SWAT™ Debugger
User's Manual

093-000258-01

For the latest enhancements, cautions, documentation changes, and other information on this product, please see the Release Notice (085-series) supplied with the software

Ordering No. 093-000258

Printed in the United States of America
Revision 01, May 1982

093-000258-01

Preface

The SWAT™ debugger is a high-level, interactive symbolic debugging system. It allows you to debug high-level language programs. You can use the SWAT debugger to validate a program at the source language level, rather than at the assembly or machine language level.

Who Should Read This Manual?

We have written this manual for any programmer who is developing and testing programs in one or more of the high-level languages that the SWAT debugger supports. We assume you have a good working knowledge of the operating system and your programming language. Realizing that many programmers have never used a high-level debugger before, we designed this manual to serve as a tutorial aid for the learner, and as a reference for those with more debugging experience.

We have organized the manual as follows:

Chapter 1	Introduces the SWAT debugger. It tells you what the SWAT debugger is, what it can do, and explains SWAT concepts.
Chapter 2	Shows you how to get the SWAT debugger running. It describes what you'll need to do before sitting down to debug a program.
Chapter 3	Presents a sample debugging session that introduces you to all the SWAT commands. We explain what the SWAT commands do, and demonstrate how to use them.
Chapter 4	Describes the SWAT commands in detail. This is a reference section that the experienced SWAT user can use to find answers to questions about the commands.
Appendix A	Provides a number of troubleshooting tips. We list things to check when you have trouble invoking or using the SWAT debugger. It also describes some of the differences between the AOS and AOS/VS SWAT debuggers.
Appendix B	Describes the errors that may occur during a SWAT debugging session. We give a full explanation for all error messages you may encounter.
Appendix C	Presents additional sample programs and audit files.

This manual is a revision, replacing the previous manual (093-258-00). A vertical bar in the outside margin indicates a change or addition to the technical information. An asterisk flags a substantial deletion.

Related Manuals

As we said earlier, we assume you are familiar with the operating system and the programming language you are using. To supplement your reading of this manual, you may want to refer to one or more of the following:

AOS Programmer's Manual (093-000120)
AOS Debugger and Disk File Editor User's Manual (093-000195)
AOS Link User's Manual (093-000254)

NOTICE

SWAT™ Debugger
User's Manual
093-000258

Revision History:	Effective with:
Original Release - September 1980	
First Revision - May 1982	(SWAT™ Rev. 2.0) (AOS/VS Rev. 2.0) (AOS Rev. 3.30)
Addendum 086-000045-00 - November 1982	(AOS SWAT™ Rev 2.1) (AOS/VS SWAT™ Rev 2.2)

A vertical bar or an asterisk in the margin of a page indicates substantive change or deletion, respectively from the previous revision.

AOS/VS Programmer's Manual (093-000241)
AOS/VS Link and Library File Editor User's Manual (093-000245)
AOS/VS Debugger and File Editor User's Manual (093-000246)

Command Line Interpreter User's Manual (AOS, AOS/VS) (093-000122)

COBOL Reference Manual (AOS/VS) (093-000289)

FORTRAN 77 Reference Manual (093-000162)

PL/I Reference Manual (AOS) (093-000204)
PL/I Reference Manual (AOS/VS) (093-000270)

AOS/VS PASCAL Reference Manual (093-000290)
AOS/VS C Language Reference and Runtime Manual (093-000264)

What Do You Think?

At the end of this manual you'll find a Remarks Form. This is your direct line to us in User Documentation — please take advantage of it. We want to know what you like and dislike about the manual. We welcome your suggestions, and *we really listen!* Only when the manual does its job can it help you do yours. So, help us help you.

Reader, Please Note:

We use these conventions for command formats in this manual:

COMMAND required *[optional]* ...

Where	Means
COMMAND	You must enter the command (or its accepted abbreviation) as shown.
required	You must enter some argument (such as a filename). Sometimes, we use: $\{\text{required}_1 \atop \text{required}_2\}$ which means you must enter *one* of the arguments. Don't enter the braces; they only set off the choice.
[optional]	You have the option of entering this argument. Don't enter the brackets; they only set off what's optional.
...	You may repeat the preceding entry or entries. The explanation will tell you exactly what you may repeat.

Additionally, we use certain symbols in special ways:

Symbol	Means
)	Press the NEW LINE or carriage return (CR) key on your terminal's keyboard.
□	Be sure to put a space here. (We use this only when we must; normally, you can see where to put spaces.)

All numbers are decimal unless we indicate otherwise; e.g., 35_8.

Finally, in examples we use

THIS TYPEFACE TO SHOW YOUR ENTRY)
THIS TYPEFACE FOR SYSTEM QUERIES AND RESPONSES.

) is the CLI prompt.

Contacting Data General

- If you have comments on this manual, please use the prepaid Remarks Form that appears after the Index. We want to know what you like and dislike about this manual.
- If you need additional manuals, please use the enclosed TIPS order form (USA only) or contact your Data General sales representative.
- If you experience software problems, please notify Data General Systems Engineering.

End of Preface

Contents

Chapter 1 - Introduction

What Is the SWAT™ Debugger? . . . 1-1
SWAT Features . . . 1-1
Revising Your Source File . . . 1-2
SWAT Concepts . . . 1-2
- Locators . . . 1-3
  - Clauses . . . 1-3
- Environments . . . 1-3
  - Procedure Blocks and Program Modules . . . 1-3
  - The Scope of Program Elements . . . 1-5
  - Environment Specifiers . . . 1-6
  - Fully Qualified Environment References . . . 1-7
  - The Breakpoint Environment . . . 1-8

Expressions . . . 1-8
- Operators . . . 1-9
- Data Type Conversions . . . 1-10

Chapter 2 - How to Run the SWAT Debugger

Required Software . . . 2-1
Installing the SWAT Debugger . . . 2-1
SWAT User Privileges . . . 2-1
Preparing a Program For Debugging . . . 2-2
- Compiling Your Program . . . 2-2
- Linking Your Program . . . 2-2
- Calling the SWAT Debugger . . . 2-3
- The Audit File . . . 2-3
- Generic Files . . . 2-5
- The User Debugger . . . 2-5
- Running the SWAT Debugger . . . 2-5

SWAT Commands and Comments . . . 2-6
Console Interrupts and Control Commands . . . 2-7
Chaining . . . 2-7
SWAT Error Handling . . . 2-8

Chapter 3 - A Sample Debugging Session

The Program . . . 3-1
- What It Does . . . 3-1

Starting the Debugging Session . . . 3-12
- The Prompt . . . 3-12
- Auditing . . . 3-13
- Making Comments . . . 3-13
- The User Debugger . . . 3-13
- The Working Environment . . . 3-14
- Breakpoints . . . 3-14
  - Proceed Count . . . 3-16
  - Action String . . . 3-16

Clearing Breakpoints ... 3-17
Listing Lines from the Source Code ... 3-17
Running the Program ... 3-19
The Breakpoint Environment ... 3-19
Examining Variables ... 3-24
Displaying Expression Results ... 3-25
Setting a Variable's Value ... 3-26
Displaying Information About Program Symbols ... 3-27
When You Need Help ... 3-27
Interpreting Error Codes ... 3-28
The SWAT Debugger and the CLI ... 3-29
The Directory and Search List ... 3-29
SWAT Command Files ... 3-30
The Null Command Response ... 3-30
Ending the Session ... 3-31
Debugging the Exchange Program ... 3-31

Chapter 4 - The SWAT Commands

Using Upper- and Lowercase ... 4-2
The SWAT Prompt ... 4-2
Delimiting Commands and Arguments ... 4-2
The Null Command ... 4-3
Abbreviating SWAT Commands ... 4-3
Locators ... 4-3
Specifying Environments ... 4-4
Placing Comments in the Audit File ... 4-4
Editing the Command Line ... 4-5
AUDIT ... 4-6
BREAKPOINT ... 4-8
BYE ... 4-11
CLEAR ... 4-12
CLI ... 4-14
CONTINUE ... 4-16
DEBUG ... 4-19
DESCRIBE ... 4-21
DIRECTORY ... 4-23
ENVIRONMENT ... 4-24
EXECUTE ... 4-26
HELP ... 4-27
LIST ... 4-29
MESSAGE ... 4-32
PREFIX ... 4-33
PROMPT ... 4-34
SEARCHLIST ... 4-35
SET ... 4-37
TYPE ... 4-40
WALKBACK ... 4-44

Appendix A - Troubleshooting Tips (Or What to Do if You're Having Trouble)

SWAT Check List A-1
Using the AOS/VS SWAT Debugger A-3
Specific Language Considerations A-3
Common User Errors A-4
Helpful Hints A-5
- Help Information from the CLI A-5
- Calling the User Debugger A-6
- Using Screenedit Features A-6
- Defining Extra Variables A-6
- Using Pointer Arithmetic A-6
- Simulating CLI Pseudo-Macros A-6

Appendix B - SWAT Error Messages

SWAT Command Line Errors B-1
Start-Up and Termination Errors B-4
- Start-Up Errors B-4
- Termination Errors B-5

SWAT Maintenance Errors B-6
System Error Messages B-6

Appendix C - More Examples

A Sample FORTRAN 77 Program and Audit File C-17
A Sample COBOL Program and Audit File C-32
A Sample PASCAL Program and Audit File C-42
A Sample C Program and Audit File C-58

Illustrations

Figure	Caption	
1-1	SWAT Locators	1-4
1-2	Program Units: Program Modules and Procedure Blocks	1-5
1-3	Using the Environment Specifier	1-7
3-1	Compilation Listing of the Module EXCHANGE	3-3
3-2	Compilation Listing of the Module CONVERSION	3-8
3-3	Flow Chart of the EXCHANGE Program	3-11
3-4	The Audit File EXCHANGE.AU.	3-42
3-5	The Audit File AUDIT.YEN.	3-50
4-1	The Help Menu	4-27
C-1	Compilation Listing of the Module EXCHANGE (AOS/VS)	C-1
C-2	Compilation Listing of the Module CONVERSION (AOS/VS)	C-4
C-3	Audit File of the EXCHANGE Program SWAT Session	C-6
C-4	Compilation Listing of the Loan Program (Main Routine).	C-18
C-5	Compilation Listing of the Subroutine PAYMENT.	C-20
C-6	Compilation Listing of the Subroutine FULL.	C-21
C-7	Audit File of the LOAN Program SWAT Session	C-23
C-8	Compilation Listing of the MORTPROG Program	C-33
C-9	Audit File of the MORTPROG SWAT Session	C-37

Tables

Table	Caption	
1-1	The SWAT Operators	1-9
1-2	Data Type Conversions	1-10
2-1	The SWAT Global Switches	2-4
2-2	SWAT Commands	2-6
4-1	SWAT Commands	4-1
4-2	Control Characters and Keys for Editing	4-5

Chapter 1
Introduction

What Is the SWAT™ Debugger?

The SWAT™ debugger is a utility that helps you detect and locate errors in high-level language programs. The SWAT debugger interacts with your program, allowing you to control and monitor the program's execution. Using the SWAT commands with a certain amount of ingenuity, you can locate errors in source level algorithms and program logic. The SWAT debugger does not automatically identify problem areas — that's not its purpose. It is a detective's tool; you are the investigator.

SWAT Features

To debug a program, you must be able to suspend its execution at any point and examine the current state of execution. A breakpoint is a debugging aid that you can attach to any executable statement. When program execution encounters a breakpoint, execution halts (*traps*) just before acting on that statement. Control then passes to the SWAT debugger.

To control program execution through the SWAT debugger, you can

- set a breakpoint at any executable statement,
- attach an optional breakpoint proceed count, which specifies the number of times the breakpoint must be encountered before the breakpoint signals a trap,
- attach an optional breakpoint action string, which specifies one or more SWAT commands to be performed when a trap occurs,
- display the locations where breakpoints are currently set, and
- clear a breakpoint from a statement.

Once execution traps, you can examine the current state of the program. For example, you can

- display or set the working environment (i.e., the scope of a particular program unit),
- display the value of a user-defined program symbol,
- display the result of an expression,
- set the value of a program variable, and
- display the locations of calling procedures.

Certain SWAT commands provide information that can help you in your debugging efforts. These commands let you

- list lines from the program's source file (if available),
- display a description of user-defined program symbols,
- obtain an explanation of a SWAT command or related topic,
- display the error message for a system error code, or
- set up an audit (log) file to record the debugging session.

You can perform special functions within the debugging session. For example, you can

- execute a previously created file of SWAT commands,
- perform one or more CLI functions within the SWAT session,
- enter the CLI process directly, or
- call the AOS or AOS/VS user debugger.

