, which transmits screen input data to the application, use:

```
ENTER
```

Transmit Screen Data with a PF Key

To simulate the pressing of a PF key, which transmits all screen input data back to the application, use the PF command. For example, to simulate pressing PF3, use:

```
PF 3
```

Simulate a PA Key

To simulate the pressing of a PA key, use the PA command. For example, to simulate pressing PA2, use:

```
PA 2
```

Receive Data from the Application

Some applications allow data to be entered from the keyboard before it sends the appropriate information to the terminal screen. In these cases, the next statement in an ACL/E program could be executed before the data it needs is transmitted from the application. The following commands instruct your ACL/E program to wait for the appropriate information to be transmitted from the application.

Suspend Action Temporarily

You can temporarily suspend ACL/E program execution for a specified number of seconds with the WAIT command. For example, the following statement suspends execution for five seconds:

```
WAIT 5
```

Output from the application can continue to be displayed on the screen during the "wait" period.

Wait for Data from the Application

You can also tell the program to wait for data from the application by including the following command:

```
WINP
```

You need to use this command only if output is going to come from the application before any action key statement (ENTER, CLEAR, PA, or PF) is executed.

You use the WINP statement primarily in startup ACL/E programs to ensure that these programs wait for the application's first screen or good morning message.

You can also use the WINP statement for applications that constantly generate output, such as CA-Remote and VM. In these cases, you may want to search the screen image each time the application sends data, and if it is not what you are looking for, use the WINP command to receive the next application data.

Problems with the WINP Command

Two problems can occur with the WINP command.

- If the application sends no data, the ACL/E program will go into an infinite wait.
- If you use WINP to wait for data to be displayed and SEARCH to examine the data after it has been displayed, it is possible for the program to go into an endless loop. For example, a program might have code similar to the following:

```
LOOP1   WINP             WAIT FOR DATA FROM APPLICATION
        SEARCH 'Ready'   IS THE READY PROMPT DISPLAYED
        BRANCH NE,LOOP1  NO, WAIT LONGER
```

If the string "Ready" is never sent from the application, the program will go into an endless loop waiting for it to be displayed. Even if the application sends data that the program does not use, the WINP command will tell the program to continue execution. Unless you take measures to prevent an endless loop, the program can go into one if unexpected data is sent from the application.

Prevent an Infinite Wait

To prevent an infinite wait, specify a time-out period using the OPTION command with the TIME operand. For example, to tell the program to branch to the statement labeled RESUME after 20 seconds of inactivity, use the following statement:

```
OPTION TIME=20,RESUME
```

Prevent an Endless Loop

There are two ways that you can prevent an endless loop:

1. You can use an accumulator to count the number of times the program executes the loop. Each time the loop is executed, increment the accumulator and compare it to a value (such as 10). Then instruct the program to branch to the end of the program when the accumulator reaches the specified value.
2. You can place an OPTION statement at the beginning of the program and set the maximum number of instructions that the program can execute. For more details on using the OPTION statement to prevent endless loops, see Prevent Uncontrolled Loops (see page 62).

Wait for Specific Data from Application

The SRCHWAIT command combines the features of the SEARCH and WINP commands. You can use the SRCHWAIT command in environments such as VM and CA-Remote, where unsolicited output arrives from the application. For example, to determine whether a VM session is displaying the "Ready" prompt, use the following statement:

```
SRCHWAIT 'Ready'
```

If the data is found or if the application is sending no more data, SRCHWAIT sets a condition code (EQ or NE) and the next statement in the program is executed.

Note: ACL/E determines whether the application will be sending more data by examining the session's SNA Bracket and Change Direction state. If the application session is "IN-BRACKET" and the application has given the "change direction indicator," ACL/E concludes that the application will send no more data.

Receive Data from the User

ACL/E also provides commands that allow you to create an ACL/E program that can receive input from the user.

Wait for Input

The INPUT command tells an ACL/E program to wait for input from the user. This command suspends all ACL/E operation until the program receives the input. Data entered at the INPUT statement can then be used by a SEARCHN or SEND command.

Suspend the Program

You can use the PAUSE command to suspend execution of a program until the user presses an action key (Enter, Clear, PA, and PF) at the terminal. When you do this, the user can select from a menu or enter data on the application screen before the next ACL/E program statement is executed. The user action is transmitted to the application, and execution resumes when the application responds to the user's input.

In the following program, the PAUSE command tells ACL/E to wait for the user to type a CICS password and press the Enter key:

```
       KEY 'LOGON CICS'           Key 'LOGON CICS.'
       ENTER                      Enter it.
       SEARCH 'ENTER PASSWORD'    See if CICS is responding.
       BRANCH NE,END              If not, quit.
       PAUSE                      If CICS prompts with
*                                 'ENTER PASSWORD,'
*                                 allow user to enter the
*                                 password.
END    STOP
```

Examine User Input

You may want an ACL/E program to examine the user's input before submitting it to the application. After an ACL/E program receives input from the user with the INPUT command, the SEARCHN statement can be used to examine the newly entered data. After the data is examined, use a BRANCH statement to determine whether the data should be sent to the application. The SEND command is used to send the data to the application.

For example, the following statements determine whether the user typed the string "HELLO, TPX." If so, the program sends the data entered by the user to the application; otherwise, the program terminates.

```
          INPUT
          SEARCHN 'HELLO, TPX'
          BRANCH EQ,GO
          STOP
GO        SEND
          STOP
```

Examine the Last Action Key Pressed

The last action key the user pressed is represented by the variable &TAID. You can use &TAID in a COMPARE statement to determine what the program should do based on what the user entered. The following example shows a simple program that redirects to a statement labeled SYS1 if the user pressed PF1, or to a line labeled SYS2 if the user pressed PF2.

```
DISPLAY MNU001
INPUT
COMPARE '&TAID','PF1'
BRANCH EQ,SYS1
COMPARE '&TAID','PF2'
BRANCH EQ,SYS2
```

Issue an Attention Interrupt

The following command issues an attention interrupt to the application:

```
ATTN
```

Note: The attention interrupt always goes to the application no matter how the Attn key is defined in your System Options Table.

Terminate the Application Session

The following command terminates the application session in which the ACL/E program is running:

```
TERMSESS
```

For instance, you would use this command to terminate an ACL/E logoff program when unexpected conditions arise.

Control Application Sessions with OPENGATE

If your installation uses OPENGATE to perform session setup and cleanup, you can set up control ACL/E for use by OPENGATE. Error messages issued by the control ACL/E have the prefix IENM. For more information, see the *Programming Guide*.

Chapter 7: Working With Data on the Screen

This chapter shows you how to use ACL/E commands to see what is displayed on the current screen. This chapter also shows you how to move the cursor around on the screen.

This section contains the following topics:

Search for Data on the Current Screen

ACL/E provides a number of commands and variables that allow you to examine and manipulate the data that appears on the application screen.

ACL/E can search for specific data displayed on the current screen. This search capability enables a program to ensure that the correct screen is displayed before it continues.

Search for a Character String

The statements from a TSO startup ACL/E program shown below search for the string "ENTER USERID" to determine whether the TSO signon screen is displayed. If the correct screen is displayed, the program enters the user ID and password or password phrase. If the correct screen is not displayed, the program terminates.

```
          SEARCH 'ENTER USERID '
          BRANCH EQ,TSO
          STOP
TSO       KEY '&USERID '
          ENTER
          KEY '&PSWD' or '&PWPH'
          ENTER
          STOP
```

The SEARCH command searches the screen image for the specified string. If the string is found, the SEARCH statement sets a condition code of EQ, which is used by the BRANCH statement. If the string is not found (NE), the program terminates.

Search for a Hexadecimal String

If the data you are searching for contains special characters that are not on your keyboard, you can search for them by specifying the hexadecimal value of the string (in single quotes) followed by the suffix X in the SEARCH statement. The following statement searches for the string "We love TPX!":

```
SEARCH 'E685409396A58540E3D7E75A'X
```

Note: When you use the X suffix, you must enclose the string in single quotes.

Search for Any Data

At times you might want an ACL/E program to determine whether the screen contains any data at all before proceeding to the next step. To do this, use the SEARCH command with the ANY operand. For example, the following statements use the SEARCH command to determine if the screen contains any non-null or non-blank data and prints a message if it is blank.

```
          SEARCH ANY
          BRANCH NE,BLANK
          STOP
BLANK     KEY 'This screen is blank'
          STOP
```

Use Variables to Represent Screen Data

You can use ACL/E's predefined screen variables to represent the screen and the data on the screen. ACL/E looks at the screen as a grid of horizontal rows and vertical columns. Variables describe the screen size, screen data, and cursor position in terms of rows, columns, and row/column coordinates.

Represent Screen Size

ACL/E variables identify the number of columns on the screen, the number of rows on the screen, and the absolute screen size.

Representing the Number of Columns

The variable &SCRNCOLS represents the number of columns on the screen. For example, if the screen is 80-columns wide, the value of &SCRNCOLS is 80.

Representing the Number of Rows

The variable &SCRNROWS represents the number of rows on the screen. For example, if the screen is 24 rows long, the value of &SCRNROWS is 24.

Representing the Absolute Screen Size

The variable &SCRNSIZE represents the absolute size of the screen; that is, the total number of coordinates or character positions on the screen.

ACL/E determines the absolute screen size, in bytes, by counting each character position on the screen. For example, in an 80-column by 24-row screen, the first row contains coordinates 1 through 80, the second row contains coordinates 81 through 160, and so on, down to row 24, which contains coordinates 1841 through 1920. The value of &SCRNSIZE would be 1920.

Represent Screen Data

ACL/E provides variables representing all data on the screen, data locations, and the data that appears at a particular location. You represent the entire screen image (that is, all data on the screen) with the variable &SCREEN.

You can use substring notation with this variable if you want to represent only a portion of the screen data. For example, to represent the ten characters beginning at position 5 on the screen, you would use the following substring notation:

```
&SCREEN(5,10)
```

Represent Location of the Screen Data

The variable &LOC represents the position of the screen data that is found as a result of a SEARCH statement.

For example, suppose the following statement is used in an ACL/E program:

```
SEARCH 'ENTER USERID'
```

If the statement finds the target string, the variable &LOC will contain the value of the screen position of the first character of the target string.

ACL/E also has variables that represent the column and row of the target string. These are &LOCCOL and &LOCROW. So if the string "ENTER USERID" starts in column 5 of row 2, the value of &LOC is 85 (80 in the first row and 5 in the second), the value of &LOCCOL is 5, and the value of &LOCROW is 2.

The data from the position indicated in &LOC to the end of the field that contains the search data is &SCRNFLD. For example, the user ID field on an application screen may contain more than just the string "ENTER USERID". It may also contain other characters so that the entire field looks something like this:

```
ENTER USERID ====>
```

In this case, the value of &SCRNFLD would be all of the characters from the first character in the field (the first letter "E") to the last character in the field (the character ">").

Search the Current Field for a String

You can look for data from &LOC to the end of the field using the SEARCHF statement. For example, the following statement determines whether the current field (represented by the variable &SCRNFLD) contains the string "HELLO USER":

```
SEARCHF 'HELLO USER'
```

If the field contained the string "ENTER USERID" instead of "HELLO USER", this statement would return a condition code of NE.

Locate the Current Cursor Position

ACL/E provides variables that represent the row and column of the cursor position. In addition, there is a variable that represents the cursor position in respect to the absolute screen size.

Locate the Cursor by Row

The variable &CSRROW represents the row of the current position of the cursor. For example, if the cursor is in row 2, the value of &CSRROW is 2.

Locate the Cursor by Column

The variable &CSRCOL represents the column of the current position of the cursor. For example, if the cursor is in column 5, the value of &CSRCOL is 5.

Locate the Cursor on the Total Screen

The variable &CURSOR represents the location of the cursor with respect to the total screen size. For example, if the screen width is 80 columns, and the cursor is in column 5 of row 2, the value of &CURSOR is 85.

Move the Cursor

There are a number of ways in which you can change the position of the cursor. Below are descriptions of the various commands you can use to move the cursor around the screen.

Move Cursor Up, Down, Left, and Right

You can move the cursor in any direction using directional commands. Note that each of these commands requires an operand indicating the number of rows or columns the cursor is to be moved.

Moving the Cursor Up

To move the cursor up a specified number of rows (lines), use the UP command. For example, the following statement moves the cursor up five lines:

```
UP 5
```

The UP command simulates pressing the up-arrow key. If the cursor reaches the top of the screen, it wraps to the bottom and continues upward from there.

Moving the Cursor Down

To move the cursor down a specified number of rows (lines), use the DOWN command. For example, the following statement moves the cursor down five lines:

```
DOWN 5
```

The DOWN command simulates pressing the down-arrow key. If the cursor reaches the bottom of the screen, it wraps to the top and continues downward from there.

Moving the Cursor Left

To move the cursor left a specified number of columns, use the LEFT command. For example, the following statement moves the cursor left five columns:

```
LEFT 5
```

The LEFT command simulates pressing the left-arrow key. If the cursor reaches the left side of the screen, it wraps to the right side and continues left from there.

Moving the Cursor Right

To move the cursor right a specified number of columns, use the RIGHT command. For example, the following statement moves the cursor right five columns:

```
RIGHT 5
```

The RIGHT command simulates pressing the right-arrow key. If the cursor reaches the right side of the screen, it wraps to the left side and continues right from there.