There are also a number of general purpose SWAT commands not directly related to debugging. These commands allow you to

- display or set the working directory,
- display or set the current search list,
- display or set the SWAT prompt, or
- display or set the SWAT response to the null command.

When resuming execution after a breakpoint trap, you can

- continue program execution from the current location, or
- alter program flow by redirecting execution.

When you've finished your debugging, you can then

- terminate the debugging session.

A SWAT session begins when you invoke the SWAT debugger and identify the program you are going to debug. When the SWAT prompt appears, the debugger is waiting for you to enter a command. During the session you can start and stop program execution. The program executes normally, unless you alter program flow or change the values of variables. When the program terminates, the SWAT debugger terminates. You can terminate both at any time by typing BYE.

Unless you choose otherwise, the SWAT debugger creates a copy of your program for the debugging session. This lets you debug a program and not worry about possibly altering the original program file.

All your breakpoints, adjusted variable values, working environment, and so on, are valid only for the duration of the debugging session. If you invoke the SWAT debugger again for the same program, you start in the main procedure with no breakpoints in effect.

*

The SWAT debugger can debug a chained series of programs. If the program you are debugging chains to another, a new debugging session begins (provided that the new program is one that the SWAT debugger can work with). The SWAT debugger terminates after the last program in the chain terminates.

If your 16-bit program uses overlays, don't worry; the SWAT debugger can handle them.

Revising Your Source File

After locating programming errors through the SWAT debugger, you must go back to the source file and make appropriate changes with a text editor. Then recompile and relink the new source file.

SWAT Concepts

To use the SWAT debugger, you must understand two SWAT concepts: the locator and the environment. The following sections explain these concepts and how you use them.

Locators

When in a debugging session, you often need to identify a certain program statement (where you want to set or clear a breakpoint, for example). To do this, you use a *locator*. A locator is either the statement's source file line number or a program label that you attached to the statement. When specifying a numeric label (as used in FORTRAN 77), precede it with an asterisk (*) to distinguish it from a line number. You can use an expression as a locator, provided that it resolves to a legal locator value.

Figure 1-1 illustrates sample locators in portions of source code.

Clauses

Some programming languages permit more than one statement on a single line of code. Some statements (such as IF/THEN) consist of distinct clauses. The SWAT debugger can recognize individual clauses within a statement, and statements within a line. To refer to a particular clause, you append an appropriate clause number to the line number locator. The language compiler you are using determines the rules for numbering clauses. For more information, refer to Appendix A, which describes debugging operations that are specific to a particular programming language.

Environments

Before getting into an explanation of SWAT environments, let's define a few terms. We'll be talking about various program elements, in particular, procedure blocks and program modules.

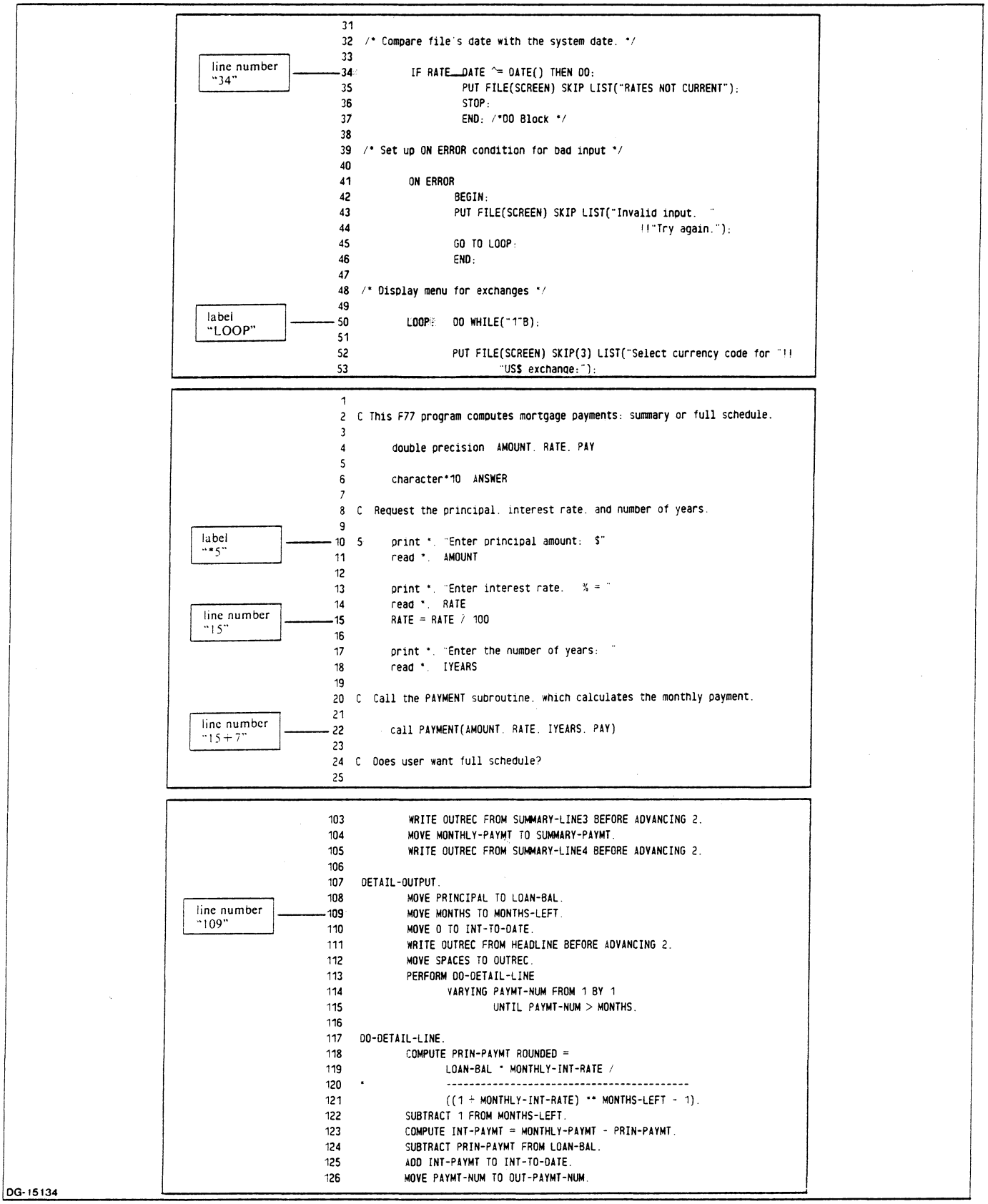
Procedure Blocks and Program Modules

A *procedure block* is a logically independent group of statements that act as a unit. Programs written in FORTRAN 77, for example, consist of one or more separate procedure blocks: the main program, each subroutine, and each subprogram.

PL/I and other block-structured languages allow nested procedure blocks. Procedure blocks that are not contained in any other block are called external procedure blocks. If embedded within a larger block, the procedure block is internal.

When building a program, you create one or more object files through the language compiler. Each object file is a separate *program module*. Depending on the language and the design of your program, a single program module can consist of one or more procedure blocks.

Figure 1-2 illustrates how program units fit together and how we refer to them.




```
31
32  /* Compare file's date with the system date. */
33
34          IF RATE_DATE ^= DATE() THEN DO;
35                  PUT FILE(SCREEN) SKIP LIST("RATES NOT CURRENT");
36                  STOP;
37                  END; /*DO Block */
38
39  /* Set up ON ERROR condition for bad input */
40
41          ON ERROR
42                  BEGIN;
43                  PUT FILE(SCREEN) SKIP LIST("Invalid input.  "
44                                                  !!"Try again.");
45                  GO TO LOOP;
46                  END;
47
48  /* Display menu for exchanges */
49
50          LOOP:   DO WHILE("1"B);
51
52                  PUT FILE(SCREEN) SKIP(3) LIST("Select currency code for "!!
53                          "US$ exchange:");
```

```
1
2   C This F77 program computes mortgage payments: summary or full schedule.
3
4         double precision  AMOUNT, RATE, PAY
5
6         character*10  ANSWER
7
8   C  Request the principal, interest rate, and number of years.
9
10  5     print *, "Enter principal amount:  $"
11        read *,  AMOUNT
12
13        print *, "Enter interest rate.   % = "
14        read *,  RATE
15        RATE = RATE / 100
16
17        print *, "Enter the number of years:  "
18        read *,  IYEARS
19
20  C  Call the PAYMENT subroutine, which calculates the monthly payment.
21
22        call PAYMENT(AMOUNT, RATE, IYEARS, PAY)
23
24  C  Does user want full schedule?
25
```

```
103           WRITE OUTREC FROM SUMMARY-LINE3 BEFORE ADVANCING 2.
104           MOVE MONTHLY-PAYMT TO SUMMARY-PAYMT.
105           WRITE OUTREC FROM SUMMARY-LINE4 BEFORE ADVANCING 2.
106
107   DETAIL-OUTPUT.
108           MOVE PRINCIPAL TO LOAN-BAL.
109           MOVE MONTHS TO MONTHS-LEFT.
110           MOVE 0 TO INT-TO-DATE.
111           WRITE OUTREC FROM HEADLINE BEFORE ADVANCING 2.
112           MOVE SPACES TO OUTREC.
113           PERFORM DO-DETAIL-LINE
114                 VARYING PAYMT-NUM FROM 1 BY 1
115                       UNTIL PAYMT-NUM > MONTHS.
116
117   DO-DETAIL-LINE.
118           COMPUTE PRIN-PAYMT ROUNDED =
119                 LOAN-BAL * MONTHLY-INT-RATE /
120   *             ------------------------------------------
121                 ((1 + MONTHLY-INT-RATE) ** MONTHS-LEFT - 1).
122           SUBTRACT 1 FROM MONTHS-LEFT.
123           COMPUTE INT-PAYMT = MONTHLY-PAYMT - PRIN-PAYMT.
124           SUBTRACT PRIN-PAYMT FROM LOAN-BAL.
125           ADD INT-PAYMT TO INT-TO-DATE.
126           MOVE PAYMT-NUM TO OUT-PAYMT-NUM.
```

Figure 1-1. SWAT Locators

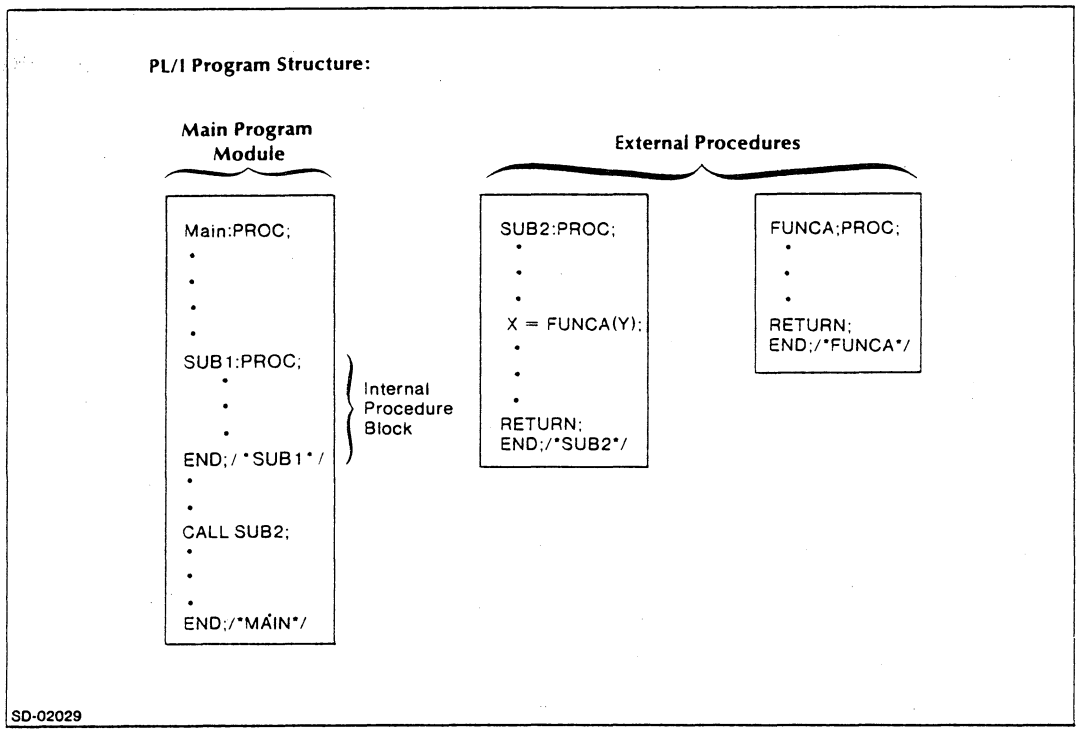


Figure 1-2. Program Units: Program Modules and Procedure Blocks

The Scope of Program Elements

Each user-defined symbol carries one and only one meaning within a certain portion of a program. The program structure (i.e., the arrangement of its procedure blocks) determines the scope of a symbol's meaning.