Set the Cursor Position

You can position the cursor at a designated screen location using the variable CURSOR in a SET statement. For example, to move the cursor to position 10 (row 1, column 10), use:

```
SET CURSOR,10
```

Simulate the HOME Key

The HOME command simulates the pressing of the HOME key and moves the cursor directly to the first input field on the screen.

```
HOME
```

Simulate the Return Key

The NL command moves the cursor to the first input field on the next line on the screen. This command simulates pressing the return key on your keyboard. You can specify how many lines down you want to move the cursor. For example, the following statement moves the cursor to the first input field on the fifth row (line) down from the current position:

```
NL 5
```

Simulate the Forward Tab Key

The TABF command simulates pressing the ®| (forward tab) key and moves the cursor to the next input field. You can specify how many input fields forward you want to move the cursor. For example, the following statement moves the cursor ahead five input fields:

```
TABF 5
```

The TABF command is ignored if the 3270 keyboard has automatic cursor skipping, in which case the cursor would automatically move to the next field.

The TABB command simulates pressing the back tab key and moves the cursor to the previous input field. You can specify how many input fields back you want to move the cursor. For example, the following statement moves the cursor back five input fields:

```
TABB 5
```

Note: If the cursor is in the first position of an input field, or anywhere in an output field, it will be moved backward by five input fields. Otherwise, it will be moved to the first position of the current input field, then moved backward four input fields. In either case, the cursor will always occupy position 1 of the target input field.

The TABL command moves the cursor to the first position of the last input field on the screen.

Type and Delete Data on the Screen

An ACL/E program can type and delete characters on the screen just as you do from a 3270 keyboard.

Type Data on the Screen

To type data on the screen, you must first use a statement that moves the cursor to where you want the data to be typed. Then use a KEY statement to type the data onto the screen. The data to be typed must be enclosed in quotes if it contains spaces or punctuation. For example, the following statement types the string "Good Morning":

```
KEY 'Good Morning'
```

You can type characters that cannot be typed on the keyboard by using hexadecimal codes as values for the string variable followed by the suffix X; for example, the following statement types the string "We love TPX!":

```
KEY 'E685409396A58540E3D7E75A'X
```

Note: When you use the X suffix, you must enclose the string in single quotes.

Switch Into and Out of Insert Mode

After the program has typed the data on the screen, you may want to insert characters between the ones already typed. To do this, use the INSERT command. This command, which simulates pressing the INSERT key, places the virtual terminal (not the physical terminal) in INSERT mode. To take the virtual terminal out of INSERT mode, use the RESET command. For example, the following program segment inserts the string "to be" at the current cursor position and then resets the virtual terminal.

```
INSERT
KEY 'to be'
RESET
```

Delete Data from the Screen

To delete data from the screen, you must first use a statement that moves the cursor to the data to be deleted. Then use one of the deletion statements described here, depending on the amount of data you want to delete.

Delete the Character at the Cursor Position

The DELETE command simulates pressing the DELETE key on the keyboard. Each DELETE statement deletes the character at the cursor position. After the character is deleted, the characters to the right of the cursor position shift one space to the left. Note that the DELETE command does not have an operand. If you want to delete multiple characters, you have to repeat the command or use a loop. For example, both of the following program segments delete five characters to the right of the cursor position.

A Program Repeating the Command

```
DELETE
DELETE
DELETE
DELETE
DELETE
```

A Program Using a Loop

```
          SET A1,0
DELETE    DELETE
          ADD A1,1
          COMPARE A1,5
          BRANCH LT,DELETE
          STOP
```

Delete Data from Cursor Position to End of the Field

The EOF command simulates pressing the EOF key and deletes all data from the cursor position to the end of the field. All characters from the cursor to the end of the field are replaced with null characters.

Replace Data in Unprotected Fields with Nulls

The ERASE command erases all data in unprotected fields on the screen. This command simulates pressing the ERASE or ERASE INPUT key and replaces all unprotected data on the screen with null characters.

Chapter 8: Defining and Manipulating Variables

This chapter explains the different kinds of variables and how they are used in ACL/E statements. This chapter also shows you how to define your own variables to use in ACL/E programs.

This section contains the following topics:

Variable Names

Variables are used to indicate items whose values will be determined when the program is executed. In addition to having predefined variables, ACL/E allows you to define your own variables.

ACL/E treats a string preceded by an ampersand (&) and ending with a delimiter (such as a space or period) as a variable name. If the ampersand is followed immediately by a blank character or period, then it is treated as the ampersand character (&).

ACL/E treats the eight characters immediately after the ampersand as the variable name unless it encounters a space, period, comma, slash, left parenthesis, quote, or another ampersand.

The concatenation character (a period) is used to produce one value that is the combination of the two or more variables or text strings.

Variable Concatenation

ACL/E works from left to right when resolving the value of a concatenated variable. If, however, a period is used to concatenate two values to form the name of a variable, ACL/E evaluates the statement from right to left, instead of left to right.

Example

Assume that the following variables have these values:

- &A—TOUCH
- &B—DOWN
- &C—B
- &AB—FIELDGOAL

The following table shows you how ACL/E will determine the value of the variables specified:

Variable	What ACL/E does	Resulting Value
&A	Uses the value of &A	TOUCH
&.A	Concatenates the ampersand character (&) with the letter "A" **Note:** When a period is used to concatenate two values to form the name of a variable, ACL/E evaluates the statement from right to left, instead of left to right.	&A
&A.TONE	Concatenates the value of &A with the character string "TONE"	TOUCHTONE
&A&B	Concatenates the value of &A with the value of &B	TOUCHDOWN
&A.&C	Concatenates the variable name &A with the value of &C (which is the letter B). **Note:** When a period is used to concatenate two values to form the name of a variable, ACL/E evaluates the statement from right to left, instead of left to right.	FIELDGOAL

Substring Notation

Substring notation can be applied to any variable. The format is:

```
&varname(pp,nn)
```

where *&varname* is the variable name, *pp* is the starting position (ranging from 1 to the number of characters in the string) and *nn* is the number of characters to be used.

Example

The following substring notation retrieves the first three digits of the value of &USERID:

```
&USERID(1,3)
```

Substring Values

If you specify a starting position that is greater than the length of the string, ACL/E will return a null value. Also, if the number of characters you specified is greater than the number of characters after the starting position, ACL/E will truncate at the end of the string.

Example

Assume that the following variables have these values:

&A—TOUCH

&B—DOWN

&C—B

&AB—FIELDGOAL

&D(1)—EXTRA

&D(2)—POINT

The following table shows you how ACL/E will determine the value of the variables specified with substring notation:

Variable	What ACL/E does	Resulting Value
&A(1,2)	Returns the value of the two characters in &A starting with the position 1	TO
&A(2,8)	Returns the value of the eight characters in &A starting with position 2, but only returns four characters since there are only four characters to the right of position 2	OUCH
&C&B(2,2)	Returns the value of &C (which is the letter "B") followed by the value of the two characters in &B starting with position 2	BOW
&C(2,3)	Since the value of &C is only one character long, this statement returns no value	null substitution

&D(1)(3,2)	Returns the value of the two characters in the third and fourth positions of the first element of &D	TR
&D(1)(2,1)&D(2)	Returns the second character in the first element of &D and the contents of &D(2)	XPOINT

Predefined Variables

ACL/E provides predefined variables that can be used to represent values such as user input, the current time, and the current date.

For a description of ACL/E's predefined variables, see the appendix Predefined Variables (see page 157).

User-defined Variables

User-defined variables are variables you can define to represent values that you need to run a specific program. After you have defined a variable, you can assign a value to it with the SET command. There are three types of user-defined variables:

- **Persistent variables** exist as long as the user has an active CA TPX session. They are accessible from any ACL/E program in any session. You define persistent variables with the UDEFINE command and delete them with the UDELETE command.
- **Transient variables** exist only for the life of the ACL/E program in which they are defined. They are accessible only from the current ACL/E program and its subroutines. You define transient variables with the VDEFINE command and delete them with the VDELETE command.
- **Global variables** are defined as CA TPX global variables. They are accessible by any ACL/E program and remain until they are deleted by a program. You define global variables with the GDEFINE command and delete them with the GDELETE command.

Characteristics

In general, user-defined variables have the following characteristics:

- User-defined variable names can be up to eight characters in length, not including the ampersand symbol (&); for example, &VARIABLE.
- User-defined variables are resolved to mixed-case character data that is left-justified and stripped of trailing blanks.
- If an undefined variable name appears as the first operand of a SET command, it is automatically defined as an uppercase transient variable.
- If an undefined variable name appears within a quoted operand, it is resolved to a null string.
- Values of transient variables are passed to any subroutine programs generated by the ACLPGM command.

Define Variables

You can define your own variables to represent values in an ACL/E program by using the UDEFINE, VDEFINE, and GDEFINE commands. The DEFINE commands each have the following format:

*x*DEFINE '*varname*',*length*,TYPE=*type*

Operand Explanations

The DEFINE commands have the following operands:

Operand	Explanation
x	Specifies either U, V, or G, indicating what kind of variable is being defined.
varname	Specifies the name of the variable.
length	Specifies the length (in bytes) of the text string or numeric value that will be assigned to the variable. If you do not specify a length, ACL/E uses the default of eight (8).

TYPE=*type*	Specifies the type of variable, which can be: CHAR - Mixed case characters. UPCHAR - Uppercase characters. NUM - Numeric characters. BIN - Binary characters. HEX - Hexadecimal characters. When you specify either a CHAR or UPCHAR variable, you can also specify NOTRUNC, which specifies that trailing blanks in the variable will not be truncated when the variable is resolved.

Example

To define a transient variable that will consist entirely of uppercase characters, contain up to 10 characters and trailing blanks, use the following command:

```
VDEFINE CUSTNAME,10,TYPE=UPCHAR,NOTRUNC
```

Usage Notes

The following usage notes apply:

1. If you use two VDEFINE, UDEFINE or GDEFINE statements in a program to define the same variable name, the second statement will delete the first variable before defining it a second time. As a result, any value assigned to the variable will be lost.
2. If you use a VDEFINE statement and a UDEFINE statement to define the same variable name in a program, a SET statement for that variable will update only the variable defined with VDEFINE. As a result, you will not be able to pass the value of the variable to another ACL/E program.

Important! Be careful when defining global variables with GDEFINE. These variables will persist after the ACL/E program has executed and will not be deleted except by an explicit GDELETE statement.

Delete Variables

When you use UDEFINE, VDEFINE, or GDEFINE to define a variable, you should always be sure to delete the variable when you no longer need the value.

Use the appropriate DELETE command to delete the variable (UDELETE, VDELETE, or GDELETE).

Important! Be careful to delete variables defined with UDEFINE or GDEFINE, because these variables will persist after the ACL/E program finishes executing.

For example, the following statement deletes the variable defined in the example in the previous section:

```
VDELETE CUSTNAME
```

Chapter 9: Displaying an Alternate Screen Image

This chapter shows you the ACL/E commands and statements used to define and display alternate screen images.

ACL/E offers two ways to display alternate screen images:

- Overlay an application screen with one or more lines of data specified in an ACL/E program.
- Display a screen image generated from a panel definition.

This section contains the following topics:

Overlay the Current Screen

To overlay the current screen image with a single string of data, you use the OUTPUT command. Each time a new OUTPUT statement is encountered, the screen is refreshed so that the data displayed by the previous statement disappears.

Format of the OUTPUT Command

The OUTPUT command has the following format:

```
OUTPUT 'text string',r/c
```

where '*text string*' is the text you want to display on the screen, *r* is the row you want the data to be displayed on, and *c* is the column in which the text will begin.

The following format is also acceptable:

```
OUTPUT 'text string',p
```

where *p* is the absolute screen position of the starting point.

Example of the OUTPUT Command

Either of the following statements will display the string "Start here" starting in row 2, column 1 of an 80-column screen:

```
OUTPUT 'Start here',2/1
```

or

```
OUTPUT 'Start here',81
```

Note: If you specify neither *r/c* nor *p*, the data will be displayed starting at the screen position represented by the variable &LOC.

Display Multiple Lines of Data

To display multiple lines of data on a single screen, use the OUTPUTC statement after the OUTPUT statement. You must include the cursor positions so that the second line of text is not typed over the first.

The OUTPUTC command functions in the same manner as OUTPUT, but does not refresh an existing alternate screen image.

Display Alternate Screen

Alternate screen images are generated from panels stored in a panel's data set. To display an alternate screen image, use the DISPLAY command with the member containing the panel in the first operand position.

The following statement displays the screen from the panel stored in member NEWSCRN:

```
DISPLAY NEWSCRN
```

Position the Cursor at a Field

To position the cursor at a particular field in the alternate screen, specify the variable name associated with the field in the second operand position.

The following statement positions the cursor in the NAME field of the screen NEWSCRN:

```
DISPLAY NEWSCRN,NAME
```

Identify the Cursor Position

The following ACL/E variables represent the location of the cursor on an alternate screen in terms of its row, column, and row/column coordinate:

Locating the Cursor by Row

The variable &TCSRROW represents the row in which the cursor is positioned on an alternate screen. For example, if the cursor is in row 2, the value of &TCSRROW is 2.

Locating the Cursor by Column

The variable &TCSRCOL represents the column in which the cursor is positioned on an alternate screen. For example, if the cursor is in column 5, the value of &TCSRCOL is 5.

Locating the Cursor on the Total Screen

The variable &TCURSOR represents the position of the cursor in relation to the absolute screen size. For example, if the cursor is in row 2, column 5 on a screen that is 80 columns wide, the value of &TCURSOR is 85.

Chapter 10: Combining Session Information

This chapter discusses the commands used to enable ACL/E programs to coordinate interaction with multiple application sessions.

This section contains the following topics:

Use Shared Variables

You can combine information from several application sessions into one customized screen using ACL/E and panels, and you can define shared variables for passing information between sessions. In addition, you can suspend execution of the main program pending notification from the other programs.

You can have an ACL/E program that requires information from two or more application sessions. ACL/E allows you to set up "persistent" variables that retain their value even if you switch sessions.

Program That Copies the Account Number

The following program defines the variable ACNTNUM in the current session, assigns it a value taken from the current screen, and calls the second ACL/E program to run in another session:

```
******* Program name: ACCOUNT1 *******
   UDEFINE ACNTNUM,6           define account number variable
   SET ACNTNUM,&SCREEN(35,6)   set account number
    ACLPGM T2,ACCOUNT2         call second program
```

Program That Types the Account Number in Another Session

The following program uses the same variable to type the account number on the third column of the fifth row on the screen:

```
******* Program name: ACCOUNT2 *******
   SET CURSOR,5/3              set the cursor position
   KEY &ACNTNUM                type the account number
   UDELETE ACNTNUM             delete the variable
```

Use Global Variables

You may want to define information that can be used by any user's ACL/E programs in any session and will remain until deleted by a program. ACL/E allows you to create global variables that are defined in the symbol table. These variables can be used by any program running on that CA TPX until deleted. Global variables are defined with the GDEFINE command and deleted with the GDELETE command.

Wait for Notification from Scheduled Programs

Often, a main ACL/E program that starts several programs, each executing in a different session, needs to wait for results from those programs. The SUSPEND command causes the main ACL/E to be suspended until it receives notification that the other programs have completed their tasks.

Examples

Suppose that you have an ACL/E program that executes in an application session called CUSTOMER. This program calls three other ACL/E programs to query different customer databases. To suspend the calling ACL/E program until it receives notification from all three other programs, you would include the following statement in the calling program:

```
SUSPEND 3
```

To tell each of the three called ACL/E programs to report back to the calling ACL/E in the CUSTOMER session, you would include the following statement in each called program:

```
NOTIFY CUSTOMER
```

When the calling program receives the number of notifications specified in the SUSPEND statement, it continues by executing the next statement.

Appendix A: Programming Tips

This appendix gives you some helpful tips for programming with ACL/E. You can use this information to solve problems that you may encounter in your ACL/E programs, or to make your ACL/E programs more efficient.

This section contains the following topics:

Define and Delete Variables

This section discusses some common situations with user-defined variables and also shows you another way to define variables in an ACL/E program.

ACL/E User-defined Variables

A thorough understanding of how ACL/E user-defined variables work will help you to use them more efficiently.

1. If you use a UDEFINE statement to define a variable and then use a VDEFINE statement to define a variable with the same name, a SET statement will update only the variable defined with the VDEFINE statement. Therefore, if the variable is used in another ACL/E program, the value will not be passed to the called program.
2. If you use a GDEFINE statement to define a variable and then use a UDEFINE statement to define a variable with the same name, a SET statement will update only the variable defined with the UDEFINE statement. The new value will not be passed to the CA TPX global symbol table.
3. If you use a VDEFINE statement to define a variable and call another ACL/E program that uses a VDEFINE statement to define the same variable, the two variables will be treated as separate variables.

   If you want to pass the value of a variable to a called program, use a UDEFINE or GDEFINE statement to define the variable.

Define Variable with a SET Statement

You can use a SET statement to define a variable in an ACL/E program. The length of a variable defined with a SET statement is set to the length of the initial value of the variable. You can also clear the variable by assigning it a null value.

For example, the following statements define a variable and give it a null value:

```
SET AVAR,'12345678'      CREATE A VARIABLE WITH LENGTH 8
                         * AND A VALUE OF '12345678'
SET AVAR,''              RESET THE VALUE OF AVAR TO NULLS
```

Define Variable for Uppercase Data

You can specify a variable with the UPCHAR attribute and use that variable to convert data to uppercase. For example, the following program converts the lower case data contained in a variable to uppercase:

```
VDEFINE LOWERVAR,10
VDEFINE UPPERVAR,10,TYPE=UPCHAR
SET LOWERVAR,'lower case'
KEY 'Value = &LOWERVAR'
INPUT
CLEAR
SET UPPERVAR,&LOWERVAR
SET LOWERVAR,&UPPERVAR
KEY 'Value = &LOWERVAR'
INPUT
CLEAR
```

Position the Cursor

After locating a string on the screen with a SEARCH statement, you can position the cursor in the input field immediately following it by using the &LOC variable and a TABF statement. &LOC contains the row-column coordinates of the first position of the string.

The following statements move the cursor to the input field following the string "ACCOUNT NUMBER":

```
SEARCH 'ACCOUNT NUMBER' FIND 'ACCOUNT NUMBER' ON THE SCREEN
SET CURSOR,&LOC         MOVE THE CURSOR TO THE TEXT
TABF 1                  TAB TO THE NEXT INPUT FIELD
```

Switch Sessions in an ACL/E Program

When activating a session from an ACL/E program, you should use a WAIT command to give the application time to start before you switch to the new session.

For example, the following statements activate the session TSO, wait for it to start, and switch to the new session:

```
COMMAND 'A TSO'       ACTIVATE THE TSO SESSION
WAIT 1                GIVE THE SESSION TIME TO START
COMMAND 'TSO'         SWITCH TO THE TSO SESSION
```

ZCONN Variable

The variable ZCONN contains the session ID of the current session. You can use this variable to make sure the session switch was successful. For example, the following statements switch to the session VM2 and then check to make sure the session switch was successful before starting an ACL/E program:

```
        COMMAND 'VM2'           SWITCH TO THE VM APPLICATION
        WAIT 1                  GIVE THE SESSION TIME TO START
        COMPARE '&ZCONN','VM2'  DID WE SWITCH SESSIONS?
        BRANCH NE,FAILED        IF NOT, STOP THE PROGRAM
        COMMAND 'S VM2 VMACL'   START PROGRAM 'VMACL' IN VM2
FAILED  STOP
```

Manage Errors

You may want an ACL/E program to send a copy of the screen to the Log when an error occurs so that the programmer can easily see what was going on when the error occurred.

The following statements send the screen to the Log when the search string is not found:

```
        SEARCH 'xxx'            SEARCH FOR DATA
        BRANCH NE,ERROR1        IF NOT FOUND, PRINT THE SCREEN
              ...
ERROR1  COMMAND 'P * LOG'       PRINT SCREEN TO TPX LOG
        WAIT 1                  GIVE 'P' COMMAND TIME TO EXECUTE
              ...
```

Display an Error Message

You also can use an OUTPUT statement to display an error message on the screen. Follow the OUTPUT statement with an INPUT command to "freeze" the error message on the terminal until the user presses an action key.

```
OUTPUT 'Can''t go on',2/16  DISPLAY ERROR MESSAGE ON SCREEN
INPUT                       WAIT FOR ACTION KEY
```

Display Error Message with OPTION Command

If an ACL/E processing error occurs, you can use an OPTION command to continue execution at a specified label with a RESUME command. The error code and message will be saved in the predefined variables &ERR_RC and &ERR_MSG.

You could use the OPTION command to display an error message:

```
       OPTION ERROR,ERROR1
            ...
ERROR1 RESUME
       OUTPUT 'ACL/E Error',2/16  ANNOUNCE ERROR
       OUTPUTC '&ERR_MSG',3/16    DISPLAY ERROR MESSAGE ON SCREEN
       INPUT                      WAIT FOR ACTION KEY
            ...
```

Write a Signon Program for Multiple Applications

You can create one signon program to be used with multiple applications. The following code would execute the appropriate statements to sign on to an application:

```
      OPTION TERM,ON                  ALLOW USER TO SEE ACTION
      SCAN &SESS_INF(&SID),'APPL=CICS' IS IT A CICS APPLICATION
      BRANCH NE,IMSAPPL               NO, IT'S AN IMS
      OPTION TERM,OFF                 DO NOT DISPLAY ACTIVITY
      WINP                            WAIT FOR CICS GOOD
         ...                          MESSAGE, SIGNON
         ...                          STATEMENTS, ETC.
IMSAPPL etc.                          PROCESS IMS SIGNON, ETC.
```

Allow Other Work to be Dispatched

ACL/E programs that perform a tedious or redundant task continue to run without stopping until the entire task is completed. If there are no breaks in the ACL/E processing, CA TPX processes the statements as fast as the system allows and work from other users might not be dispatched until the program stops.

To prevent this, use a WAIT command to periodically interrupt the program, possibly once per iteration of a loop, to allow other work to be dispatched. Other commands that relinquish control are ENTER, PF, PA, INPUT, PAUSE, SEND, and SUSPEND.

A runaway limit can be set for a program to limit the number of times a loop can be executed. See Prevent Uncontrolled Loops (see page 62).

Sample Program

The following sample program uses a WAIT command to allow other work to be dispatched.

```
      SRCHWAIT 'EDIT --- '      AT PDS MEMBER LIST?
      BRANCH NE,STOP            IF NOT, STOP
      SET A1,&CURSOR            NOTE CURSOR LOCATION
GO    TABF 2                    TAB 2 LINES
      COMPARE A1,&CURSOR        CURSOR AT HOME POSITION?
      BRANCH EQ,STOP            YES, END OF LIST
      KEY 'S'                   SELECT MEMBER
      ENTER
CHG   KEY 'C ''*'' ''$'' ALL'   CHANGE ALL * TO $
      ENTER
      PF 3                      SAVE CHANGE
      WAIT 1                    ALLOW OTHER WORK TO DISPATCH
      BRANCH ANY,GO             PROCESS NEXT MEMBER
STOP  STOP
```

Prevent VTAM Buffers from Overloading

ACL/E programs that are written to perform a repetitive task may frequently display redundant data on the screen. A program that updates the screen more quickly than VTAM can deliver the data to the screen can cause the screen to flicker and overload the VTAM buffers. If the program does not have PAUSE or INPUT statements to request data from the user, the data most likely does not need to be displayed on the screen.

Commands to Use

If the application image is currently displayed on the screen, commands that send a change to the screen include DELETE, DOWN, EOF, HOME, KEY, LEFT, NL, RIGHT, SET CURSOR, TABB, TABF, TABL, OPTION TERM/OFF, and UP. If an alternate screen image is displayed, the OUTPUT and OUTPUTC commands also update the screen.

Sample Program

To prevent overloading of the VTAM buffers, place an OPTION TERM,OFF statement at an appropriate place in the program.

```
      OPTION TERM,OFF          DO NOT DISPLAY ACTION
      SRCHWAIT 'EDIT --- '     AT PDS MEMBER LIST?
      BRANCH NE,STOP           IF NOT, STOP
      SET A1,&CURSOR           NOTE CURSOR LOCATION
GO    TABF 2                   TAB 2 LINES
      COMPARE A1,&CURSOR       CURSOR AT HOME POSITION?
      BRANCH EQ,STOP           YES, END OF LIST
      KEY 'S'                  SELECT MEMBER
      ENTER
CHG   KEY 'C ''*'' ''$'' ALL'  CHANGE ALL * TO $
      ENTER
      PF 3                     SAVE CHANGE
      WAIT 1                   ALLOW OTHER WORK TO DISPATCH
      BRANCH ANY,GO            PROCESS NEXT MEMBER
STOP  STOP
```

TPXDEMO Session

TPXDEMO is a special CA TPX internal application. Its sole purpose is to serve as a base from which to display panels or run ACL/E programs. It should be defined as type TPX in the Application Characteristics Table (ACT).

When you sign on to a TPXDEMO session, you are greeted by a standard panel (TEN0062). The panel consists of a command line and a brief description.

Possible Uses

This application has many uses in ACL/E:

- Any facility built around fixed screen layouts, such as a tutorial, can be accessed as a session on the Menu by using a signon ACL/E program. If such a tutorial relates to an application system, it can be instantly accessed by the end user while working in the application.
- By making the TPXDEMO application the first session on the CA TPX Menu and setting it to start at logon, an initial set-up ACL/E program can be performed.
- You can use the TPXDEMO application as a way of allowing users to choose an initial application from the panel, as shown in Specifying an Initial Application from the Logo Panel (see page 105) in this chapter.
- You can use the TPXDEMO application in a program allowing users to use PF3 to log off from the Menu, as shown in Logging Off with PF3 (see page 105) in this chapter.

Specify Initial Application from the Logo Panel

Some installations must allow users to choose the initial application when signing on at the Logo panel. To do this:

1. Define a user variable on the Logo panel to contain the initial session ID. Alternatively, use the Account field and change the label on the panel definition.
2. Define a TPXDEMO session as the first session on the Menu, and specify it as invisible. Set "Display Menu" to N for the user.
3. Use the following ACL/E program as the signon program for TPXDEMO. It assumes that the "Account" field of the Logo panel is being used to contain the session ID.

```
        OPTION TERM,OFF             HIDE FROM USER
        COMPARE '&SNACCTV',' '      ANY APPL SELECTED?
        BRANCH EQ,EXIT              IF NOT, SHOW MENU
        COMMAND 'A &SNACCTV'        ACTIVATE IT IF NECESSARY
        WAIT 2                      LET IT COME UP
        COMMAND '&SNACCTV'          SWITCH TO IT
EXIT    KEY 'END'                   END TPXDEMO
        ENTER
        WAIT 3                      GIVE IT TIME TO SWITCH
```

The final wait is coded to ensure that the session switch occurs before the ACL/E program ends, to prevent the TPXDEMO screen image from being sent to the terminal. It will not affect the timing of the process.