The scope of a program symbol extends from the procedure block in which it is defined to any inferior (embedded) block in which the same identifier is not redefined. The scope of a program symbol does not extend to an enclosing block or to an external block.

At any one moment, the SWAT debugger works within the scope of a particular procedure block. This scope is called the *working environment.*

When you refer to a program symbol, the SWAT debugger interprets the reference according to the working environment. If you specify a symbol that is unknown in the working environment, you receive an error message, unless you indicate the environment in which it is defined. (We describe how to do this shortly.) If a particular symbol has more than one meaning within the program, the debugger interprets the reference according to the symbol's meaning within the working environment.

When a SWAT debugging session begins, the working environment is that of the main procedure. You can change the working environment as necessary so that you can refer to program elements in various procedure blocks. The ENVIRONMENT command, which we describe later, lets you set the working environment. You supply an environment specifier as an argument to this command.

IMPORTANT: The initial identifying statement of an internal procedure block does not reside within that block, but is part of the containing block.

Environment Specifiers

It is often necessary to refer to program elements in different parts of a program, or to specify a particular block. To do this, use the ENVIRONMENT command, (which we explain later), with an *environment specifier*. An environment specifier identifies the procedure block whose scope you wish the SWAT debugger to emulate. The scope of the working environment determines which program symbols you can refer to directly.

The environment structure is very similar to the directory/file structure you use. Just as you specify a root directory with a colon (:), you use the same symbol to specify the root environment. The root environment contains all the external procedures in your program.

The general format for an environment specifier is

[:external-procedure] [[:]internal-procedure [:internal-procedure]...]

where:

external-procedure identifies a procedure block (such as a main program or subprogram) that is not contained within any other block.

internal-procedure identifies an embedded procedure block.

To identify an external or internal procedure, use its name, or if it is unnamed, its line number. (The main procedure of a FORTRAN 77 program is called .MAIN.)

An environment specifier can contain more than one internal procedure identifier, depending on how deeply embedded the block is.

Figure 1-3 diagrams a program that consists of three external procedures and a number of embedded internal blocks. In the explanations that follow, we use this example to illustrate the use of environment specifiers. Refer to this figure as you read.

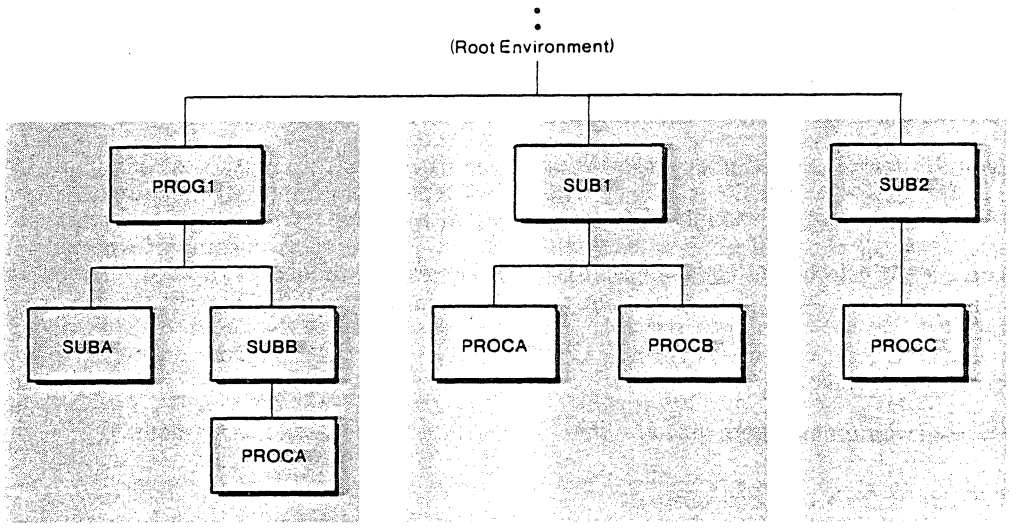
To identify an external procedure, begin the environment specifier with a colon, then append the name of the procedure block. This means that the procedure is located directly under the root environment. For example, the specifier :PROG1 refers to the main program module PROG1. The specifiers :SUB1 and :SUB2 identify the other two external blocks.

You can also use the keyword @MAIN to refer to the main program module. In our example, @MAIN identifies the procedure block PROG1.

How you specify an internal procedure depends on your current environment. If a procedure is contained immediately within the current procedure, you can identify this internal procedure simply by name (or line number, if it is unnamed). If the current environment is that of PROG1, for example, we can use the environment specifier SUBB to refer to that procedure. This is analogous to naming a file that is located in the current directory.

To refer to a procedure embedded at a lower level, begin the specifier with the immediately contained block, then append the necessary block names, separating each with a colon. If the current environment is that of PROG1, the specifier SUBB:PROCA identifies the PROCA block.

To specify an internal procedure in another external procedure, you must begin the environment specifier at the root environment, then work your way down to the desired block. If the current environment is somewhere within the PROG1 module, for example, the environment specifier :SUB2:PROCC identifies the internal block PROCC within the SUB2 procedure.



Beginning in the environment of the main program module PROG1, the following sequence of environment specifiers will move you to the environment indicated:

1. :SUB1	The procedure block SUB1.
2. PROCA	The procedure block PROCA under SUB1.
3. :SUB2:PROCC	The procedure block PROCC under SUB2.
4. ^	The procedure block SUB2.
5. @MAIN	The main program module, PROG1.
6. SUBB:PROCA	The procedure block PROCA under SUBB.
7. :PROG1:SUBA	The procedure block SUBA under PROG1.

SD-02030

Figure 1-3. Using the Environment Specifier

To refer to an immediately superior procedure block within the current module, you can use a caret (^) to begin the specifier. If you are located in the procedure block PROCA within the main program module, for example, the environment specifier ^^SUBA moves you two environments up (to PROG1), then down to the SUBA block located within PROG1.

Fully Qualified Environment References

Earlier we explained that you can use a line number or label to identify a statement. A simple line number locator refers to a source line within the current module. A simple statement label locator is meaningful only if the label is defined within the working environment.

You can overcome these restrictions by preceding the simple locator with an environment specifier. The result is a fully-qualified locator, which provides a complete pathname to the locator. Use the format

environment-specifier:locator

to refer to a line number or label that a simple locator cannot identify.

Looking again at Figure 1-3, let's assume that the working environment is the SUBB block under PROG1. A simple line number locator refers to a line within the current module (PROG1). If we use a simple label locator, the label must be known within the SUBB environment.

To refer to a line in the SUB2 module, we must fully qualify the locator. The simple locator 34, for example, points to line 34 in PROG1, the current module. To refer to line 34 in the SUB2 module, we must use the fully qualified locator :SUB2:34.

To identify a label that is unknown to the current environment, we must also fully qualify the locator. If the label LOOPER is defined within the PROCC block, for example, we need to use the locator :SUB2:PROCC:LOOPER to refer to that label.

For certain SWAT commands you need to specify program symbols. To refer to a symbol that is unknown in the working environment, you must similarly qualify the reference. Use the format

environment-specifier:symbol

to identify program elements that are undefined within the current environment.

The Breakpoint Environment

When program execution encounters a statement carrying a breakpoint, it halts before executing that statement. The working environment becomes the scope of the procedure block containing the statement where the trap occurred. This environment is called the breakpoint environment.

The SWAT debugger interprets all references to program symbols relative to the working environment. Thus, to refer to a symbol that is unknown in the current environment, you must fully qualify it.

By limiting your immediate scope to that of the working procedure block, the SWAT debugger allows you to see the referencing limitations in the block. You can determine which of similarly named variables a statement actually refers to.

The first statement of an embedded procedure block lies within the scope of the enclosing environment. If execution traps at such a statement, you cannot set or display the values of variables within the embedded block. Set a breakpoint at an executable statement within the block if you want access to its variables.

Expressions

Many SWAT commands accept expressions as arguments. An expression can consist of a user-defined symbol (variable or constant), a numeric literal, string constant, or a combination of variables, literals, constants, operators, and parentheses.

Label, file, character, and bit constants, for example, are acceptable expression elements. You can also use array elements, based or structure references, environment specifiers, and subexpressions.

An error occurs if the expression violates array bounds, includes incompatible data types, or is otherwise invalid.

A single variable or constant can serve as an expression. To build a complex expression, however, you need to use one or more operators.

You can use a comma to separate one argument from another. Although it is normally optional, a comma may become necessary to prevent multiple arguments from being interpreted as parts of a single expression, or to separate the expression from a command or keyword.

Operators

SWAT expressions can contain one or more operators, depending on the context of the expression.

Table 1-1 lists the operators that the SWAT debugger recognizes.

Table 1-1. The SWAT Operators

	Operator		Represents
Arithmetic Operators	+	(prefix)	unary plus
	−	(prefix)	unary minus
	+	(infix)	addition
	−	(infix)	subtraction
	*	(infix)	multiplication
	/	(infix)	division
	=	(infix)	assignment
	:=	(infix)	assignment
Logical Operators	&	(infix)	logical AND
	!	(infix)	logical OR (inclusive)
Environment Operators	^	(prefix)	superior procedure block
	:	(prefix)	root environment
	:	(infix)	block identifier separator
Structure Operator	.	(infix)	level separator
Pointer Operator	->	(infix)	based reference
Indirect Operator	^	(suffix)	indirect reference

Prefix operators precede an operand and refer to that operand only. For example: +45, −2.7, and ^EXCHANGE. Except for the caret (^), you can separate the operator and operand with one or more spaces.

Infix operators define a relationship between two operands. The operator appears between the operands. For example: 36+420, GROSS − TAXES, (85.0 * FACTOR / 250.0), LOGA&LOGB, 40R8->BP.

A suffix operator appears at the end of the operand to which it refers. The operator must follow its operand with no intervening spaces. For example: DIRECT^, 500R8^, (PTR+5)^.

Data Type Conversions

In general, the SWAT debugger accepts mixed mode expressions to the same extent as the programming language you are using. Therefore, follow the conventions of your programming language when building expressions in a SWAT debugging session.

When two operands of different data types are related through an operator, the operand of the lower ranking data type is converted to the data type of the other operand. The result is therefore that of the higher ranking data type. The precedence for implicit conversions is as follows:

Highest rank:	CHARACTER
	BIT
	CHARACTER CONSTANT
	FLOAT
	DECIMAL
	DISPLAY
Lowest rank:	BINARY

In certain cases, data conversion is not possible, so an error results. Table 1-2 describes the results of various data type conversions. The data types listed at the top of the table correspond to the right operand; those at the side correspond to the left operand. These conversions apply when using the arithmetic operators +, −, *, and /.

Table 1-2. Data Type Conversions

	BINARY	DISPLAY	DECIMAL	FLOAT	CHARACTER CONSTANT	BIT	CHARACTER
BINARY	—	DISPLAY	DECIMAL	FLOAT	error	error	error
DISPLAY	DISPLAY	DISPLAY[1]	DECIMAL	FLOAT	error	error	error
DECIMAL	DECIMAL	DECIMAL	DECIMAL[1]	FLOAT	error	error	error
FLOAT	FLOAT	FLOAT	FLOAT	—	error	error	error
CHARACTER CONSTANT	error	error	error	error	—	error	error
BIT	error	error	error	error	CHARACTER CONSTANT	—	error
CHARACTER	error	error	error	error	—	error	—

1. If both operands are either DECIMAL or DISPLAY, data type conversion occurs to set the precision and scale of the result.

End of Chapter

Chapter 2
How to Run the SWAT Debugger

Before using the SWAT debugger, you need to complete a few preliminary steps. This chapter outlines everything you must do.

Required Software

The SWAT debugger operates with only certain software revisions of the operating system, the language compiler, and the Link utility. Check the SWAT Release Notice, or ask your system manager if you are not sure if the SWAT debugger will work with your software. The release notice lists the earliest revisions of related software that support the SWAT debugger. If your system operates with these or later revisions, *and* if the software revisions are compatible with each other, you can run the SWAT software.

Installing the SWAT Debugger

The system manager loads the SWAT files from the release tape into your system. Included among these files is PARSWAT.SR and SWATERMES.SR, which contain SWAT error message information. Before you can run the SWAT debugger successfully, the system manager must use the EMASM macro to build the SWAT error message file, SWATERMES.OB. Finally, the system manager updates the system error message file, ERMES, by joining the SWAT error message object file with all others. See the SWAT Release Notice for additional information.