Log Off with PF3

You can use the TPXDEMO application to allow users to log off from the Menu by pressing PF3. Define a session using TPXDEMO with a session key of PF3. This session could be named LOGOFF and appear on the menu or could be made invisible.

Use this ACL/E program as a signon program for the LOGOFF session:

```
OPTION TERM,OFF             HIDE FROM USER
SIGNOFF                     PERFORM /F
WINP                        GET PANEL
KEY 'END'                   END TPXDEMO
ENTER
```

Appendix B: Sample Programs

This appendix contains some sample programs that you may find useful. These programs can be copied almost verbatim, with only minor changes such as session IDs or program names.

This section contains the following topics:

A Cursor Sensitive Help Program

You can use ACL/E to create a program that displays cursor sensitive help for an application that does not have its own help facility. The code provided here shows the three different parts of a help program. Your help program will require much more code, but the basic principles of a help program are provided here.

Program That Waits for the User to Select Help

In the following program, PF1 is the HELP key. You can use any action key as the help key, but it should be a key that is not used by the application. The following code passes all input from the terminal directly to the application until the user presses PF1. When the user presses PF1, the ACL/E help program (HELPACL) is invoked.

```
LOOP    INPUT                   WAIT FOR AN ACTION KEY
        COMPARE '&TAID','PF1'   DID THE USER PRESS PF1?
        BRANCH NE,SKIP          IF NOT, SEND IT TO THE APPL
        ACLPGM HELPACL          IF PF1, INVOKE HELP PROGRAM
SKIP    SEND                    SEND ACTION KEY TO APPLICATION
        BRANCH ANY,LOOP         GO BACK AND WAIT FOR ANOTHER
```

Program That Finds the Screen ID

The following help program determines what screen is currently displayed so that the appropriate help panel can be provided. The program looks for an application identifier that is typically found in a specific location. In this example, we assume that the identifier starts at position two of the first line and is five characters long.

```
APPHELP  COMPARE &SCREEN(2,5),'PANEL'  IS THE ID THERE?
         BRANCH NE,SKIPIT              IF NOT, KEEP PROCESSING INPUT
         ACLPGM UHP&SCREEN(7,3)        START THE APPROPRIATE ACLPGM
SKIPIT   INPUT                         ACCEPT INPUT FROM THE USER AND
         SEND                          PASS IT TO THE APPLICATION
         BRANCH ANY,APPHELP            KEEP CHECKING THE SCREEN
```

Program That Displays the Help Screens

The following help program uses the DISPLAY command to present a panel you have defined in the panel library. You can make the help cursor sensitive by checking the value of &CURSOR. In this program, there are two help panels for general areas of the screen: UH001 for cursor positions 1 to 160 and UH001A for cursor positions 161 to 240. There are also field specific help panels for cursor positions 320 and 332. When the user is finished with a help panel, the program redisplays the application screen and invokes the program that monitors the user's input for PF1.

```
UHP001   SET A1,&CURSOR       NOTE CURRENT CURSOR POSITION
         COMPARE A1,160       CURSOR IN POS 160 OR LESS?
         BRANCH LE,GENHELP    GET GENERAL HELP SCREEN
         COMPARE A1,240       CURSOR IN POS 240 OR LESS?
         BRANCH LE,HELPA      GET HELP FOR SECTION A
         COMPARE A1,320       CURSOR IN POSITION 320?
         BRANCH EQ,HELPB      GET HELP FOR FIELD B
         COMPARE A1,332       CURSOR IN POSITION 332?
         BRANCH EQ,HELPC      GET HELP FOR FIELD C
GENHELP  DISPLAY UH001        DISPLAY GENERAL HELP PANEL
         BRANCH ANY,INP       GO AND WAIT FOR INPUT
HELPA    DISPLAY UH001A       DISPLAY HELP FOR SECTION A
         BRANCH ANY,INP       GO AND WAIT FOR INPUT
HELPB    DISPLAY UH001B       DISPLAY HELP FOR FIELD B
         BRANCH ANY,INP       GO AND WAIT FOR INPUT
HELPC    DISPLAY UH001C       DISPLAY HELP FOR FIELD C
         BRANCH ANY,INP       GO AND WAIT FOR INPUT
INP      INPUT                WAIT FOR USER TO PRESS ACTION KEY
         OPTION IMAGE,APPL    REDISPLAY APPLICATION SCREEN
         BRANCH ANY,CONTINUE      ...
```

A Program That Processes PDS Members

You can use the following program to automate a tedious and repetitive task involving the members of a PDS. This program selects each member, processes the changes, and stops when it has reached the end of the list.

```
        SEARCH 'PRIMARY OPTION MENU'  START ACL FROM ISPF
        BRANCH NE,DONE1               NOT IN ISPF, STOP
        KEY '2'                       SELECT EDIT
        ENTER
        HOME                          PUT CURSOR IN 1ST INPUT FIELD
        TABF 8                        TAB TO DATA SET NAME
        KEY 'data set name'           KEY NAME OF PDS
        ENTER
        SET A1,&CURSOR                SET HOME CURSOR LOCATION
LOOP    TABF 1                        TAB TO PDS MEMBER NAME
        COMPARE A1,&CURSOR            CURSOR AT COMMAND LINE?
        BRANCH EQ,DONE2               YES, FINISHED THE LIST
        KEY 'S'                       NO, SELECT THE MEMBER
        ENTER
        C '''???''' '''&&&''' ALL     MAKE CHANGES TO MEMBER
        PF 3
        BRANCH ANY,LOOP               GO TO NEXT MEMBER
DONE1   HOME
        KEY 'NOT IN ISPF PRIMARY'     MUST START ACL IN ISPF
DONE2   STOP
```

A Program That Prints a PDS Member from ISPF Edit or Browse

The following ACL/E program will generate a quick print request of a PDS member from ISPF edit or browse:

```
        OPTION FLOW,OFF
        OPTION TERM,ON
        HOME                    PUT CURSOR IN COMMAND LINE
        UP 1                    LOCATE START OF THE
        LEFT 3                  DATA SET NAME
        SET A1,&CURSOR          NOTE ITS LOCATION
        SET LOC,&CURSOR         SET LOC VARIABLE FOR SEARCHF
        SEARCHF ')'             LOOK FOR PAREN IN CURRENT FIELD
        BRANCH NE,NOMEMBER      IF NOT FOUND, END PROGRAM
        SET A2,&LOC             SET A2 TO LOCATION OF RIGHT PAREN
        SUB A2,A1               SET A2 TO LENGTH UP TO RIGHT PAREN
        ADD A2,1                ADD 1 FOR THE RIGHT PAREN
        HOME                    PUT CURSOR ON THE COMMAND LINE
        KEY 'TSO PRINTO ''&SCREEN(&A1,&A2)'''
        ENTER                   PRINT THE MEMBER
        STOP
NOMEMBR OUTPUT 'MEMBER NAME NOT SPECFIED',2/16
        STOP                    PRINT ERROR MESSAGE AND STOP
```

A Program That Signs the User off CA TPX

This program logs a user off all of the active sessions and signs the user off at the end of the day. The program executes a termination ACL/E program for each active session and then signs the user off CA TPX.

Note: This program is redundant if you have already specified termination ACL/E programs for all of a user's sessions to execute when the user issues the "K" or "F" command.

```
OPTION TERM,OFF          END OF DAY, NO NEED TO WATCH
ACLPGM T1,TOFF           HAVE ACLPGM TOFF LOGOFF TSO 1
ACLPGM T3,TOFF           HAVE ACLPGM TOFF LOGOFF TSO 3
ACLPGM CICST,CICSOFF     HAVE ACLPGM CICSOFF LOGOFF CICS
ACLPGM IMSP,IMSOFF       HAVE ACLPGM IMSOFF LOGOFF IMSPROD
    ... etc.
SIGNOFF                  SIGN USER OFF TPX
STOP
```

A Program That Reassigns PF Keys

You can use the variable &TAID to change user input for an application that uses non-standard PF keys. The following program displays a customized panel with the standard PF key values, accepts the user's input, and changes the &TAID entered by the user to one that the application accepts.

```
        DISPLAY CUSTMPNL        DISPLAY CUSTOMIZED PANEL
        INPUT                   WAIT FOR USER ACTION
        COMPARE &TAID,'PF1'     WAS PF1 ENTERED?
        BRANCH EQ,HELP          YES, USER WANTS HELP
        COMPARE &TAID,'PF3'     WAS PF3 ENTERED?
        BRANCH EQ,END           YES, USER WANTS TO END
        COMPARE &TAID,'PF7'     WAS PF7 ENTERED?
        BRANCH EQ,NEXT          YES, USER WANTS NEXT SCREEN
        COMPARE &TAID,'PF8'     WAS PF8 ENTERED?
        BRANCH EQ,PREV          YES, USER WANTS PREVIOUS SCREEN
           ...
HELP    PF 9                    SEND PF9 INSTEAD OF PF1
        BRANCH ANY,RETURN
END     PF 12                   SEND PF12 INSTEAD OF PF3
        BRANCH ANY,RETURN
NEXT    PF 11                   SEND PF11 INSTEAD OF PF7
        BRANCH ANY,RETURN
PREV    PF 10                   SEND PF10 INSTEAD OF PF8
        BRANCH ANY,RETURN
            ...
```

A Program That Submits an Action Key

If you have users who work from terminals that do not have some action keys, such as CLEAR or PA keys, you can write an ACL/E program to pass the key to the application as if the user had pressed the key on the keyboard.

Program That Submits the CLEAR Key

This program passes the CLEAR key to the application:

```
   CLEAR                SUBMIT THE CLEAR KEY
   STOP
```

Program That Submits the PA3 Key

This program passes the PA3 key to the application:

```
PA 3                    SUBMIT THE PA3 KEY
STOP
```

A Program That Displays the User ID and Terminal ID

A possible name for this ACL/E program is "WHO". The program displays the current user ID and terminal ID on the terminal to aid in answering Help Desk questions.

```
OPTION TERM,ON
HOME
KEY 'YOU ARE &USERID AT &NETNAME'
STOP
```

A Program That Monitors Storage Statistics

This program can be set up to continuously run in TPXOPER and take storage statistics periodically and write them to the Log. These statistics might then be used to gather data to make tuning decisions.

```
       OPTION MAXI,0          NO LIMIT TO EXECUTION
       OPTION TERM,ON
LOOP   KEY 'D STOR'           DISPLAY STORAGE BELOW THE LINE
       ENTER
       COMMAND 'P * LOG'      SEND RESULT TO TPX LOG
       WAIT 4
       KEY 'D STORXA'         DISPLAY STORAGE ABOVE THE LINE
       ENTER
       COMMAND 'P * LOG'      SEND RESULT TO TPX LOG
       WAIT 1800              WAIT 30 MINUTES
       BRANCH ANY,LOOP        GET NEW STORAGE STATISTICS
```

Appendix C: Actions and Commands

This appendix lists the actions that you can perform in an ACL/E program and the commands that you use to do so. Each table in the appendix contains actions that perform the same type of function.

This section contains the following topics:

To Manipulate and Examine the Screen Image

Action	Command	Operands
Delete a character from the virtual screen.	DELETE	
Erase to the end of a field.	EOF	
Clear all input fields.	ERASE	
Type text on the virtual screen.	KEY	'*string*' [C|X]
Put the virtual terminal in insert mode.	INSERT	
Take the virtual terminal out of insert mode.	RESET	
Search the virtual screen image for *'string'*.	SEARCH	'*string*'
Search the current field from &LOC for *'string'.*	SEARCHF	'*string*'
Search new data for *'string'* before sending it to the application.	SEARCHN	'*string*'

To Move the Cursor

Action	Command	Operands
Move the cursor up *n* lines.	UP	*n*
Move the cursor down *n* lines.	DOWN	*n*
Move the cursor left *n* columns.	LEFT	*n*
Move the cursor right *n* columns.	RIGHT	*n*
Set the cursor position at screen position *n* or row/column *n/n*.	SET	CURSOR,*[n[/n]]*
Move the cursor to the first input field on the screen.	HOME	
Position the cursor at the first input field of the *n*th row from the current position. Rows without an input field are not counted.	NL	*n*
Move the cursor *n* input fields forward.	TABF	*n*
Move the cursor *n* input fields back.	TABB	*n*
Move the cursor to the last input field on the screen.	TABL	

To Transmit to the Application

Action	Command	Operands
Send an ATTENTION to the application.	ATTN	
Clear the virtual screen image and send the CLEAR key to the application.	CLEAR	
Transmit modified data fields to the application.	ENTER	
Transmit modified fields with the program function key *n*.	PF	*n*
Transmit program attention key *n*.	PA	*n*
Transmit modified data with the current value of &TAID.	SEND	

To Control the ACL/E Program's Execution

Action	Command	Operands
Invoke another ACL/E program.	ACLPGM	[*sessionID*,]*aclpgm-name*
Receive input from the terminal.	INPUT	
Branch to *label* on *condition*.	BRANCH	*condition*,*label*
Set option *option*	OPTION	{*option*,*value*}
Stop the ACL/E program to let the user enter data.	PAUSE	
Terminate execution of this ACL/E program.	STOP	
Terminate the application session immediately.	TERMSESS	
Wait *n* seconds.	WAIT	*n*
Search for *'string'* and wait if it is not found or if application will not be sending more data.	SRCHWAIT	*'string'*

To Perform Calculations and Logical Operations

Action	Command	Operands
Set the value of an accumulator or a variable.	SET	*{accumulator,value}* {*variable*, *value*}
Add an accumulator or an integer to an accumulator.	ADD	{*accumulator*,*accumulator*} {*accumulator*,*integer*}
Subtract an accumulator or an integer from an accumulator.	SUB	{*accumulator*,*accumulator*} {*accumulator*,*integer*}
Compare an accumulator or a string with an accumulator, an integer, or a string, and set a condition code.	COMPARE	{*accumulator*,*accumulator*} {*accumulator*,*n*} {*accumulator*,*string*} {*string*,*accumulator*} {*string*,*n*} {*string*,*string*}
Search a string for another string and set a condition code.	SCAN	{*string1*,*string2*}

To Issue Commands and Use Services

Action	Command	Operands
Issue a command *'string'*.	COMMAND	*'string'*
Sign the user off (as if a /F command was issued).	SIGNOFF	
Log the terminal off (as if a /K command was issued).	LOGOFF	

To Help Troubleshoot an ACL/E Program

Action	Command	Operands
Write a message in the log specified in the LOG DD statement of the startup procedure.	MSG	*'message'*
Continue execution of program at specified label and save error message and return code in variables.	OPTION	ERROR,*label*
Display the data flow between the application session and the ACL/E program on the terminal screen.	OPTION	TERM,ON
Record the executed statements in the Message Log.	OPTION	FLOW,ON

To Define and Delete Variables

Action	Command	Operands
Define CA-TPX-level (global) variables.	GDEFINE	*variable*,n,TYPE=*type*
Delete CA-TPX-level (global) variables.	GDELETE	*variable*
Define user-level (persistent) variables.	UDEFINE	*variable*,*n*,TYPE=*type*
Delete user-level (persistent) variables.	UDELETE	*variable*
Define ACL/E-level (transient) variables.	VDEFINE	*variable*,*n*,TYPE=*type*
Delete ACLE-level (transient) variables.	VDELETE	*variable*

Action	Command	Operands
Set the value of a variable.	SET	*variable,n*

To Display an Alternate Screen Image

Action	Command	Operands
Display the panel from 'member'.	DISPLAY	*member*[,*cursor-variable*]
Display composite of application image and output *'string'*.	OUTPUT	'*string*'
Continue displaying additional output *'string'*.	OUTPUTC	'*string*'
Remove the alternate screen image displayed with DISPLAY, OUTPUT, or OUTPUTC and redisplay the application image.	OPTION	*IMAGE,APPL*

To Integrate Intra-Session ACL/E Execution

Action	Command	Operands
Notify the main program executing in *'sessionID'* of the scheduled program completion.	NOTIFY	*sessionID*
Suspend execution of the main program until notification is received from all *n* scheduled programs.	SUSPEND	*n*

Appendix D: Command Summary

This appendix lists the ACL/E commands, their syntax, a description of their operands, and usage information.

This section contains the following topics:

Command Syntax Diagrams

Command syntax is diagrammed along the *command line* from left to right. If the entire syntax cannot fit on one line, it splits and is continued below the first line.

Sample COMMAND Format

Refer to the following sample diagram and descriptions for an explanation of command syntax.

```
->>---COMMAND--(1)-------------------------------------------------------------
---->

->---required operand1--(2)--- = variable1|variable2 ---(3)-------------------
-(5)->
  |-required operand2-----------------------------------------------------------
-|
  +-required operand3= variable ------------------------------------------------
-+
                                    +- optional operand -(4)-required variable1-+
                                                            |-required variable2-|
                                                            +-required variable3-+

                                                          |----------------------
---+
                                                          V
   | (7)
->-----------------------------------------------------------------------------
------><
    |-- default operand --(6)----------------------|    |-- repeatable operand1
--|
    +-- optional operand ---------------------------+    +-- repeatable operand2
--+
                          |-- optional variable1 --|
                          +-- optional variable2 --+
```

Notes:

1. The COMMAND always comes first on the command line and is printed in all capitals.
2. Required operands fall *on* the line.
3. Variables are printed in *italics*. A choice of variables can be represented with the "|" separator.
4. Optional operands are diagrammed *below* the command line.
5. Lists of operands, one of which must (required) or may (optional) be specified, are listed vertically in *groups*.
6. Default operands are underlined.
7. Operands that can be *repeated*, or a list or operands, more than one of which can be specified, have a repeating arrow.

ACLPGM Command

The ACLPGM command schedules an ACL/E program to be executed.

Format

The ACLPGM command has the following syntax:

```
>>--- ACLPGM ----------------------progname ---------><
                +- sessid, -+
```

Operands

The ACLPGM command has the following operands:

Operand	Explanation
sessid	Specifies the session the specified ACL/E program is to run in. The session must be active when the ACLPGM command is executed.
progname	Specifies the name of the ACL/E program that the ACLPGM command is to schedule.

Usage

If you do not specify a session, the program runs in the current session and the current program is suspended until the called program completes execution.

All user-defined variables are available to the ACL/E program that is called.

ADD Command

The ADD command adds the value of an accumulator or integer to an accumulator.

Format

The ADD command has the following syntax:

```
>>---accumulator 1 ------- accumulator 2 ---------------><
                        +- n --------------+
                        +- variable --------+
```

Operands

The ADD command has the following operands:

Operand	Explanation
accumulator 1	Specifies one of the four accumulators, A1 through A4, that is incremented by the ADD command.
accumulator 2	Specifies an accumulator whose contents is added to the accumulator you specified in *accumulator 1*.
n	Specifies an integer that is added to the specified accumulator.
variable	Specifies a numeric variable that is added to the specified accumulator.

ATTN Command

The ATTN command issues an attention interrupt to the application.

Format

The ATTN command has the following syntax:

```
>>----ATTN -------------------------------><
```

Usage

The attention interrupt is sent to the application regardless of how the Attn key is defined in the users System Options Table.

BRANCH Command

The BRANCH command causes the ACL/E program to branch to a specified line depending on the current value of the condition code.

Format

The BRANCH command has the following syntax:

```
->>---BRANCH------ ANY, ------------------- label ------><
                +- condition code, ---+
```

Operands

The BRANCH command has the following operands:

Operand	Explanation
ANY	Makes the BRANCH command branch regardless of the current value of the condition code.
condition code	Specifies a possible condition code value. If the condition code set in a previous command matches this value, the BRANCH command branches. Possible condition code values are EQ, NE, GT, LT, LE, or GE.
label	Specifies the program line that the BRANCH command branches to. You cannot specify a variable in this operand.

Usage

The condition code that the BRANCH command acts on is set by the most recent COMPARE, SEARCH, SEARCHF, SEARCHN, SEARCHWAIT, or HOME statement.

CLEAR Command

The CLEAR command simulates the pressing of the Clear key and clears the screen. As with all of the action keys (Enter, Clear, PA, and PF*n*), ACL/E program execution is suspended until a response is received from the application.

Format

The CLEAR command has the following syntax:

```
->>---- CLEAR ------------------------------------><
```

COMMAND Command

The COMMAND command executes a command within an ACL/E program.

Format

The COMMAND command has the following syntax:

```
->>---- COMMAND -------  'TPX command' -------------------><
```

Operands

The COMMAND command has the following operand:

Operand	Explanation
TPX command	Specifies the command that is executed by the COMMAND command. The command character is automatically supplied by ACL/E.

COMPARE Command

The COMPARE command determines the relationship between two operands and sets a condition code based on the results of the comparison.

The COMPARE command can compare:

- The contents of one of the accumulators (A1 through A4) and the contents of another accumulator
- The contents of one of the accumulators (A1 through A4) and an integer
- Two strings
- A string and a variable.

Format

The COMPARE command has the following syntax:

```
->>---- COMPARE ------- operand 1 ------ operand 2 ------------><
```

Operands

The COMPARE command has the following operands:

Operand	Explanation
operand 1	Specifies one of the two items that are compared by the COMPARE command.
operand 2	Specifies the item that the item you specified in *operand 1* is compared with.

Usage

The condition code set by the COMPARE command determines if a subsequent BRANCH command performs a branch. A BRANCH command should be the next executable statement after the COMPARE command.

The COMPARE command sets one of the following condition codes:

EQ—equal to

NE—not equal to

GT—greater than

LT—less than

LE—less than or equal to

GE—greater than or equal to

You can use substring notation when specifying a string as an operand in the COMPARE statement. See the chapter "Defining and Manipulating Variables."

DELETE Command

The DELETE command simulates the pressing of the Delete key. It deletes the character at the cursor position and shifts the remaining data in the field one space to the left.

Format

The DELETE command has the following syntax:

```
->>---- DELETE -----------------------------------><
```

DISPLAY Command

The DISPLAY command displays an alternate screen image.

Format

The DISPLAY command has the following syntax:

```
>>--- DISPLAY ------ panel name ----------------------------><
                                  +- ,varname ---+
```

Operands

The DISPLAY command has the following operands:

Operand	Explanation
panel name	Specifies the alternate screen panel displayed by the DISPLAY command.
varname	Specifies a variable. The cursor is placed in the field associated with this variable.

DOWN Command

The DOWN command moves the cursor down the screen. If the cursor reaches the bottom of the screen, it wraps to the top of the screen and continues downward from there.

Format

The DOWN command has the following syntax:

```
>>------ DOWN ------ n -------------------------------><
```

Operands

The DOWN command has the following operand:

Operand	Explanation
n	Specifies how many lines down the screen the cursor is moved.

ENTER Command

The ENTER command simulates the pressing of the Enter key and transmits screen input data back to the application. As with all of the action keys (Enter, Clear, PA, and PF*n*), program execution is suspended until a response is received from the application.

Format

The ENTER command has the following syntax:

```
>>------ ENTER ------------------------------------------><
```

EOF Command

The EOF command simulates the pressing of the Erase EOF key and replaces all characters on the screen from the cursor to the end of the field with null characters.

Format

The EOF command has the following syntax:

```
>>------ EOF -------------------------------><
```

GDEFINE Command

The GDEFINE command defines a global variable. This variable can be set or referenced by any program running on this CA-TPX system.

Format

The GDEFINE command has the following syntax:

```
>>--- GDEFINE --- varname,n ------------------------------------------------><
                                +-TYPE= ----- ,CHAR ------------------------+
                                |             +- NOTRUNC ----+              |
                                |                            |              |
                                |-- ,UPCHAR ---------------------------------|
                                |             +- NOTRUNC ----+              |
                                |                            |              |
                                |-- ,NUM ------------------------------------|
                                |-- ,BIN ------------------------------------|
                                +-- ,HEX ------------------------------------+
```

Operands

The GDEFINE command has the following operands:

varname

Specifies the name of the transient variable defined in the GDEFINE command.

n

Specifies thelength of thevariable. Thedefault length iseight.

TYPE=

Specifies the type of variable defined in the UDEFINE command. Must specify CHAR, UPCHAR, NUM, BIN, or HEX.

CHAR

Indicates that the variable is a character variable.

UPCHAR

Indicates that the variable is an uppercase character variable.

NUM

Indicates that the variable is a numeric variable.

BIN

Indicates that the variable is a binary variable.

HEX

Indicates that the variable is a hexadecimal variable.

NOTRUNC

Indicates that any trailing blanks in the variable will not be deleted when variable substitution takes place. This operand can be used only with the CHAR and UPCHAR operands.

GDELETE Command

The GDELETE command deletes a global variable.

Format

The GDELETE command has the following syntax:

```
>>------ GDELETE ------ varname ---------------------------><
```

Operands

The GDELETE command has the following operand:

Operand	Explanation
varname	Specifies the name of the global variable deleted in the GDELETE command.

HOME Command

The HOME command simulates the pressing of the Home key and moves the cursor to the first unprotected input field on the screen.

Format

The HOME command has the following syntax:

```
>>------ HOME ----------------------------------------><
```

INPUT Command

The INPUT command suspends operation of the ACL/E program until input is received from the terminal. When input arrives it is merged with the current screen image and execution resumes.

The data received by the INPUT command can by used by subsequent SEARCHN and SEND commands.

Format

The INPUT command has the following syntax:

```
>>------ INPUT ---------------------------------------><
```

INSERT Command

The INSERT command simulates the pressing of the Insert key and places the virtual terminal in INSERT mode. The RESET command takes the terminal out of INSERT mode.

Format

The INSERT command has the following syntax:

```
>>------ INSERT -------------------------------------><
```

KEY Command

The KEY command types a specified string on the screen.

Format

The KEY command has the following syntax:

```
>>------ KEY ------ string ------------------------------><
                +- 'string' -----------------+
                                |- C ---|
                                +- X ---+
```

Operands

The KEY command has the following operands:

Operand	Explanation
string	Specifies the string that the KEY command displays on the screen. If the string contains spaces or punctuation, or you specify the C or X operands, it must be enclosed in single quotes.
C	Indicates that the *string* you specified consists of normal keyboard characters. This is the *default*.
X	Specifies that the *string* you specified consists of hex codes representing EBCDIC characters.

LEFT Command

The LEFT command moves the cursor to the left. If the cursor reaches the left edge of the screen, it wraps around to the right side and continues moving left.