SWAT User Privileges

As a SWAT user, you need certain privileges before you can successfully execute the SWAT debugger. Your user profile must allow you to

- create a process without blocking
- create at least three son processes

If your user profile does not grant the necessary privileges, the error message

CALLER NOT PRIVILEGED FOR THIS ACTION

or

TOO MANY SUBORDINATE PROCESSES

appears when you try to run the SWAT debugger. Check with your system operator to ensure that you have the privileges outlined above. If you do not, the system operator can change your profile.

You may need to be able to create more than three son processes. If, for example, you intend to use the SWAT CLI command, you must be able to create at least four son processes. If any son process in turn creates other processes, your profile must allow you to create that many more.

Preparing a Program For Debugging

At this point you should have both access to the necessary SWAT files and the required user privileges. Now you follow three basic steps to begin a debugging session:

1. Compile each program module, using the /DEBUG switch with the modules you want to be able to debug.
2. Link the program modules and library files using the /DEBUG switch.
3. Call the SWAT debugger.

Before using an updated version of the SWAT debugger with programs that you debugged with an earlier revision, be sure to relink them (using the /DEBUG switch) so that the program file incorporates the latest SWAT software.

Compiling Your Program

The SWAT debugger requires special information not normally built into a program's object file. To provide this information, you append the /DEBUG switch to your compiler command. Use this general format:

XEQ { CC / COBOL / F77 / PASCAL / PL1 / other } /DEBUG//.../ program//*switch* .../

This command produces a program object file called program.OB.

You can use the SWAT debugger to examine a program module only if you used the /DEBUG switch when compiling the module. So, for programs with more than one module, use the /DEBUG switch with each one that you intend to debug. The program must contain at least one module compiled in this way.

Linking Your Program

The Link utility uses your compiled module(s) and library files to produce a program file. If you intend to debug the program file using the SWAT debugger, append the /DEBUG switch to the Link command. The Link utility then builds a debuggable program file and two additional files that the SWAT software needs.

You must include the SWAT interface file, SWATI.OB (or SWATI16.OB for the 16-bit SWAT debugger under AOS/VS), in your Link command line. Be sure that you have access to this file before calling the Link utility.

The general format for the Link command line is:

XEQ LINK/DEBUG//.../ program//.../ routines SWATI lang_libraries

IMPORTANT: The SWAT interface file (SWATI or SWATI16) must appear *before* all language library files in the Link command line, otherwise your program will execute without allowing you to debug it.

You can also use the Link macro provided by your programming language. For additional information, refer to the appropriate programmer's manual. Some examples of Link macros are:

CCL.CLI	for AOS/VS C programs
CLINK.CLI	for AOS/VS COBOL programs

F77LINK.CLI	for FORTRAN 77 programs (AOS and AOS/VS)
PASLINK.CLI	for AOS/VS PASCAL programs
PL1LINK.CLI	for PL/I programs (AOS and AOS/VS)

If you use these link macros, do not specify SWATI or SWATI16 as an argument; the macro includes this file for you when you use the /DEBUG switch.

When you include the /DEBUG switch in your Link command, the Link utility produces a program file called program.PR, which you can debug using the SWAT software. Link also creates two special files: the debugger lines file (program.DL) and the debugger symbols file (program.DS).

The debugger lines file includes data taken from the debugger lines blocks of your program's object file. It determines the runtime and source file locations of all program lines that reside in the modules you selected to debug. When you refer to a line or block during a debugging session, the SWAT debugger uses this file to resolve the reference.

The debugger symbols file includes data taken from the debugger symbols blocks of your program's object file. The debugger symbols block supplies standard relocation information for this data through relocation dictionary entries. The symbols file describes all names and blocks defined within the modules you selected to debug. When you refer to a name or block during the debugging session, the SWAT debugger uses this file to resolve the reference.

Even though you build a program file using the /DEBUG switch, you can execute the program without using the SWAT software. The debugger information generated by the /DEBUG switch does not interfere with normal program execution.

Calling the SWAT Debugger

You are now ready to execute the SWAT software and debug your program. The format of the SWAT invocation command line is as follows:

XEQ SWAT*[/global_switch ...]* program*[/switch ...] [argument[/switch ...]]*

where:

program	is the name of the program file you want to debug.
argument	is any argument required by your program.

You can append any appropriate switches to the program and its argument(s).

Table 2-1 lists the global switches that you can append to the SWAT invocation command. Some of these switches may require additional privileges, which you must grant to the SWAT debugger. By default, a son process has the same privileges as its father. Refer to the CLI User's Manual for additional information.

The Audit File

To keep a record of a debugging session dialog, you can set up an audit file. When the audit file is open, the SWAT debugger appends all debugging commands and responses to the file. During the debugging session, you can close the audit file, then later open the same or another audit file.

There are two ways to set up an audit file. You can use the /AUDIT switch when you invoke the SWAT debugger, or you can use the AUDIT command within a debugging session. In each case, the SWAT debugger uses the default audit file unless you provide the name of another file. The name of the default audit file is your program's name with the .AU extension.

Table 2-1. The SWAT Global Switches

Switch	Use
/AUDIT/=*pathname*/	Assigns an audit file and turns auditing on. The default audit file is program.AU. If the file does not exist, the SWAT debugger creates it; otherwise, the debugger appends the audit dialog to the file.
/BREAK	Creates a break file on abnormal termination of your program. (AOS/VS only — see note below.)
/CALLS=n	Sets the maximum number of concurrent system calls for the program.
/CONSOLE=consolename	Assigns a terminal for the program you are debugging. (If you omit this switch, the program uses the same terminal as the SWAT debugger.)
/CPU=seconds	Sets the maximum CPU time for the program in seconds.
/DACL	Does not pass the default ACL to the program.
/DATA=pathname	Sets the program's DATAFILE, which identifies the generic @DATA file.
/DEBUG	Starts the program in the AOS or AOS/VS user debugger, granting write access to the program. (This switch is incompatible with the /NOCONSOLE switch.)
/INPUT=pathname	Sets the program's generic @INPUT file.
/LIST=pathname	Sets the program's LISTFILE, which identifies the generic @LIST file.
/MEMORY=pages	Sets the maximum memory size for the program in pages.
/NAME=name	Assigns a simple process name to the program.
/NOCONSOLE	Prevents assignment of generic files for the program. (This switch is incompatible with the /DEBUG switch. To use the SWAT CLI command or console interrupts, you must use this switch.)
/NOCOPY	Prevents automatic copying of the program file on start-up. The SWAT debugger opens the original program file.
/OUTPUT=pathname	Sets the program's generic @OUTPUT file.
/PREEMPTIBLE	Makes the program pre-emptible.
/PRIORITY=n	Sets the priority for the program.
/RESIDENT	Makes the program resident.
/SONS=n	Sets the maximum number of sons that the program can create. The default number is one fewer than the number of sons remaining to the process that called the SWAT debugger. (This default is not the same as that of the CLI PROCESS command.)
/USERNAME=name	Assigns a username to the program.
/WSMAX=pages	Sets the maximum number of pages allowed in main memory at one time. (AOS/VS only — see note below.)
/WSMIN=pages	Sets the minimum number of pages allowed in main memory at one time. (AOS/VS only — see note below.)

The /BREAK, /WSMAX, and /WSMIN switches apply only to the 16- or 32-bit SWAT debugger running under AOS/VS. The AOS SWAT debugger ignores these switches.

If the named or default audit file already exists, the SWAT software appends the debugging dialog to the file. If the file does not exist, the SWAT debugger creates it.

The audit file records only SWAT commands (including comments) and responses. It does not reflect any input or output for the program you are debugging.

Generic Files

To specify a filename for one or more of your program's generic files, you can use the /CONSOLE, /DATA, /INPUT, /LIST, and /OUTPUT switches. If you omit a switch, your program uses the current default filename associated with the generic file. The /NOCONSOLE switch prevents the passing of the generic files to your program.

The User Debugger

When invoking the SWAT software, you can use the /DEBUG switch to begin execution of your program in the AOS or AOS/VS user debugger. This switch grants the user debugger write access to your program. When you proceed from the user debugger, the SWAT session begins.

You can, however, use the DEBUG command to invoke the user debugger within a SWAT session. If you did not use the /DEBUG switch when calling the SWAT debugger, the user debugger will not have write access to the shared portion of your program. When you exit the user debugger, your SWAT session continues where it left off. Refer to the *AOS Debugger and File Editor User's Manual* or the *AOS/VS Debugger User's Manual* for more information.

The SWAT debugger uses certain symbols that the user debugger interprets as delimiters. As a safeguard, you should assign another terminal for your program (using the /CONSOLE, /INPUT, and /OUTPUT switches). Your program and the user debugger then interact with the second terminal, allowing the SWAT software exclusive use of the original terminal.

You cannot assign a terminal to your program if the terminal is currently enabled under the EXEC process. To disable such a terminal, enter the following CLI command from the OP process:

```
CONTROL @EXEC DISABLE @CONn
```

where CONn identifies the terminal.

Running the SWAT Debugger

The SWAT debugger begins executing as soon as you key in the command line. If the debugger cannot start up correctly, you receive an error message. Appendix B lists the errors that can occur when you try to execute the SWAT debugger.

By default, the SWAT software builds a temporary copy of the program file for debugging. This eliminates the danger of altering the original program file. To debug the original file, include the /NOCOPY switch when calling the SWAT debugger.

When the SWAT debugger is ready to accept your first command it displays the message

SWAT REVISION nn.nn ON mm/dd/yy AT hh:mm:ss
PROGRAM -- :pathname

then displays its prompt, which is a right angle bracket followed by a space:

```
>
```

SWAT Commands and Comments

When the SWAT prompt appears, you can enter a SWAT command, a null command, or a comment line. Table 2-2 lists the SWAT commands and gives a brief explanation of their use. Chapter 3 introduces each command as it takes you through a sample debugging session. Chapter 4 is a reference section where you can look up a complete and concise description of each command.

Use a semicolon (;) to separate commands entered on the same line. A complete command line cannot exceed 511 characters.

A SWAT command line can extend over more than one typed line on the terminal. If you simply continue typing, your command line wraps around to the next line. You can also use an ampersand (&) to end an incomplete line, then continue it on the next line. In either case, the SWAT debugger acknowledges that you are continuing the command by preceding the prompt on the next line with an ampersand. For example:

```
> AUDIT ON;BREAKPOINT 23,34,76,81,112;CONTINUE;&↵
&>ENVIRONMENT;CLI "TIME";EXECUTE "DBUG"↵
```

Table 2-2. SWAT Commands

Command	Use
AUDIT	Reports or sets the audit status.
BREAKPOINT	Displays or sets a breakpoint.
BYE	Terminates the SWAT debugger.
CLEAR	Clears a breakpoint.
CLI	Performs a CLI function.
CONTINUE	Initiates or resumes program execution.
DEBUG	Calls the AOS or AOS/VS user debugger.
DESCRIBE	Displays information about a program symbol.
DIRECTORY	Displays or sets the current directory.
ENVIRONMENT	Displays or sets the working environment.
EXECUTE	Executes a file of SWAT commands.
HELP	Displays information about SWAT topics.
LIST	Displays one or more lines from the program source file.
MESSAGE	Displays an error message for an error code.
PREFIX	Displays or sets the SWAT command prompt.
PROMPT	Displays or sets the null command response.
SEARCHLIST	Displays or sets the search list.
SET	Assigns a value to a variable.
TYPE	Displays the value of an expression.
WALKBACK	Displays the current location and calling locations.

You can abbreviate any SWAT command, provided that the abbreviation distinguishes it from all other commands. For example, you can abbreviate the BREAKPOINT command to BR, which distinguishes it from the BYE command. If you enter B only, the SWAT debugger cannot tell if you mean BREAKPOINT or BYE.

If you set up a SWAT audit file, you can place comments in the file to supplement the record of your debugging session. To enter a comment, type a percent sign (%), then the comment. The SWAT debugger ignores everything to the right of the percent sign, but copies the line to the audit file (if auditing is on).

Console Interrupts and Control Commands

If the SWAT debugger has exclusive ownership of the terminal, it can recognize a console interrupt sequence.

To grant the SWAT software exclusive ownership of the console device, use the /NOCONSOLE or /CONSOLE switch (or both) when invoking the debugger.