Format

The LEFT command has the following syntax:

```
>>------ LEFT ------ n ------------------------><
```

Operands

The LEFT command has the following operand:

Operand	Explanation
n	Specifies how many spaces the cursor is moved.

LOGOFF Command

The LOGOFF command simulates the /K command.

Format

The LOGOFF command has the following syntax:

```
>>------ LOGOFF -------------------------------------><
```

Usage

If you have specified K or F in the "Inactivate on" field of the user's definition, the sessions will not be inactivated. In this case you must use COMMAND 'K' or COMMAND 'F' instead of the LOGOFF command.

MSG Command

The MSG command writes a string to the message log. This command is helpful in debugging ACL/E programs.

Format

The MSG command has the following syntax:

```
>>------ MSG ------ string -------------------------><
                 +- 'string' ---+
```

Operands

The MSG command has the following operand:

Operand	Explanation
string	Specifies text sent to the CA-TPX message log by the MSG command. The string can be any text up to 60 characters in length.

NL Command

The NL command simulates the pressing of the Return key and advances the cursor to the first input field after the beginning of the next row on the screen.

Format

The NL command has the following syntax:

```
>>------ NL ---- n ----------------------------------------><
```

Operands

The NL command has the following operand:

Operand	Explanation
n	Specifies how many times the NL command advances the cursor.

NOTIFY Command

The NOTIFY command adds one to the NOTIFY count of an ACL/E program executing in the specified session.

Format

The NOTIFY command has the following syntax:

```
>>------ NOTIFY ------ sessionID ------------------------><
```

Operands

The NOTIFY command has the following operand:

Operand	Explanation
sessionID	Specifies the session containing an executing ACL/E program whose notify count is incremented by the NOTIFY command.

Usage

The NOTIFY command works in conjunction with a SUSPEND command to allow communication between ACL/E programs running in different sessions. When a suspended ACL/E program's NOTIFY count is equal to or greater than its SUSPEND count, the program resumes execution.

For more information, see the SUSPEND command.

OPTION Command

The OPTION command controls various options associated with the execution of ACL/E programs.

Format

The OPTION command has the following syntax:

```
>>---- OPTION ------- ERROR, -----label ----------------------><
                  |                                          |
                  |-- FLOW, ----- ON ------------------------|
                  |            +- OFF --+                    |
                  |                                          |
                  |-- IMAGE, ---- APPL ----------------------|
                  |            +- APPL_NOSEND ----+----------|
                  |                                          |
                  |-- MAXI, ----- n -------------------------|
                  |            +- 50 ---+                    |
                  |                                          |
                  |-- KEYERR, --- ON ------------------------|
                  |            +- OFF --+                    |
                  |                                          |
                  |-- TERM, ----- ON ------------------------|
                  |            +- OFF --+                    |
                  |                                          |
                  |-- TIME= ---- N,label --------------------|
                  |           +- OFF --------+               |
                  |                                          |
                  +-- RESIDENT, ----- ON --------------------+
                                    +- OFF --+
```

Operands

The OPTION command has the following operands:

Operand	Explanation
ERROR, *label*	Indicates a label at which program execution will continue in case an ACL/E processing error occurs. At the label you can: Issue the RESUME command to resume execution of the program. Terminate execution of the program. Perform other ACL/E processing. When this operand is used, the variable &ERR_RC is set with the return code of the error, and the variable &ERR_MSG is set with the text of the error message.
FLOW,ON|OFF	Indicates whether or not the ACL/E program records the sequence of executed commands in the message log. The *default* is OFF, which prevents commands from being recorded in the log.
IMAGE,APPL|APPL_NOSEND	IMAGE,APPL indicates that an alternate image is to be removed from the screen and the application image redisplayed. IMAGE,APPL_NOSEND removes the alternate image, but does not redisplay the application image.
MAXI,*n*|50	Indicates the total number of statements (*n*) that can be executed by an ACL/E program. The *default* is 50. You can specify an integer between zero and 64,000.
KEYERR,ON|OFF	Indicates whether or not the ACL/E program terminates if an error occurs when the program is keying data into an input field. The *default* is ON, which causes the program to terminate. This operand exists for compatibility with earlier versions of ACL/E, and it is recommended that the ERROR operand be used. KEYERR, OFF overrides the OPTION,ERROR command if a -KEY-OPERATION ENCOUNTERED AN OUT-OF-BOUNDS CONDITION error occurs.
TERM,ON|OFF	Indicates whether or not the application data is sent to the terminal during ACL/E program execution. The *default* is OFF, which prevents data from being sent.

Operand	Explanation
TIME=*N,label*	Indicates a time-out value for ACL/E processing. If no activity occurs for the time-out period, the program resumes at the statement marked with the *label*. *N* specifies the time-out period in seconds. A value of OFF prevents the program from timing out.
TIME,OFF	Prevents the program from timing out.
RESIDENT,ON|OFF	Indicates whether or not the ACL/E program remains in storage after the program ends. The *default* is OFF, which prevents the program from being stored.

OUTPUT Command

The OUTPUT command creates an alternate screen image that is a composite of the application screen image and a specified string.

Format

The OUTPUT command has the following syntax:

```
>>------ OUTPUT ------ string ------------------------------><
                   +- 'string' --------------------+
                                  |-- ,p -----|
                                  +--  ,r/c --+
```

Operands

The OUTPUT command has the following operands:

Operand	Explanation
string	Specifies the string that the OUTPUT command displays on the application screen image.
p	Specifies the starting location for the *string* on the screen as an absolute screen position.
r/c	Specifies the starting location for the *string* on the screen as a row number and column number.

Usage

The *default* starting location for the string is the location specified in the &LOC variable.

OUTPUTC Command

The OUTPUTC command functions in the same manner as the OUTPUT command, except that it does not refresh the existing screen image. This allows you to place multiple output lines on a single screen image without the old lines being erased.

Format

The OUTPUTC command has the following syntax:

```
>>------ OUTPUTC ------ string ----------------------------><
                    +- 'string' --------------------+
                                    |-- ,p -----|
                                    +--  ,r/c --+
```

Operands

The OUTPUTC command has the following operands:

Operand	Explanation
string	Specifies the string that the OUTPUTC command displays on the application screen image.
p	Specifies the screen starting location for the *string* as an absolute screen position.
r/c	Specifies the screen starting location for the *string* as a row number and column number.

Usage

The *default* screen starting location for the string is the location specified in the &LOC variable.

PA Command

The PA command simulates the pressing of a PA key. As with all of the action keys (Enter, Clear, PA, and PF*n*), ACL/E program execution is suspended until a response is received from the application.

Format

The PA command has the following syntax:

```
>>------ PA ------ key number ---------------------------><
```

Operands

The PA command has the following operand:

Operand	Explanation
key number	Specifies the PA key that the PA command simulates. The *key number* must be an integer between one and three.

PAUSE Command

The PAUSE command suspends ACL/E execution to allow the user to press an action key (Enter, CLEAR, PA, or PF*n*). This allows the user to perform some action, such as selecting from a menu or entering data before the next ACL/E statement is executed. The user's action is transmitted to the application, and ACL/E execution resumes when the application responds. This operand exists for compatibility with earlier versions of ACL/E, and it is recommended that the INPUT statement or a combination of INPUT and SEND statements be used.

Format

The PAUSE command has the following syntax:

```
>>------ PAUSE ----------------------------------><
```

PF Command

The PF command simulates the pressing of a PF key and transmits all screen input data to the application. As with all of the action keys (Enter, Clear, PA, and PF*n*), ACL/E program execution is suspended until a response is received from the application.

Format

The PF command has the following syntax:

```
>>------ PF ------ key number --------------------------><
```

Operands

The PF command has the following operand:

Operand	Explanation
key number	Specifies the PF key that the PF command simulates. The *key number* must be an integer between 1 and 24.

RESET Command

The RESET command simulates the pressing of the Reset key and takes the virtual terminal out of INSERT mode. The INSERT command places the virtual terminal in INSERT mode.

Format

The RESET command has the following syntax:

```
>>------ RESET -------------------------------------><
```

RESUME Command

The RESUME command resumes execution of a program after an ACL/E processing error. The program resumes execution at the line following the statement that caused the error.

This command can be used only in conjunction with an OPTION command with the ERROR operand.

Format

The RESUME command has the following syntax:

```
>>------ RESUME -----------------------------------><
```

RIGHT Command

The RIGHT command simulates the pressing of the right arrow key. If the cursor reaches the right edge of the screen, it wraps around to the left side and continues moving right.

Format

The RIGHT command has the following syntax:

```
>>------ RIGHT ------ n ----------------------------><
```

Operands

The RIGHT command has the following operand:

Operand	Explanation
n	Specifies how many times the RIGHT command advances the cursor.

SCAN Command

The SCAN command searches a specified string for another specified string of characters. If it finds the string it is looking for, the SCAN command sets the condition code to EQ. If the string is not found it is set to NE.

Format

The SCAN command has the following syntax:

```
>>-- SCAN -- string -- ,target string ------------------------><
                                        +--,START= -- n --+
```

Operands

The SCAN command has the following operands:

Operand	Explanation
string	Specifies the string that the SCAN command searches for the target string.
target string	Specifies the string that the SCAN command searches for within the specified string.
,START=*n*	Specifies the character position within the search string at which the SCAN command starts looking for the target string. The default value is zero (0).

SEARCH Command

The SEARCH command searches the screen either for a specified string of characters or for any non-null or non-blank data. If it finds what it is looking for, the SEARCH command sets the condition code to EQ.

Format

The SEARCH command has the following syntax:

```
>>--- SEARCH ------ ANY ------------------------------------><
                 +----- string ----------------------+
                    +-- 'string' --+    |-- C --|
                                   +-- X --+
```

Operands

The SEARCH command has the following operands:

Operand	Explanation
ANY	Sets a condition code of EQ if the SEARCH command finds any non-null or non-blank data on the screen.
string	Specifies the string that the SEARCH command searches for on the screen. If the string contains any punctuation marks or spaces, or you specify the C or X operand, it must be enclosed with single quotes.
C	Indicates that the *string* you specify consists of normal keyboard characters. This is the *default*.

Operand	Explanation
X	Indicates that the *string* you specify consists of hexadecimal codes.

Usage

A BRANCH statement should always be the next executable statement after the SEARCH statement.

SEARCHF Command

The SEARCHF command searches an area of the screen image, starting at the screen position specified in the &LOC variable and ending at the end of the current field. If it finds what it's looking for, the SEARCH command sets the condition code to EQ.

Format

The SEARCHF command has the following syntax:

```
>>-- SEARCHF ------ ANY ------------------------------------><
                 +----- string ----------------------+
                    +-- 'string' --+    |-- C --|
                                   +-- X --+
```

Operands

The SEARCHF command has the following operands:

Operand	Explanation
ANY	Sets a condition code of EQ if the SEARCHF command finds any non-null or non-blank data on the screen.
string	Specifies the string that the SEARCHF command searches for on the screen. If the string contains any punctuation marks or spaces, or you use the C or X operand, it must be enclosed with single quotes.
C	Indicates that the *string* you specified consists of normal keyboard characters. This is the *default*.
X	Indicates that the *string* you specified consists of hexadecimal codes representing EBCDIC characters.

Usage

The SEARCHF command is often used after a successful SEARCH operation, because the &LOC variable is set by the SEARCH command.

A BRANCH command should be the first executable statement following the SEARCHF command.

SEARCHN Command

The SEARCHN command searches the data received by the immediately preceding INPUT statement and sets a condition code of EQ if it finds what it is looking for.

Format

The SEARCHN command has the following syntax:

```
>>--- SEARCHN ---- ANY ------------------------------------><
                +----- string -------------------------+
                   +-- 'string' --+     |-- C --|
                                        +-- X --+
```

Operands

The SEARCHN command has the following operands:

Operand	Explanation
ANY	Sets a condition code of EQ if the SEARCHN command finds any non-null or non-blank data on the screen.
string	Specifies the string that the SEARCHN command searches for on the screen. If the string contains any punctuation marks or spaces, or you specify the C or X operand, it must be enclosed with single quotes.
C	Indicates that the *string* you specified consists of normal keyboard characters. This is the *default*.
X	Indicates that the *string* you specified consists of hexadecimal codes representing EBCDIC characters.

Usage

A BRANCH statement should always be the next executable statement after the SEARCHN statement.

SEND Command

The SEND command sends all modified fields in the screen image and the current Attention Identifier (AID) to the application.

Format

The SEND command has the following syntax:

```
>>---- SEND ---------------------------------><
```

Usage

This command allows an ACL/E program to send data to the application after receiving it from the user with an INPUT statement.

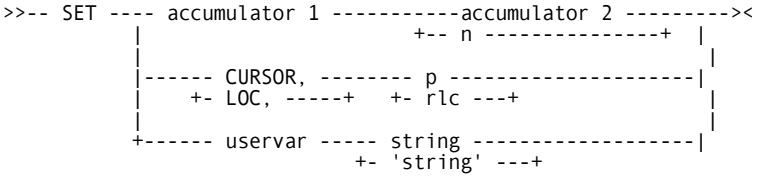
The current AID is contained in the &TAID variable. If you want to send a value other than the current value, use the ACL/E command that performs the action you want (for example, the ENTER or PF*n* commands). You can use the SET statement to set the value of &TAID to an action key and then use the SEND command.

SET Command

The SET command can set the value of an accumulator, the &CURSOR or &LOC variable, or a user defined variable.

Format

The SET command has the following syntax:



Operands

The SET command has the following operands:

Operand	Explanation
accumulator 1	Specifies the accumulator whose contents is set by the SET command.
accumulator 2	Specifies the accumulator whose value is assigned to the accumulator you specified in *accumulator 1*.
n	Specifies an integer value assigned to the accumulator specified in *accumulator 1*.
CURSOR	Assigns a specified screen location to the &CURSOR variable.
LOC	Assigns a specified screen location to the &LOC variable.
p	Specifies a screen location as an absolute screen position.
r/c	Specifies a screen location as a row and column number.
uservar	Specifies a user defined variable whose contents is set by the SET command.
string	Specifies a string assigned to the *uservar* you specified in the SET command.

SIGNOFF Command

The SIGNOFF command simulates the CA-TPX /F command.

Format

The SIGNOFF command has the following syntax:

```
>>---- SIGNOFF ----------------------------------------><
```

Usage

If you have specified F in the "Inactivate on" field of the user's definition, the sessions will not be inactivated by the SIGNOFF command. In this case you must use COMMAND 'F' instead of the SIGNOFF command.

SRCHWAIT Command

The SRCHWAIT command passes control to the next ACL/E instruction only if one of the following conditions occur:

- The specified string is found.

  In this case the condition code is set to EQ.

- The SNA RU indicates an "IN-BRACKET" state with the terminal owning direction and no data being sent by the application.

  In this case the condition code is set to NE.

Format

The SRCHWAIT command has the following syntax:

```
>>---- SRCHWAIT ----------- string --------------------><
                         +-- 'string' --+
```

Operands

The SRCHWAIT command has the following operand:

Operand	Explanation
string	Specifies the string that the SRCHWAIT command looks for.

Usage

The SRCHWAIT command combines the features of the SEARCH and WINP commands, and is intended to be used in an environment such as VM, CA-Remote Console, OMEGAMON or NetView, in which unsolicited RUs are arriving from the application.

STOP Command

The STOP command terminates execution of the ACL/E program. This command is not necessary at the end of the program, but can be used to terminate execution at some other place in the program.

Format

The STOP command has the following syntax:

```
>>---- STOP ------------------------------------------><
```

SUB Command

The SUB command subtracts a value from one of the four accumulators (A1 through A4).

Format

The SUB command has the following syntax:

```
>>--- SUB --- accumulator 1 --------accumulator 2 ----------><
                                |-- n ---------------|
                                +-- variable --------+
```

Operands

The SUB command has the following operands:

Operand	Explanation
accumulator 1	Specifies the accumulator to be decremented.
accumulator 2	Specifies the accumulator whose value is subtracted from the accumulator specified in *accumulator 1*.
n	Specifies an integer that is subtracted from the accumulator specified in *accumulator 1*.
variable	Specifies a numeric variable that is subtracted from the accumulator specified in *accumulator 1*.

SUSPEND Command

The SUSPEND command adds a specified number to the current SUSPEND count and suspends ACL/E execution if the new SUSPEND count is greater than the current NOTIFY count. Execution continues when enough NOTIFY events are received from other ACL/E programs to make the NOTIFY count greater than or equal to the SUSPEND count.

Format

The SUSPEND command has the following syntax:

```
>>---- SUSPEND ------- n ------------------------------><
```

Operands

The SUSPEND command has the following operand:

Operand	Explanation
n	Specifies the number added to the SUSPEND count.

Usage

The SUSPEND count is contained in the &SUSPNDCT variable and the NOTIFY count is contained in the &NOTIFYCT variable.

It is a good idea to use an OPTION TIME= statement in your program in case the ACL/E program issuing the NOTIFY events fails.

TABB Command

The TABB command simulates the back tab key and moves the cursor to the beginning of the current input field (if it is in an input field) and then to previous input fields. The cursor is always moved to the first position of the last input field it has moved to.

Format

The TABB command has the following syntax:

```
>>---- TABB ------- n --------------------------------><
```

Operands

The TABB command has the following operand:

Operand	Explanation
n	Specifies how many times the TABB command moves the cursor.

TABF Command

The TABF command moves the cursor forward a specified number of input fields. The cursor is moved to the first position of the target field.

Format

The TABF command has the following syntax:

```
>>---- TABF ------- n ---------------------------------><
```

Operands

The TABF command has the following operand:

Operand	Explanation
n	Specifies how many times the TABF command moves the cursor.

TABL Command

The TABL command moves the cursor to the last input field on the current screen.

Format

The TABL command has the following syntax:

```
>>---- TABL ----------------------------------------><
```

TERMSESS Command

The TERMSESS command immediately terminates the application session associated with the executing program. This command can be used in an ACL/E logoff program when unexpected conditions arise.

Format

The TERMSESS command has the following syntax:

```
>>---- TERMSESS ----------------------------------><
```

UDEFINE Command

The UDEFINE command defines a user-level (persistent) variable. This variable can be set or referenced by any ACL/E program on any session belonging to this CA-TPX user but is deleted when the user signs off.

Format

The UDEFINE command has the following syntax:

```
>>---- UDEFINE --- varname,n ------------------------------------------------->
<
                                  +-TYPE= ----- CHAR ------------------------+
                                  |             |            +- NOTRUNC ----+  |
                                  |             |                           |  |
                                  |             |-- UPCHAR -----------------|  |
                                  |             |            +- NOTRUNC ----+  |
                                  |             |                           |  |
                                  |             |-- NUM --------------------|  |
                                  |             |-- BIN --------------------|  |
                                  |             +-- HEX --------------------+  |
```

Operands

The UDEFINE command has the following operands:

Operand	Explanation
varname	Specifies the name of the transient variable defined in the UDEFINE command.
n	Specifies the length of the variable. The *default* length is eight.
TYPE=	Specifies the type of variable defined in the UDEFINE command. Must specify CHAR, UPCHAR, NUM, BIN, or HEX.
CHAR	Indicates that the variable is a character variable.
UPCHAR	Indicates that the variable is an uppercase character variable.
NUM	Indicates that the variable is a numeric variable.
BIN	Indicates that the variable is a binary variable.
HEX	Indicates that the variable is a hexadecimal variable.
NOTRUNC	Indicates that any trailing blanks in the variable will not be deleted when variable substitution takes place. This operand can be used only with the CHAR and UPCHAR operands.

UDELETE Command

The UDELETE command deletes a user-level variable.

Format

The UDELETE command has the following syntax:

```
>>------ UDELETE ------ varname --------------------><
```

Operands

The UDELETE command has the following operand:

Operand	Explanation
varname	Specifies the name of the user-level variable deleted by the UDELETE command.

UP Command

The UP command moves the cursor up the screen. If the cursor reaches the top of the screen, it wraps and continues upwards from the bottom of the screen.

Format

The UP command has the following syntax:

```
>>------ UP ------ n --------------------------------><
```

Operands

The UP command has the following operand:

Operand	Explanation
n	Specifies how many lines the cursor is moved by the UP command.

VDEFINE Command

The VDEFINE command defines an ACL/E-level (transient) variable. This variable can be set or referenced by programs called by the current ACL/E program but is automatically deleted when the current program terminates.

Format

The VDEFINE command has the following syntax:

```
>>-- VDEFINE --- varname,n ------------------------------------------------------><
                                +-TYPE= ----- CHAR -------------------------+
                                            |           +- NOTRUNC ----+      |
                                            |                          |      |
                                            |-- UPCHAR ----------------------|
                                            |           +- NOTRUNC ----+      |
                                            |                          |      |
                                            |-- NUM --------------------------|
                                            |-- BIN --------------------------|
                                            +-- HEX --------------------------+
```

Operands

The VDEFINE command has the following operands:

Operand	Explanation
varname	Specifies the name of the transient variable defined in the VDEFINE command.
n	Specifies the length of the variable. The *default* length is eight.
TYPE=	Specifies the type of variable defined in the VDEFINE command. Must specify CHAR, UPCHAR, NUM, BIN, or HEX.
CHAR	Indicates that the variable is a character variable.
UPCHAR	Indicates that the variable is an uppercase character variable.
NUM	Indicates that the variable is a numeric variable.
BIN	Indicates that the variable is a binary variable.
HEX	Indicates that the variable is a hexadecimal variable.
NOTRUNC	Indicates that any trailing blanks in the variable will not be deleted when variable substitution takes place. This operand can be used only with the CHAR and UPCHAR operands.

VDELETE Command

The VDELETE command deletes a transient variable.

Format

The VDELETE command has the following syntax:

```
>>------ VDELETE ------ varname -------------------------><
```

Operands

The VDELETE command has the following operand:

Operand	Explanation
varname	Specifies the name of the transient variable deleted in the VDELETE command.

WAIT Command

The WAIT command suspends program execution for a specified number of seconds. Output from the application still appears on the screen during this period if OPTION TERM is set to ON.

Format

The WAIT command has the following syntax:

```
>>------ WAIT ------ n ----------------------------------><
```

Operands

The WAIT command has the following operand:

Operand	Explanation
n	Specifies how many seconds the WAIT command suspends program execution.

WINP Command

The WINP command causes the ACL/E program to wait for data from the application.

Format

The WINP command has the following syntax:

```
>>------ WINP ---------------------------------------><
```

Appendix E: Predefined Variables

This appendix lists the ACL/E predefined variables and their attributes.

For a list of all variables that do not have operands, see the Predefined Variables (see page 157) table in this chapter. For a list of variables that return information about a user-specified session or application, see the Predefined Function Variables (see page 161) table in this chapter. The variables are listed in alphabetical order.

This section contains the following topics:

Predefined Variables

The following table lists the ACL/E predefined variables that do not require an operand.

Variable	Function
&A*n*	This variable represents any of four accumulators, &A1 through &A4, which can contain any positive integer from 0 through 16,777,215 (16M-1). Accumulators can be set, added into, subtracted from, and compared against other values, or included in any string as &A*n* (where n equals 1, 2, 3, or 4). When used in a string this way (for example, 487&A1.ABC), the value of &A1 is substituted into the string before the statement is executed. For instance, if &A1=333 in the last example, the value of the string would be 487333ABC.
&AID	This variable represents the attention identifier byte from the last action taken.
&APPL	This variable represents the VTAM APPLID of the session in which the ACL/E is running.
&APPLID	This variable represents the VTAM APPLID of the session in which the ACL/E is running. For applications that issue CLSDST PASS, such as TSO, this variable can have a different value than &APPL.
&CSRCOL	This variable represents the column value of the &CURSOR variable.
&CSRROW	This variable represents the row value of the &CURSOR variable.

Variable	Function
&CURSOR	This variable represents the current cursor location as a screen position. Row 1, Column 1 is position 1. On an 80 by 24 display, Row 2, Column 1 is position 81, and so on. Use of &CURSOR as the receiving field of a SET command causes cursor movement within the current screen image.
&ERR_MSG	This variable contains the text of the error message that occurred during ACL/E processing, and is set when you use the ERROR operand on the OPTION command.
&ERR_RC	This variable contains the return code of the error that occurred during ACL/E processing, and is set when you use the ERROR operand on the OPTION command.
&IPADDR	This variable represents either an IPv4 or IPv6 TCP/IP address. If this is an IPv4 address, it will be represented as a 4 octet address. If this is an IPv6 address, it will be represented as an 8, 16 bit, hexadecimal value address.
&IPPORT	This variable represents the 1 to 4 byte TCP/IP port number of the terminal connection, in decimal. Leading zeroes are suppressed.
&IPVECT	This variable represents the entire TCP/IP vector (x'64') from the CINIT.
&LOC	This variable represents a data location as a screen position. Row 1, column 1 is position 1. The SEARCH command sets &LOC to the screen position of the first byte of data that satisfies an equal condition. &LOC is used to determine the current field location for &SCRNFLD and for SEARCHF.
&LOCCOL	This variable contains the column value of the &LOC variable.
&LOCROW	This variable contains the row value of the &LOC variable.
&NETNAME	This variable represents the VTAM network name of the physical terminal where the current application session is active.
&NPSWD	This variable represents the new password the user defined after signing on. If the user kept the initial password, CA TPX assigns a value of two question marks (??) to this variable.
&NPWPH	This variable represents the new password phrase the user defined after signing on. If the user kept the initial password phrase, CA TPX assigns a value of two question marks (??) to this variable.
&OPSWD	This variable represents the password that the user initially employed when signing on to CA TPX.

Variable	Function
&OPWPH	This variable represents the password phrase that the user initially employed when signing on to CA TPX.
&Pn	These variables, &P1 through &P8, represent the respective items in the PARM parameter statement associated with this session, through User or Profile Maintenance. You also can use these variables as string variables or even as return codes by setting them as desired. As with all variables, you can include &P*n* variables within a string, and their values will be substituted before the statement is executed. &P0 has a special meaning. When you start an ACL/E program with the /S command, you can enter text to the right of the ACL/E program name. This text will become &P0 within the program. For example: /S ACCTACL DEPARTMENT A &P0 was given the value DEPARTMENT A. If &P variables are unused, they have a value of eight question marks.
&PRINTER	This is the eight-character name of the default printer for the owner of the session in which the ACL/E is running.
&PSWD	This variable represents the current password for the user. CA TPX assigns a value of two question marks (??) if "NONE" is specified in the "Security System" field in user maintenance.
&PWPH	This variable represents the current password phrase for the user. CA TPX assigns a value of two question marks (??) if "NONE" is specified in the "Security System" field in user maintenance.
&SCREEN	This variable represents the entire screen image. It is usually used with substring notation to limit the amount of data involved.
&SCRNCOLS	This is the number of columns on the terminal screen.
&SCRNFLD	This variable represents the data from &LOC to the end of the current field.
&SCRNROWS	This is the number of rows on the terminal screen.
&SCRNSIZE	This is the absolute screen size, in bytes. For example, 80 columns by 24 rows gives &SCRNSIZE a value of 1920.
&SID	This variable represents the session ID of the current session.