Whether or not the SWAT debugger owns the console device exclusively, you can use the CTRL-C CTRL-B and CTRL-C CTRL-E control command sequences. The CTRL-C CTRL-B command terminates your program. The CTRL-C CTRL-E sequence produces the same result, but also generates a break file.

Chaining

If the program you are debugging chains to another program, the SWAT debugger continues to run until the final program terminates. If a program chains to one that the SWAT software cannot debug (because you did not compile and link it with the /DEBUG switch), the SWAT debugger remains active, but you cannot debug the new program. If, however, a program chains to one that the SWAT software can debug, the debugger clears any breakpoints remaining in the previous program, displays a new announcement message, then begins a debugging session with the new program.

IMPORTANT: When one program chains to another, the new program's .PR file begins execution. If you debug the new program with the SWAT debugger, you will not be working with a copy of the program. It is as though you used the /NOCOPY switch. A system crash during the debugging session may then leave the .PR file accidentally modified. Also, if two users are working with the same program file, they may interfere with each other.

If you are debugging a series of chained programs, but the first program is not one that the SWAT software can work with (i.e., you did not use the /DEBUG switch when compiling and linking it), you must use the /NOCOPY switch when invoking the SWAT debugger.

SWAT Error Handling

During the course of a debugging session the SWAT software may encounter errors. There are three levels of errors that can occur:

Warning

Soft

Fatal

A warning message indicates that a problem may exist as a result of the current operation.

When a soft error occurs, the SWAT debugger displays an error message explaining the problem. In most cases, the command line contains an error in syntax, an invalid argument, or is otherwise inappropriate. After displaying the error message, the SWAT debugger prompts you for another command.

When a fatal error occurs, both your program and the SWAT software terminate. Depending on the error, you receive an error message from either the SWAT debugger or the operating system.

Each command description in Chapter 4 lists the errors most likely to occur when using the command. Refer also to Appendix B, which lists all the SWAT error messages and gives a complete explanation of each.

End of Chapter

Chapter 3
A Sample Debugging Session

In this chapter we take you through a sample SWAT debugging session. You will see how each SWAT command works, what options are available to you, and how to use the commands to isolate errors in your source program. If you haven't already read Chapters 1 and 2, please do. The background they provide will help you with the material in this chapter.

This sample session uses a PL/I program running on an AOS system. Appendix C presents a compilation listing and audit file of the same program running under the 32-bit AOS/VS SWAT debugger. Appendix C also illustrates the use of the SWAT debugger with programs written in FORTRAN 77, COBOL, PASCAL, and C.

The Program

Before we begin the debugging session, let's look at the program we'll be working with. The program, called EXCHANGE, is written in PL/I. It was designed for the accounting department of an American-based company having branch offices in eight other countries. The program calculates monetary exchange values based on daily exchange rates for the currencies the company deals with.

The program has two separate modules: the main program (EXCHANGE) and an external subroutine called CONVERSION. The main program module also contains an embedded subroutine labeled CHECK.

What It Does

As we explain how the program works, refer to the program listings (Figures 3-1 and 3-2) and the program flow chart (Figure 3-3). When it begins to run, the EXCHANGE program reads the date and exchange rate information stored in a disk file called RATE_FILE. The file might look like this:

```
811124,
37.52,2.2465,2.4543,1194,1.7725,1.1865,3.4190,225.90,
.02663,.4458,.4081,.000835,.5566,.8434,.2924,.004471,
```

The first data item is the date of the exchange rates in the format YYMMDD, which corresponds to PL/I's built-in DATE function. (The other entries are the 16 exchange rates.) Later we describe the RATES table and explain what each element represents. If the date in this file does not match the system date (indicating that today's rates have not been entered), the program displays the message

RATES NOT CURRENT

and terminates. If the rate file does not exist, or if it lacks the required information, the program terminates after displaying the message

RATES NOT AVAILABLE

If the dates match, the program enters a loop so that it will continually be available to perform exchange calculations. First it displays a menu of the currencies handled by the company.

Select currency code for US$ exchange:

1 Belgian francs
2 W. German marks
3 Dutch guilders
4 Italian lire
5 Swiss francs
6 Canadian dollars
7 Saudi riyals
8 Japanese yen

(Type 0 to end the program)
Enter the currency code:

The operator types the code number identifying the currency involved in the exchange. The valid currency code numbers are 1 through 8; code 0 signals the program to terminate. The subroutine CHECK verifies that the operator has given a valid code. The program stops if it encounters an invalid code.

After determining the type of currency, the program asks the operator

Do you want to convert US$ into Foreign currency?
Enter 'Y' or 'N':

If the operator responds with anything other than "*Y*" or "*y*", the program assumes the exchange will be from the foreign currency into American dollars. Based on the operator's response, the program sets an index to locate the appropriate exchange rate in the table RATES.

The program reads the RATES table directly from the disk file. It is organized as follows:

RATE	Represents	For exchanging
(1)	Belgian francs per $1 US	US$ to BF
(2)	West German marks per $1 US	US$ to DM
(3)	Dutch guilders per $1 US	US$ to gld
(4)	Italian lire per $1 US	US$ to Lit
(5)	Swiss francs per $1 US	US$ to SF
(6)	Canadian dollars per $1 US	US$ to Can$
(7)	Saudi Arabian riyals per $1 US	US$ to SRI
(8)	Japanese yen per $1 US	US$ to Y
(9)	US dollars per 1 Belgian franc	BF to US$
(10)	US dollars per 1 West German mark	DM to US$
(11)	US dollars per 1 Dutch guilder	gld to US$
(12)	US dollars per 1 Italian lira	Lit to US$
(13)	US dollars per 1 Swiss franc	SF to US$
(14)	US dollars per 1 Canadian dollar	Can$ to US$
(15)	US dollars per 1 Saudi Arabian riyal	SRI to US
(16)	US dollars per 1 Japanese yen	Y to US$

(The program requires a specific exchange rate for converting American dollars to foreign currency and for converting foreign currency to American dollars. These exchange rates are not necessarily reciprocals.)

If the operator indicates an exchange of American dollars into foreign currency, the program sets the table index according to the currency code entered. (The first eight entries correspond directly to the currency codes in the menu for this type of exchange.) To convert foreign currency into American dollars, the program adds eight to the currency code, thereby using the second half of the table. To exchange Dutch guilders, for example, the operator enters a currency code of 3. The program uses RATES(3) to convert American dollars into Dutch guilders, and RATES(3+8) to convert guilders to American dollars.

The program calls the subroutine CONVERSION, which requests the incoming currency amount in American dollars:

ENTER US$:

or in foreign currency

ENTER gld:

(for example). The routine performs the calculation using the appropriate exchange rate, then displays the result in the format

X equivalent to: Y

where *X* represents the incoming currency, and *Y* represents the resulting currency. After the CONVERSION routine displays the result, the program returns to the top of the loop, presenting the currency menu and awaiting a currency code for the next exchange operation. To terminate the program, the operator enters 0 instead of a currency code (digits 1 through 8).

```
SOURCE FILE: EXCHANGE
COMPILED ON 11/24/81   AT 12:14:42   BY PL/I REV  2.31
OPTIONS: PL1/DEBUG/L=EXLIST.EXCHANGE


 1   EXCHANGE:
 2           PROCEDURE;
 3
 4           /* This program calculates currency exchange values based on
 5   *       daily international exchange rates.  The operator selects
 6   *       the type of exchange then enters the incoming amount.  The
 7   *       program performs the calculation and displays the result. */
 8
 9           DECLARE (SCREEN, KEYBOARD, RATE_FILE) FILE;
10           DECLARE CONVERSION ENTRY(FIXED BINARY,FIXED DECIMAL(10,6));
11           DECLARE SELECTION FIXED BINARY;
12           DECLARE 1 DAILY_RATES,
13                   2 RATE_DATE CHARACTER(6),
14                   2 RATES(16) FIXED DECIMAL(10,6);
15           DECLARE RESPONSE CHARACTER(1);
16           DECLARE INDX FIXED BINARY;
17
18           OPEN FILE(SCREEN) STREAM OUTPUT PRINT TITLE('@OUTPUT');
19           OPEN FILE(KEYBOARD) STREAM INPUT TITLE('@INPUT');
20           OPEN FILE(RATE_FILE) STREAM INPUT;
21
```

Figure 3-1. Compilation Listing of the Module EXCHANGE (continues)

```
22   /* Read in today's exchange rates.  Compare date with system date. */
23
24           GET FILE(RATE_FILE) LIST(DAILY_RATES);
25
26           ON ENDFILE(RATE_FILE)
27           BEGIN;
28                   PUT FILE(SCREEN) SKIP LIST("NO RATES AVAILABLE.");
29                   STOP;
30           END; /*BEGIN Block for End of File condition */
31
32   /* Compare file's date with the system date. */
33
34           IF RATE_DATE ^= DATE() THEN DO;
35                   PUT FILE(SCREEN) SKIP LIST("RATES NOT CURRENT");
36                   STOP;
37                   END; /*DO Block */
38
39   /* Set up ON ERROR condition for bad input */
40
41           ON ERROR
42                   BEGIN;
43                   PUT FILE(SCREEN) SKIP LIST("Invalid input.  "
44                                                   !!"Try again.");
45                   GO TO LOOP;
46                   END;
47
48   /* Display menu for exchanges */
49
50           LOOP:   DO WHILE("1"B);
51
52                   PUT FILE(SCREEN) SKIP(3) LIST("Select currency code for "!!
53                           "US$ exchange:");
54                   PUT FILE(SCREEN) SKIP(2) EDIT
55                           ("1","Belgian francs","2","W. German marks","3",
56                           "Dutch guilders","4","Italian lire","5","Swiss francs",
57                           "6","Canadian dollars","7","Saudi riyals","8",
58                           "Japanese yen","(Type 0 to end the program)",
59                           "Enter the currency code:  ")
60                           (8(X(5),A(1),X(3),A(16),SKIP),SKIP,A(27),SKIP,A(26));
61
62                   GET FILE(KEYBOARD) LIST(SELECTION);
63
64                   CALL CHECK(SELECTION);
65
66   /* Request type of exchange */
67
68           PUT FILE(SCREEN) SKIP LIST("Do you want to convert US$ into"
69                   !!" Foreign currency?");
70           PUT FILE(SCREEN) SKIP LIST("Enter 'Y' or 'N':  ");
71           GET FILE(KEYBOARD) LIST(RESPONSE);
```

Figure 3-1. Compilation Listing of the Module EXCHANGE (continued)

```
72
73          IF RESPONSE = "Y" THEN INDX = SELECTION;
74          ELSE IF RESPONSE = "y" THEN INDX = SELECTION;
75                  ELSE INDX = SELECTION + 8;
76
77          CALL CONVERSION(INDX, RATES(INDX));
78
79  END;/* LOOP */
80
81  CHECK:
82          PROCEDURE(SEL);
83          DECLARE SEL FIXED BINARY;
84
85          IF SEL > 0 THEN
86                  IF SEL < 9 THEN RETURN;
87          PUT FILE(SCREEN) SKIP LIST("Exchange program ended.");
88          STOP;
89  END;/* CHECK SUBROUTINE */
90
91  END;/* EXCHANGE */
```

Figure 3-1. Compilation Listing of the Module EXCHANGE (continued)

```
EXTERNAL ENTRY POINTS

NAME          CLASS       SIZE   LOC    ATTRIBUTES

EXCHANGE      CONSTANT                  ENTRY EXTERNAL


PROCEDURE EXCHANGE ON LINE 2

NAME          CLASS       SIZE   LOC    ATTRIBUTES

(ONUNIT)      CONSTANT                  ENTRY
(ONUNIT)      CONSTANT                  ENTRY
SCREEN        CONSTANT                  FILE EXTERNAL
KEYBOARD      CONSTANT                  FILE EXTERNAL
RATE_FILE     CONSTANT                  FILE EXTERNAL
CONVERSION    CONSTANT                  ENTRY EXTERNAL
SELECTION     AUTOMATIC   1W     8      FIXED BINARY(15)
DAILY_RATES   AUTOMATIC   102C   10     STRUCTURE
RATE_DATE     MEMBER      6C     0C     CHAR(6)
RATES         MEMBER      96C    6C     ARRAY FIXED DECIMAL(10.6)
RESPONSE      AUTOMATIC   1C     61     CHAR(1)
INDX          AUTOMATIC   1W     9      FIXED BINARY(15)
LOOP          CONSTANT                  LABEL
CHECK         CONSTANT                  ENTRY


BEGIN BLOCK (ONUNIT) ON LINE 27

NAME          CLASS       SIZE   LOC    ATTRIBUTES


BEGIN BLOCK (ONUNIT) ON LINE 42

NAME          CLASS       SIZE   LOC    ATTRIBUTES


PROCEDURE CHECK ON LINE 82

NAME          CLASS       SIZE   LOC    ATTRIBUTES

SEL           PARAMETER   1W     V      FIXED BINARY(15)
```