Variable	Function
&TAID	This is the last AID (attention identifier) byte received from the terminal in response to the last ACL/E INPUT command. &TAID defaults to ENTER if no INPUT command has been issued. You can use the SET command to assign any valid AID value (PF1-24, CLEAR, PA1-3, or ENTER) to &TAID prior to issuing a SEND command.
&TCSRCOL	This variable represents the column value of &TCURSOR.
&TCSRROW	This variable represents the row value of &TCURSOR.
&TCURSOR	This variable represents the current cursor location of the alternate screen image as a screen position. Row 1, column 1 is position 1. On an 80 by 24 display, row 2,column 1 is position 81, and so on.
&USERID	This variable represents either the user's signon ID or the value specified in the "ACL Userid" field in User or Profile Maintenance.
&VTERM	This variable represents the name of the virtual terminal associated with the session.
&ZAPPLID	This variable represents the VTAM name (applid) of the current CA TPX.
&ZCONN	This variable represents the session ID of the currently connected session (the session currently being displayed on the terminal).
&ZDATE	This variable represents the U.S. format date (*mm/dd/yy*).
&ZEDATE	This variable represents the European format date (*dd/mm/yy*).
&ZJDATE	This variable represents the Julian format date (*yyddd*).
&ZLDATE	This variable represents the U.S. long format date (*mm/dd/yyyy*).
&ZLEDATE	This variable represents the European long format date (*dd/mm/yyyy*).
&ZLJDATE	This variable represents the Julian long format date (*yyyyddd*).
&ZMODEL	This variable represents the terminal model type (for example, 3278-2).
&ZNEWS1	This variable is used on the LOGO panel for distributing a brief message to users.
&ZNEWS2	This variable is used on the LOGO panel for distributing a brief message to users.
&ZSMFID	This variable represents the SMF ID of the z/OS system running CA TPX. This is a system parameter for GCS.

Variable	Function
&ZSYSID	This variable represents the name of the System Options Table (SMRT) currently in use by CA TPX.
&ZTIME	This variable represents the current time.
&ZTERMID	This variable represents VTAM Terminal ID.

Predefined Function Variables

The following table lists the ACL/E predefined variables that are functions that return information about a user-specified session or application.

Variable	Function
&APPL_INF(*applID*)	This variable provides information about the specified application. The variable can return one or more of the following values: **ACTIVE**—Indicates that the application is defined and is accepting logons. **INACT**—Indicates that the application is defined but is not accepting logon. **QUIESCED**—Indicates that the application has one or more active sessions. New sessions will not be started. **TYPE=TPX**—Indicates that the application is an internal application such as TPXMAIL or TPXVIEW. **UNDEFINED**—Indicates that the application is not defined to CA TPX.
&MSIZE_OF(*string*)	This variable represents the maximum possible length of the specified string after variables in the string have been substituted.

Variable	Function
&SESS_INF(*sessionID*)	This variable represents information about the specified session. It can return one or more of the following values: **ACTIVE**—Indicates that the session is currently active. **ACL=program**—Indicates that the session is active and the specified ACL/E program is running in the session. **APPL=appl**—Indicates that the specified application is running the session. **CURRENT**—Indicates that the session is the user's current session. **INACT**—Indicates that the session is not currently active, although it has been active during the user's current session with CA TPX. **NEVACT**—Indicates that the session has never been active during the user's current session with CA TPX. **UNDEFINED**—The specified session is not a valid session name.
&SIZE_OF(*string*)	This variable represents the length of the specified string after all variables in that string have been substituted.
&TYPE_OF(*variable*)	This variable represents the specified variable's type. This variable can return one or more of the following values: **ACL**—Indicates that the variable was defined in ACL/E. **ARRAY**—Indicates that this is an array variable. **BIN**—Indicates that this is a binary variable. **CHAR**—Indicates that this is a character variable. **HEX**—Indicates that this is a hexadecimal variable. **NOTRUNC**—Indicates that the variable will not be stripped of trailing blanks when substitution takes place. **NUM**—Indicates that this is a numeric variable. **UNDEFINED**—Indicates that the variable is not defined. **UPCHAR**—Indicates that this is an uppercase character variable.

Appendix F: Return Codes

This appendix lists the ACL/E return codes that are issued when an ACL/E processing error occurs.

This section contains the following topics:

Return Codes and Messages

When you use the OPTION statement with the ERROR operand, the variable &ERR_RC is set to the value of the return code and the variable &ERR_MSG is set to the value of the corresponding message text.

Return Code List

The following table lists the ACL/E return codes that are issued when an ACL/E processing error occurs. When you use the OPTION statement with the ERROR operand, the variable &ERR_RC is set to the value of the return code and the variable &ERR_MSG is set to the value of the corresponding message text.

Return Code	Message Text
RC_01	SCRIPT EXCEEDED INSTRUCTION EXECUTION LIMIT
RC_02	NO OPCODE ENCOUNTERED IN STATEMENT
RC_03	STATEMENT CONTAINS AN INVALID OPCODE
RC_04	-ADD- OPERATION CONTAINED INVALID OPERAND 1
RC_05	-ADD- OPERATION CONTAINED INVALID OPERAND 1
RC_06	-ADD- OPERATION CONTAINED INVALID OPERAND 2
RC_07	-ADD- OPERATION CONTAINED INVALID OPERAND 2
RC_08	-ADD- OPERATION CONTAINED INVALID OPERAND 2
RC_09	-BRANCH- OPERATION CONTAINED AN INVALID OPERAND 1
RC_10	-BRANCH- OPERATION CONTAINED AN INVALID OPERAND 2
RC_11	-BRANCH- OPERATION CONTAINED AN INVALID OPERAND 2
RC_12	-BRANCH- OPERATION CONTAINED AN INVALID OPERAND 2

Return Code	Message Text
RC_13	-COMPARE- OPERATION CONTAINED AN INVALID OPERAND 1
RC_14	-COMPARE- OPERATION CONTAINED AN INVALID OPERAND 1
RC_15	-COMPARE- OPERATION CONTAINED AN INVALID OPERAND 1
RC_16	-COMPARE- OPERATION CONTAINED AN INVALID OPERAND 2
RC_17	-COMPARE- OPERATION CONTAINED AN INVALID OPERAND 2
RC_18	-COMPARE- OPERATION CONTAINED AN INVALID OPERAND 2
RC_19	UNUSED
RC_20	-DOWN- OPERATION CONTAINED AN INVALID OPERAND 1
RC_21	-DOWN- OPERATION CONTAINED AN INVALID OPERAND 2
RC_22	UNUSED
RC_23	UNUSED
RC_24	OPERATION CONTAINED AN INVALID OPERAND 1
RC_25	-KEY- OPERATION ENCOUNTERED AN OUT-OF-RANGE CONDITION
RC_26	-LEFT- OPERATION CONTAINED AN INVALID OPERAND 1
RC_27	-LEFT- OPERATION CONTAINED AN INVALID OPERAND 1
RC_28	-MSG- OPERATION CONTAINED AN INVALID OPERAND 1
RC_29	-OPTION- OPERATION CONTAINED AN INVALID OPERAND 1
RC_30	-OPTION- OPERATION CONTAINED AN INVALID OPERAND 2
RC_31	-OPTION- OPERATION CONTAINED AN INVALID OPERAND 2
RC_32	-OPTION- OPERATION CONTAINED AN INVALID OPERAND 2
RC_33	-OPTION- OPERATION CONTAINED AN INVALID OPERAND 2
RC_34	-PA- OPERATION CONTAINED AN INVALID OPERAND 1
RC_35	-PA- OPERATION CONTAINED AN INVALID OPERAND 1
RC_36-RC_39	UNUSED
RC_40	-PF- OPERATION CONTAINED AN INVALID OPERAND 1
RC_41	-PF- OPERATION CONTAINED AN INVALID OPERAND 1
RC_42	-RIGHT- OPERATION CONTAINED AN INVALID OPERAND 1
RC_43	-RIGHT- OPERATION CONTAINED AN INVALID OPERAND 1
RC_44	UNUSED
RC_45	-SCRIPT- OPERATION CONTAINED AN INVALID OPERAND 1

Return Code	Message Text
RC_46	-SCRIPT- OPERATION CONTAINED AN INVALID OPERAND 1
RC_47	UNUSED
RC_48	-SEARCH- OPERATION CONTAINED AN INVALID OPERAND 1
RC_49	UNUSED
RC_50	-SET- OPERATION CONTAINED AN INVALID OPERAND 1
RC_51	-SET- OPERATION CONTAINED AN INVALID OPERAND 1
RC_52	-SET- OPERATION CONTAINED AN INVALID OPERAND 2
RC_53	-SET- OPERATION CONTAINED AN INVALID OPERAND 2
RC_54	-SET- OPERATION CONTAINED AN INVALID OPERAND 2
RC_55	VARIABLE SUBSCRIPT TOO LARGE OR NON-NUMERIC
RC_56	UNUSED
RC_57	UNUSED
RC_58	SCRIPT TERMINATED DUE TO RUNAWAY CONDITION
RC_59	ACL TIMEOUT ERROR - INVALID SB POINTER IN SEB
RC_60	-SUB- OPERATION CONTAINED AN INVALID OPERAND 1
RC_61	-SUB- OPERATION CONTAINED AN INVALID OPERAND 1
RC_62	-SUB- OPERATION CONTAINED AN INVALID OPERAND 2
RC_63	-SUB- OPERATION CONTAINED AN INVALID OPERAND 2
RC_64	-SUB- OPERATION CONTAINED AN INVALID OPERAND 2
RC_65	UNUSED
RC_66	-UP- OPERATION CONTAINED AN INVALID OPERAND 1
RC_67	UNUSED
RC_68	-UP- OPERATION CONTAINED AN INVALID OPERAND 1
RC_69	UNUSED
RC_70	-EOF- OPERATION ENCOUNTERED OUT-OF-RANGE CONDITION
RC_71	-DELETE- OPERATION ENCOUNTERED OUT OF RANGE CONDITION
RC_72	-EOF- TRIED DOING EOF ON A PROTECTED FIELD
RC_73	UNUSED
RC_74	-OPTION- TIMEOUT VALUE IS NOT NUMERIC
RC_75	-OPTION- TIMEOUT ADDRESS IS NOT A VALID LABEL

Return Code	Message Text
RC_76-RC_85	UNUSED
RC_86	-TABF- OPERATION CONTAINED AN INVALID OPERAND 1
RC_87	UNUSED
RC_88	-TABF- OPERATION CONTAINED AN INVALID OPERAND 1
RC_89	UNUSED
RC_90	-COMMAND- OPERAND-1 IS NOT A VALID TPX COMMAND
RC_91	-COMMAND- OPERAND-1 IS NOT A VALID TPX COMMAND
RC_92-RC_95	UNUSED
RC_96	-TABB- OPERATION CONTAINED AN INVALID OPERAND 1
RC_97	UNUSED
RC_98	-TABB- OPERATION CONTAINED AN INVALID OPERAND 1
RC_99	ACLPGM STARTED ANOTHER ACLPGM FOR INVALID SESSION
RC100	UNUSED
RC101	-DISPLAY- PANEL VIOLATES NAMING CONVENTIONS
RC102	-DISPLAY- PANEL NAME NOT FOUND IN LIBRARY
RC103	-DISPLAY- PANEL NOT BUILT DUE TO DEFINITION ERRORS
RC104	-*DEFINE- OPERAND-1 INVALID
RC105	-*DEFINE- OPERAND-2 INVALID
RC106	-*DEFINE- OPERAND-2 IS NOT NUMERIC
RC107	-*DEFINE- OPERATION FAILED
RC108	-BRANCH- OPERATION CONTAINED AN INVALID OPERAND 1
RC109	UNUSED
RC110	-*DELETE- OPERAND-1 INVALID
RC111	-*DELETE- OPERATION FAILED
RC112	-NL- OPERATION CONTAINED AN INVALID OPERAND 1
RC113	-NL- OPERATION CONTAINED AN INVALID OPERAND 1
RC114	-OPTION- OPERAND-1 INVALID
RC115	OPERAND-2 INVALID
RC116	-SET- INVALID ROW/COLUMN SPECIFICATION
RC117	OPERAND-1 IS NOT NUMERIC

Return Code	Message Text
RC118	OPERAND-1 INVALID
RC119	OPERAND-2 INVALID
RC120	-NOTIFY- OPERAND-1 IS NOT A VALID SESSION NAME
RC121	-NOTIFY- OPERAND-1 IS NOT AN ACTIVE SESSION
RC122	****** ACL/E FUNCTIONS NOT AUTHORIZED ******
RC123	UNUSED
RC124	-SUBSTRING- NON-NUMERIC VALUE SPECIFIED
RC125	-VARIABLE RESOLUTION OVERFLOW. CALL TPX SUPPORT.
RC126	SMRT OPTION 22 DOES NOT ALLOW ACL ON SMV1 SESSION
RC127-RC255	UNUSED

Index

D

E

G

H

I

K

L

W

X