Figure 3-1. Compilation Listing of the Module EXCHANGE (continued)

```
FILE:  EXCHANGE                          11/24/81        12:14:42

COMPILE-TIME STATISTICS

PHASE          TIME             SYMBOL TABLE I/O
SETUP           0:00:05
PASS1           0:00:06         0
DECLARE         0:00:00         0
PASS2           0:00:09         0
ALLOCATOR       0:00:05         0
CHAIN           0:00:08         8
PASS3           0:00:26         14
FINAL           0:00:00         0
TOTAL           0:00:59         22

LINES COMPILED 91
LINES PER MIN 90
SYMBOL TABLE PAGES USED 8. MAX 256
CONSTANTS AND INT STATIC 73
PROCEDURE CODE 1019
```

Figure 3-1. Compilation Listing of the Module EXCHANGE (concluded)

```
SOURCE FILE: CONVERSION
COMPILED ON 11/24/81   AT 12:16:34   BY PL/I REV  2.31
OPTIONS: PL1/DEBUG/L=CONLIST.CONVERSION


  1
  2   CONVERSION:
  3           PROCEDURE(CODE, RATE);
  4
  5           DECLARE CODE FIXED BINARY;
  6           DECLARE RATE FIXED DECIMAL(10,6);
  7           DECLARE CURRENCY(8) CHARACTER(4) VARYING STATIC INTERNAL
  8                   INIT("BF","DM","gld","Lit","SF","Can$","SRI","Y");
  9           DECLARE FVAL FIXED DECIMAL(11,2);
 10           DECLARE DVAL PICTURE "$$$$$$$$9.99";
 11           DECLARE (SCREEN, KEYBOARD) FILE;
 12
 13           IF CODE < 8 THEN DO;
 14                           PUT FILE(SCREEN) SKIP LIST("ENTER US$: ");
 15                           GET FILE(KEYBOARD) LIST(DVAL);
 16                           FVAL = DVAL * RATE;
 17                           PUT FILE(SCREEN) EDIT(DVAL,
 18           " US equivalent to:  ",FVAL, CURRENCY(CODE))
 19           (SKIP(2),X(5),F(11,2),A(20),F(11,2),X(1),A(4),SKIP(2));
 20                           RETURN;
 21                           END; /* DO GROUP FOR DOLLAR CONVERSION */
 22
 23           ELSE DO;
 24           PUT FILE(SCREEN) SKIP LIST("ENTER ",CURRENCY(CODE-8),":  ");
 25           GET FILE(KEYBOARD) LIST(FVAL);
 26           DVAL = FVAL * RATE;
 27           PUT FILE(SCREEN) EDIT(FVAL, CURRENCY(CODE-8),
 28           " equivalent to:  ",DVAL," US")
 29           (SKIP(2), X(5),F(11,2),X(1),A(4),A(17),F(11,2),A(3),SKIP(2));
 30           RETURN;
 31           END; /* DO GROUP FOR FOREIGN CURRENCY CONVERSION */
 32
 33   END; /* CONVERSION */
```

Figure 3-2. Compilation Listing of the Module CONVERSION (continues)

```
EXTERNAL ENTRY POINTS

NAME          CLASS       SIZE   LOC    ATTRIBUTES

CONVERSION    CONSTANT                  ENTRY EXTERNAL


PROCEDURE CONVERSION ON LINE 3

NAME          CLASS       SIZE   LOC    ATTRIBUTES

CODE          PARAMETER   1W     V      FIXED BINARY(15)
RATE          PARAMETER   6C     V      FIXED DECIMAL(10.6)
CURRENCY      STATIC      24W    8      ARRAY VARYING CHAR(4) INITIAL
FVAL          AUTOMATIC   6C     8      FIXED DECIMAL(11.2)
DVAL          AUTOMATIC   12C    11     PICTURE '$$$$$$$$9.99'
SCREEN        CONSTANT                  FILE EXTERNAL
KEYBOARD      CONSTANT                  FILE EXTERNAL


FILE:  CONVERSION                       11/24/81       12:16:34

COMPILE-TIME STATISTICS

PHASE          TIME             SYMBOL TABLE I/O
SETUP           0:00:05
PASS1           0:00:03         0
DECLARE         0:00:00         0
PASS2           0:00:10         0
ALLOCATOR       0:00:02         0
CHAIN           0:00:10         4
PASS3           0:00:09         4
FINAL           0:00:00         0
TOTAL           0:00:39         8

LINES COMPILED 33
LINES PER MIN 48
SYMBOL TABLE PAGES USED 4, MAX 256
CONSTANTS AND INT STATIC 100
PROCEDURE CODE 380
```

Figure 3-2. Compilation Listing of the Module CONVERSION (concluded)

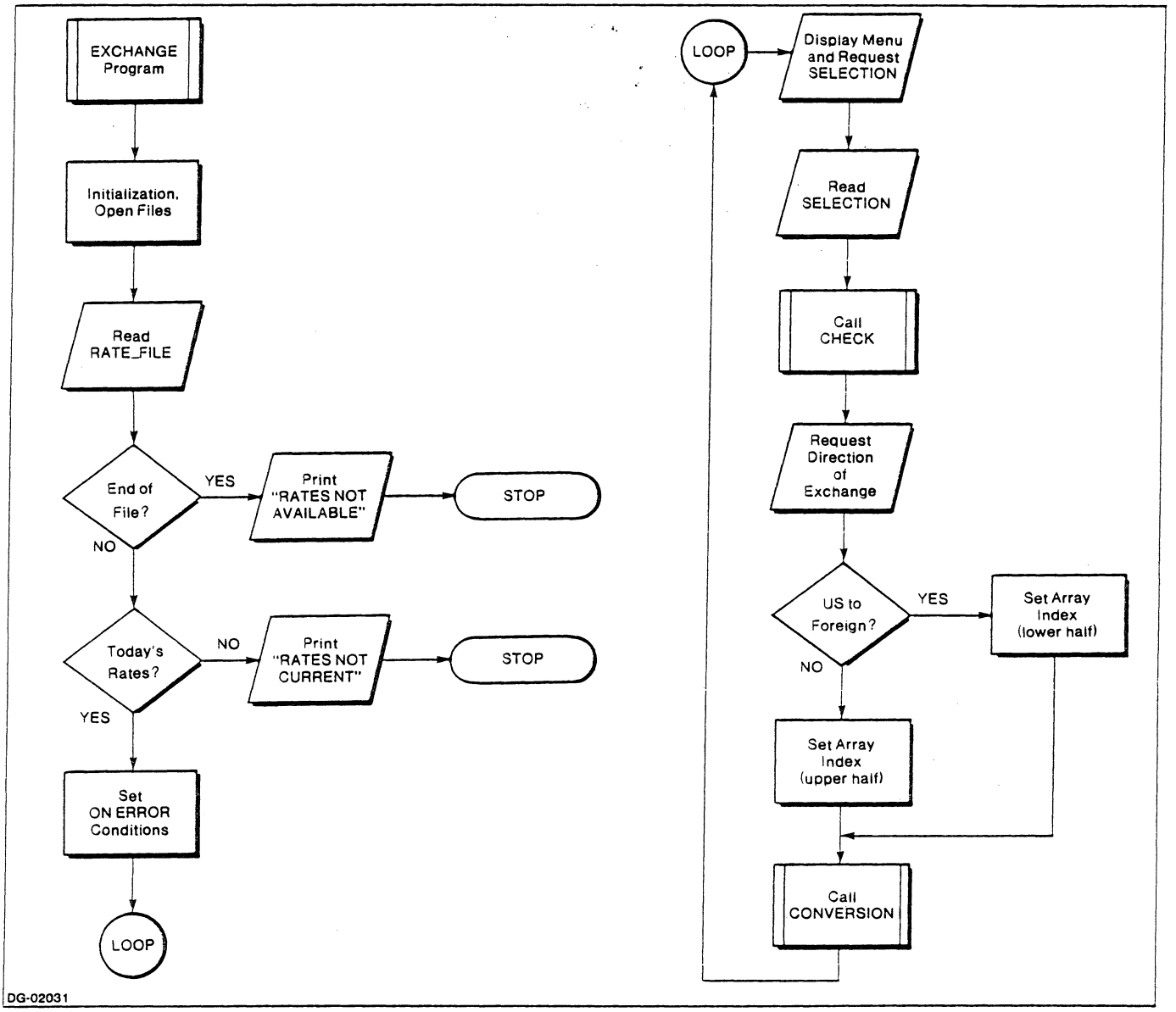


Figure 3-3. Flow Chart of the EXCHANGE Program (continues)

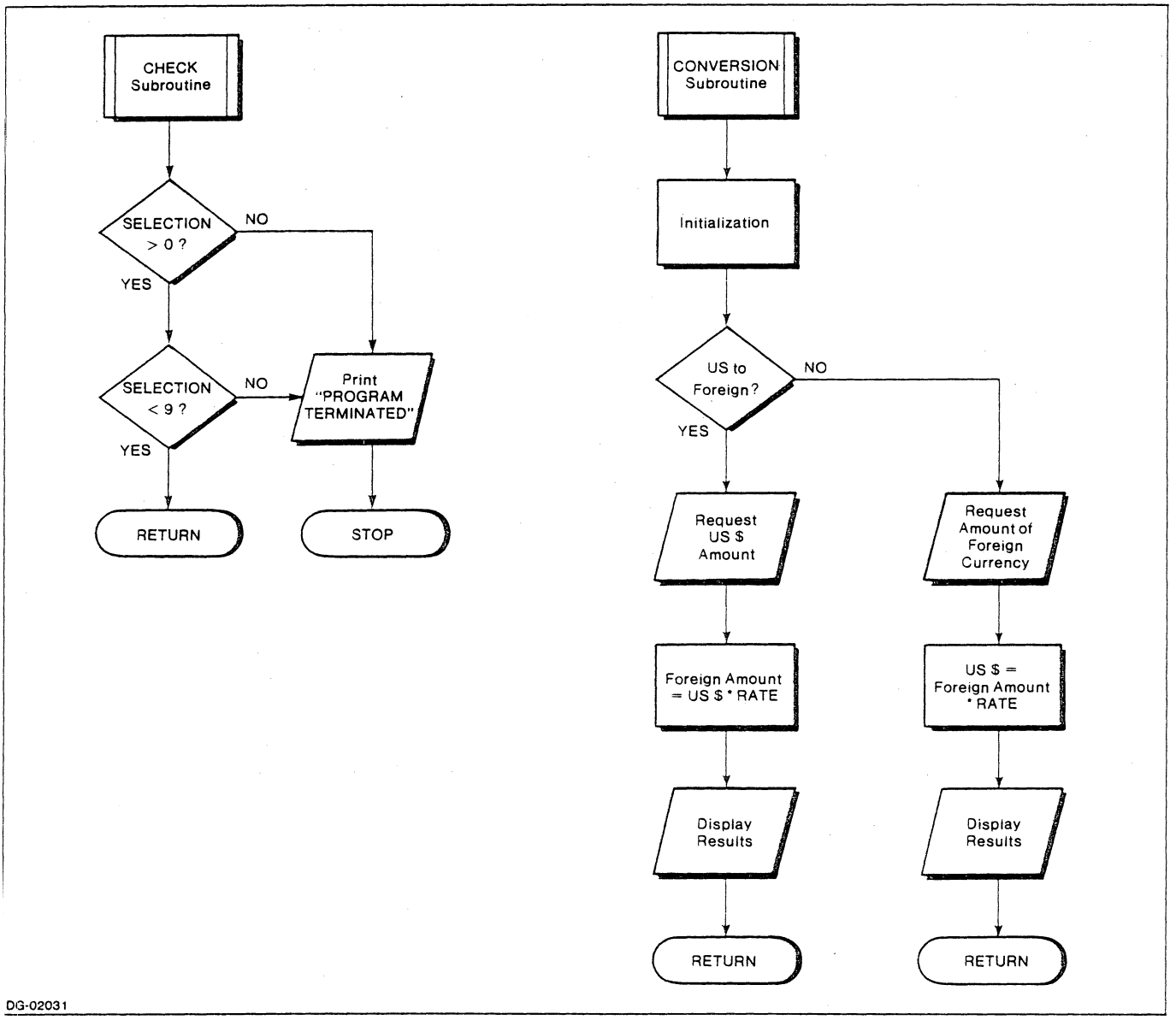


Figure 3-3. Flow Chart of the EXCHANGE Program (concluded)

Starting the Debugging Session

Now that you are familiar with what the program does, we can begin a debugging session. Assuming the source program has been compiled and linked with the /DEBUG switch, and that the SWAT debugger has access to all the necessary files, we can invoke the SWAT software.

It will be useful to keep a record of the debugging session, so we'll set up an audit file. (A copy of the audit file for this debugging session appears in Figure 3-4 at the end of this chapter.) From the CLI, we type the command

```
XEQ SWAT/AUDIT EXCHANGE ↵
```

This command invokes the SWAT software, requests it to open an audit file with the default name EXCHANGE.AU, and identifies the program we are about to debug. The SWAT debugger begins running, displays its announcement message, then gives you a prompt.

```
SWAT REVISION 02.00 ON 11/24/81 AT 10:09:13
PROGRAM – :UDD:TOM:EXCHANGE
>
```

Now the SWAT debugger is ready to accept a command. Before we get into the actual debugging of the program, we'll introduce you to each SWAT command. The following sections use the EXCHANGE program to illustrate how each command works and explain the available options. After presenting all the SWAT commands, we'll begin a new session devoted to debugging the EXCHANGE program.

The Prompt

When the debugger is ready to accept a command, it displays a prompt followed by a space. When you begin a debugging session the prompt is a right angle bracket (as just shown).

You can change the SWAT prompt by typing PREFIX followed by a string of up to 32 characters. Enclose the string in quotation marks or single apostrophes. To set the SWAT prompt to "NEXT COMMAND?", we enter

```
> PREFIX "NEXT COMMAND?" ↵
```

or

```
> PREFIX 'NEXT COMMAND?' ↵
```

The debugger then responds with its new prompt.

```
NEXT COMMAND?
```

The new prompt remains in effect until you change it. To return the prompt to its original form, we provide a new string.

```
NEXT COMMAND? PREFIX ">" ↵
>
```

Auditing

Since we assigned an audit file when we called the debugger, auditing is automatically enabled when the debugging session begins. The AUDIT command displays whether auditing is currently ON or OFF.

```
> AUDIT ↵
ON
```

When auditing is on, the debugger copies all commands and responses to the audit file. To disable auditing and close the audit file, we type

```
> AUDIT OFF ↵
OFF
```

This command closes the current audit file, which is EXCHANGE.AU. Until we turn auditing on again, our debugging dialog will not appear in the audit file.

To re-enable the default audit file, we enter

```
> AUDIT ON ↵
ON
```

To open a different audit file, enter its filename following the ON keyword. Enclose the filename in quotation marks or apostrophes. Only one audit file can be open at a time. Therefore, when you open an audit file, the SWAT debugger automatically closes any currently open audit file.

Making Comments

Instead of giving the SWAT debugger a command, you can write a comment. Comments are useful when you are building an audit file to record the debugging session. You can annotate an audit file to explain your debugging strategies, or to remind yourself about changes you want to make to the source code. To make a comment, enter a percent sign (%) in the first space, then follow it with the comment text. For example:

```
> %This is a comment line. ↵
```

The debugger copies a comment to the audit file (if open), but ignores it otherwise.

The User Debugger

The SWAT debugger provides an easy interface with the AOS or AOS/VS user debugger. If you use the /DEBUG switch, the debugging session begins in the user debugger. When you exit the user debugger, the SWAT session begins. During a SWAT session, you can call the user debugger by typing DEBUG. If you used the /DEBUG switch, the user debugger will have write access privileges. If you call the user debugger, but did not use the /DEBUG switch, write access is denied. We did not use the /DEBUG switch, so if we type

```
> DEBUG ↵
AOS USER DEBUGGER, REV 3.40
WRITE ACCESS DENIED
+
```

the user debugger begins execution, and we receive a prompt. To exit the user debugger we type P ↵ (or, for the AOS/VS debugger, ESC R).

```
+P ↵

>
```

Our SWAT session returns, signaled by the SWAT prompt.

Your interactions with the user debugger do not appear in the audit file.

The Working Environment

The ENVIRONMENT command displays the name of the block the SWAT debugger is currently operating in. This is your *working environment*. If we type

```
> ENVIRONMENT ↵
```

the SWAT debugger responds

```
:EXCHANGE
```

Our working environment is the main program module called EXCHANGE. To set the working environment, simply enter an environment specifier with the command. (Chapter 1 explains how to build an environment specifier.)

To move to the environment of the external procedure CONVERSION, we type

```
> ENVIRONMENT :CONVERSION ↵
:CONVERSION
```

The working environment is now that of the module called CONVERSION.

We can use the @MAIN keyword to return to the main program module.

```
> ENVIRONMENT @MAIN ↵
:EXCHANGE
```

Now we're back where we started.

Breakpoints

The most useful tool for debugging a program is the breakpoint. When you attach a breakpoint to a program statement or clause, the program halts or *traps* when execution encounters the breakpoint. The SWAT debugger takes over allowing you to examine the state of affairs at this instant in execution, that is, before the breakpoint statement executes. The SWAT debugger gives you a maximum of 20 breakpoints to use. You can clear a breakpoint from a statement and reset it elsewhere. We explain how to do this later in the chapter.

The key to successful debugging is the strategic placement of breakpoints.

To set a breakpoint, use the BREAKPOINT command with a locator that identifies a statement or clause. A simple locator can be a line number within the current program module, or a label that is defined within the working environment. If the locator does not satisfy these conditions, you must fully qualify it by indicating the block where the line number or label exists.

You can't (and wouldn't want to) set a breakpoint at a nonexecutable statement.

Let's begin by setting a breakpoint at line 24. This will trap execution when the program is about to read the daily exchange rate information from RATE_FILE.

```
> BREAKPOINT 24 ↓
Set at :EXCHANGE:24
```

We can set multiple breakpoints with one command:

```
> BREAKPOINT 34 62 64 87 ↓
Set at :EXCHANGE:34
Set at :EXCHANGE:62
Set at :EXCHANGE:64
Set at :EXCHANGE:CHECK:87
```

We can set breakpoints anywhere within the current program module by using a simple line number. We can use simple statement labels as locators only if the label is defined within the working environment.

```
> BREAKPOINT LOOP ↓
Set at :EXCHANGE:50
```

To set a breakpoint in a block outside the current program module, we must fully qualify the locator, or change the working environment. Observe the next series of commands:

```
> BREAKPOINT :CONVERSION:13 ↓
Set at :CONVERSION:13

> ENVIRONMENT :CONVERSION ↓
:CONVERSION

> BREAKPOINT 16 26 ↓
Set at :CONVERSION:16
Set at :CONVERSION:26

> BREAKPOINT ↓
Set at :EXCHANGE:24
Set at :EXCHANGE:34
Set at :EXCHANGE:50
Set at :EXCHANGE:62
Set at :EXCHANGE:64
Set at :EXCHANGE:CHECK:87
Set at :CONVERSION:13
Set at :CONVERSION:16
Set at :CONVERSION:26
```

If you do not supply a locator, the BREAKPOINT command displays all current breakpoints.

The EXCHANGE program contains two brief BEGIN blocks: one for an end-of-file and one for an error. The BEGIN block is an embedded procedure block just as is the subroutine CHECK. The SWAT debugger uses the line number of the BEGIN block to identify this unnamed procedure block.

```
> ENVIRONMENT @MAIN ↵
:EXCHANGE

> BREAKPOINT 28 ↵
Set at :EXCHANGE:27:28
```

In the same way, you can use the line number of an unnamed block to identify the block:

```
> ENVIRONMENT 42 ↵
:EXCHANGE:42

> BREAKPOINT 43 ↵
Set at :EXCHANGE:42:43
```

Proceed Count

You can set a proceed count for a breakpoint. This means that you can instruct the debugger to ignore the breakpoint a certain number of times before causing a trap at the breakpoint. This feature is useful for debugging within a loop. You can allow execution to proceed through the loop a certain number of times before trapping.

To set a proceed count of five for the statement on line 71, we type

```
> BREAKPOINT 71 COUNT=5 ↵
Set at :EXCHANGE:71 count=5
```

You can change the proceed count at any time by using the BREAKPOINT command with a new proceed count.

```
> BREAKPOINT 71 COUNT=4 ↵
Reset at :EXCHANGE:71 count=4
```

The proceed count must be a positive integer value not greater than 32767. You can use an expression as the proceed count, provided that it resolves to an acceptable value. The default proceed count is one.

Action String

Another feature of the BREAKPOINT statement is that you can assign an action string. The action string specifies one or more SWAT commands to be executed when a trap occurs at that breakpoint. To attach an action string, use the ACTION keyword with the BREAKPOINT command. Follow the keyword with an equal sign and a string of SWAT commands. Enclose the string in quotation marks or apostrophes. The commands within the string must appear with any necessary arguments. You cannot pass arguments to action string commands interactively.

If, when execution traps at statement 71, we want to list that statement, we would enter the command

```
> BREAKPOINT 71 ACTION = "LIST 71" ↵
Reset at :EXCHANGE:71 count=4 action="LIST 71"
```

To clear an action string from a breakpoint, assign a null string ("").

Clearing Breakpoints

To remove a breakpoint from a statement, use the CLEAR command. As with BREAKPOINT, you supply a locator that points to a statement or clause where you've set a breakpoint. Remember that you must fully qualify a line number outside the current program module, or a label locator that lies outside the working environment.

To clear the breakpoint we just set, type

```
> CLEAR 71 ↵
Cleared at :EXCHANGE:71 count=4 action="LIST 71"
```

You can clear more than one breakpoint at a time by supplying more than one locator.

```
> CLEAR 28 43 62 87 ↵
Cleared at :EXCHANGE:27:28
Cleared at :EXCHANGE:42:43
Cleared at :EXCHANGE:62
Cleared at :EXCHANGE:CHECK:87

> BREAKPOINT ↵
Set at :EXCHANGE:24
Set at :EXCHANGE:34
Set at :EXCHANGE:50
Set at :EXCHANGE:64
Set at :CONVERSION:13
Set at :CONVERSION:16
Set at :CONVERSION:26
```

To clear a breakpoint that is in another program module, supply a locator that specifies the module. For example:

```
> CLEAR :CONVERSION:13 ↵
Cleared at :CONVERSION:13
```

If you want to clear all the current breakpoints in your program, type CLEAR @ALL. The SWAT keyword @ALL in this case represents every breakpoint throughout the program.

```
> CLEAR @ALL ↵
Cleared at :EXCHANGE:24
Cleared at :EXCHANGE:34
Cleared at :EXCHANGE:64
Cleared at :EXCHANGE:50
Cleared at :CONVERSION:16
Cleared at :CONVERSION:26

> BREAKPOINT ↵
>
```

Listing Lines from the Source Code

The LIST command lets you display one or more lines from the program source file. Thus, you can visualize where you are in the program, you can see portions of code so that you can select locators, and you can work without fumbling with a line printer listing. For example, when the SWAT debugger halts at a breakpoint, you can display the breakpoint statement and the code surrounding it.

As with BREAKPOINT and CLEAR, you supply a locator with the command. If you give a single locator, the debugger displays a single line.

```
> LIST 26 ⤶
26 ON ENDFILE(RATE_FILE)
```

To display a range of lines, give two locators: the first locator indicates where to begin the display, the second, where to end it. The largest range the SWAT debugger accepts is 1 through 8191 (AOS) or 1 through 32767 (AOS/VS). If you specify a range larger than the available code, the debugger ignores your error and displays all the code within the specified range.

```
> LIST 26 30 ⤶
26      ON ENDFILE(RATE_FILE)
27      BEGIN;
28          PUT FILE(SCREEN) SKIP LIST("NO RATES AVAILABLE.");
29          STOP;
30      END; /* BEGIN Block for End of File condition */
```

The rules for using locators are as we've described for the BREAKPOINT and CLEAR commands: you can use line numbers or statement labels. To identify a line number outside the current module, or a label outside the working environment, qualify the locator with an environment specifier. If you specify a range, both locators must identify lines in the same external procedure.

When listing lines from the source code, the SWAT debugger identifies the lines where you set a breakpoint by placing the letter B after the line number.

```
> BREAKPOINT 24 34 64 ⤶
Set at :EXCHANGE:24
Set at :EXCHANGE:34
Set at :EXCHANGE:64

> LIST 20 35 ⤶
20          OPEN FILE(RATE_FILE) STREAM INPUT;
21
22      /* Read in today's exchange rates. Compare date with system date. */
23
24B         GET FILE(RATE_FILE) LIST(DAILY_RATES);
25
26          ON ENDFILE(RATE_FILE)
27          BEGIN;
28              PUT FILE(SCREEN) SKIP LIST("NO RATES AVAILABLE.");
29              STOP;
30          END; /*BEGIN Block for End of File condition */
31
32      /* Compare file's date with the system date. */
33
34B         IF RATE_DATE ^= DATE() THEN DO;
35              PUT FILE(SCREEN) SKIP LIST("RATES NOT CURRENT");
```

To list the entire contents of the current module, use the keyword @ALL instead of the two locators. (You can use the CTRL-S and CTRL-Q key pairs to pause and restart screen display of a long listing.)

The LIST command assumes that the source file is available. If there is no source file for the program, or if you altered or renamed it after compilation, the LIST command does not work. You then receive the system message:

FILE DOES NOT EXIST

A missing source file does not affect the other SWAT commands.

Running the Program

Now that we've explained how breakpoints work, the next step toward finding program bugs is to execute portions of the program. To begin program execution, use the CONTINUE command. Usually the CONTINUE command picks up execution from a breakpoint trap and "continues" execution. Because our program hasn't started yet, the CONTINUE command initiates execution at the first statement of the main program module. Execution proceeds until either the program terminates (for whatever reason) or execution encounters a breakpoint.

Many programs do not require operator intervention (as ours does) to terminate. If you issue a CONTINUE command when debugging such a program, and have not set a breakpoint somewhere in the path of execution, your program runs to completion (or termination) without trapping. You'll find both your program and the SWAT session terminated.

Let's see what happens when we type

```
> CONTINUE ↵
```

Breakpoint trap at :EXCHANGE:24

The program begins execution and halts at the first breakpoint it encounters. This in itself can give you information. You know where execution started and you know where it stopped. If there are decisions made along the way, you may be able to trace the path of execution. The strategic placement of breakpoints can assist you. If control can branch to a variety of statements, you can set a breakpoint at each possibility. When you issue the CONTINUE command, you can observe where the path of execution leads.

The Breakpoint Environment

Now that we've issued a CONTINUE command, the SWAT debugger takes on an entirely new dimension. The program has actually begun to run. When execution encounters a breakpoint, the program halts just before executing the breakpoint statement. Then the SWAT software takes over.

The environment where a breakpoint trap occurs is called the *breakpoint environment*. When execution traps, the working environment becomes the breakpoint environment. *This is not necessarily the environment you were in when you issued the CONTINUE command.* The breakpoint trap message reports your current location. In our case, the breakpoint message was

Breakpoint trap at :EXCHANGE:24

If we display the environment we see

```
> ENVIRONMENT ↵
:EXCHANGE
```

Our current location is line 24 in the :EXCHANGE module. Now that program execution has begun, the debugger automatically reports the current breakpoint trap position when displaying the current breakpoints.

```
> BREAKPOINT ↵
Set at :EXCHANGE:24
Set at :EXCHANGE:34
Set at :EXCHANGE:64
Current location is :EXCHANGE:24
```

If you change the working environment from the breakpoint environment, you can easily return to the breakpoint environment by using the @BREAK keyword with the ENVIRONMENT command.

```
> ENVIRONMENT ↵
:EXCHANGE

> ENVIRONMENT :CONVERSION ↵
:CONVERSION

> BREAKPOINT 16 26 ↵
Set at :CONVERSION:16
Set at :CONVERSION:26

> ENVIRONMENT @BREAK ↵
:EXCHANGE
```

Using the LIST command, we can display the statement where execution trapped. SWAT identifies the breakpoint trap statement by placing the letter C after its line number.

```
> LIST 20 28 ↵
20          OPEN FILE(RATE_FILE) STREAM INPUT;
21
22      /* Read in today's exchange rates. Compare date with system date. */
23
24C           GET FILE(RATE_FILE) LIST(DAILY_RATES);
25
26          ON ENDFILE(RATE_FILE)
27          BEGIN;
28              PUT FILE(SCREEN) SKIP LIST("NO RATES AVAILABLE.");
```

To view code surrounding the current breakpoint trap statement, use the @BREAK keyword with the LIST command. The debugger displays the current breakpoint statement with up to 10 lines of source code to either side. You can use this keyword in any environment.

```
> LIST @BREAK ↵
14              2 RATES(16) FIXED DECIMAL(10,6);
15          DECLARE RESPONSE CHARACTER(1);
16          DECLARE INDX FIXED BINARY;
17
18          OPEN FILE(SCREEN) STREAM OUTPUT PRINT TITLE("@OUTPUT");
19          OPEN FILE(KEYBOARD) STREAM INPUT TITLE("@INPUT");
20          OPEN FILE(RATE_FILE) STREAM INPUT;
21
22      /* Read in today's exchange rates. Compare date with system date. */
23
24C           GET FILE(RATE_FILE) LIST(DAILY_RATES);
25
26          ON ENDFILE(RATE_FILE)
27          BEGIN;
28              PUT FILE(SCREEN) SKIP LIST("NO RATES AVAILABLE.");
29              STOP;
30          END; /*BEGIN Block for End of File condition */
31
32      /* Compare file's date with the system date. */
33
34B           IF RATE_DATE ^= DATE() THEN DO;
```

By default, the SWAT debugger performs the LIST @BREAK command when you enter a null command (NEW LINE only). You can simply press the NEW LINE key after a trap to display the breakpoint trap statement. (Later in this chapter we describe the PROMPT command, which you can use to change the null command response.)

```
> CONTINUE ↵
```

```
Breakpoint trap at :EXCHANGE:34

> l
24B         GET FILE(RATE_FILE) LIST(DAILY_RATES);
25
26          ON ENDFILE(RATE_FILE)
27          BEGIN;
28              PUT FILE(SCREEN) SKIP LIST("NO RATES AVAILABLE.");
29              STOP;
30          END; /*BEGIN Block for End of File condition */
31
32      /* Compare file's date with the system date. */
33
34C           IF RATE_DATE ^= DATE() THEN DO;
35              PUT FILE(SCREEN) SKIP LIST("RATES NOT CURRENT");
36              STOP;
37              END; /*DO Block */
38
39      /* Set up ON ERROR condition for bad input */
40
41          ON ERROR
42              BEGIN;
43              PUT FILE(SCREEN) SKIP LIST("Invalid input. "
44                  !!"Try again.");
```

When you resume execution, you can specify a proceed count (as with the BREAKPOINT command) by including the COUNT keyword with the CONTINUE command. The proceed count applies to the current breakpoint. To set a proceed count of 2 for the breakpoint at line 34, type

```
> CONTINUE COUNT=2 l
*
Select currency code for US$ exchange:

    1   Belgian francs
    2   W. German marks
    3   Dutch guilders
    4   Italian lire
    5   Swiss francs
    6   Canadian dollars
    7   Saudi riyals
    8   Japanese yen

(Type 0 to end the program)
Enter the currency code: 1 l

Breakpoint trap at :EXCHANGE:64
```

When (and if) the debugger encounters the breakpoint at EXCHANGE:34, it decrements the proceed count by 1. When the proceed count equals 0 the SWAT software traps at the breakpoint and resets the proceed count to 1. (In our program, statement 34 will not execute again, so we can clear this breakpoint.)

As you can see, the program displayed the menu of currencies and requested a currency code. We supplied the value 1, which represents Belgian francs. Note that the I/O performed by the program does not appear in the audit file. You can use comments to remind yourself about messages that appear on the screen or entries that you make.

```
> CLEAR 34 ⟩
Cleared at :EXCHANGE:34 count=2
```

The AT keyword, when used with the CONTINUE command, allows you to alter program flow. You provide a locator that identifies the statement where execution is to resume. If you omit the AT keyword, the program continues by executing the statement where the breakpoint trap occurred. To redirect program flow, specify a locator within the breakpoint environment.

For example, to redisplay the program menu, we could enter

```
> CONTINUE AT LOOP ⟩

Select currency code for US$ exchange:

    1  Belgian francs
    2  W. German marks
    3  Dutch guilders
    4  Italian lire
    5  Swiss francs
    6  Canadian dollars
    7  Saudi riyals
    8  Japanese yen

(Type 0 to end the program)
Enter the currency code: 2 ⟩

Breakpoint trap at :EXCHANGE:64
```

We changed program flow so that the menu would appear again. This time we entered 2 as the currency exchange code.

Changing the program flow can result in unexpected results. Skipping or repeating code can affect the values of program variables. So, use the CONTINUE AT command with care.

Examining Variables

When a breakpoint trap occurs, you can examine the contents of program variables. To do this, use the TYPE command. You can examine a nonstack variable regardless of the working environment. Variables whose values are determined by the current stack, however, can be examined only if they are defined in the breakpoint environment.

```
> TYPE DAILY_RATES.RATE_DATE ↵
"811124"

> TYPE SELECTION ↵
2

> TYPE RESPONSE ↵
"<000>"

> TYPE INDX ↵
0

> TYPE DAILY_RATES.RATES(4) ↵
1194.000000
```

You can use a simple locator if the working environment is the breakpoint environment. If you have changed the working environment, you can still display breakpoint environment variables by qualifying the locator.

Be careful to interpret the displayed value in terms of the point of program execution. In other words, you must determine, based on the path of execution, whether the displayed value represents uninitialized storage, a value set in an earlier iteration, or a current value.

By default, the TYPE command normally displays the contents of a variable according to its declared data type. You can, however, specify the display format by using one of the following display mode keywords:

@BIT
@CHARACTER
@FLOAT
@INTEGER
@POINTER
@Rn (Only for the 32-bit AOS/VS SWAT debugger)

To request a particular display mode, follow the variable name in the TYPE command line with one of the above keywords, such as:

```
TYPE SELECTION @BIT
```

When you use the @BIT keyword with TYPE, the SWAT debugger displays the bit pattern occupying the variable's storage area. The size of the storage area determines the number of bits displayed. We display the contents of the variable SELECTION as a bit string.

```
> TYPE SELECTION @BIT ↵
"0000000000000010"
```

The @CHARACTER keyword interprets the same bit string as a series of ASCII characters. The SWAT debugger translates each group of eight bits into an ASCII value. If the result is a nonprintable character, the debugger displays it as one or more octal values enclosed in angle brackets.

```
> TYPE SELECTION @CHARACTER ↵
"<000><002>"
```

The @FLOAT keyword interprets the bit string as a floating-point value. The SWAT debugger displays the result in scientific notation. Depending on the size of the variable, the debugger displays it with either single or double precision. Refer to Chapter 4 for details.

```
> TYPE SELECTION @FLOAT ↵
9.0466E74
```

The keyword @INTEGER directs the SWAT debugger to interpret only the first 16 bits of the storage area as an integer (or fixed binary) value. The result can range from −32768 through +32767. The debugger displays the value as a 16-bit integer unless the variable's size is 2 words. A double-precision value appears as a 32-bit integer.

```
> TYPE SELECTION @INTEGER ↵
2
```

The @POINTER keyword causes the TYPE command to interpret the variable's bit string as one or more octal pointer values. Each 16-bit group translates into a pointer value of 6 octal digits.

```
> TYPE SELECTION @POINTER ↵
000002R8
```

If you are using the 32-bit AOS/VS SWAT debugger, you can use the @R keyword to display a numeric value in a specific radix. You can use radix values 2 (@R2) through 16 (@R16). See Appendix A for more information about this AOS/VS SWAT feature.

Displaying Expression Results

You can also use the TYPE command to display the result of an expression. The expression may contain one or more variables. When the expression contains mixed data types, the debugger follows the data type conversion rules shown in Table 1-2. Make sure that all elements of an expression are compatible.

Observe the next few commands and responses.

```
> TYPE SELECTION + 8 ↵
10

> TYPE SELECTION + INDX ↵
2

> TYPE INDX - SELECTION ↵
-2
```

The SWAT debugger works under the same restrictions as your programming language when evaluating an expression. For example, if an integer calculation produces an overflow condition in the programming language, the debugger's result will also overflow.

```
> % Integer division produces integer result.↵
> TYPE 34/2 ↵
17
> TYPE 35/2 ↵
17
> TYPE 999999999 * 10 ↵
1410065398
> % Overflow causes a meaningless result. ↵
> TYPE 999999999. * 10. ↵
9999999990.
```

The SWAT debugger converts single-precision values to double-precision values if the expression contains elements of both precisions.

The SWAT debugger assumes that all numeric constants and variables are decimal values. It interprets items with the data types file, entry, pointer, and label as octal values. If necessary, you can indicate the base for a constant by using the radix symbol. To do this, simply follow the value with R and the base. For example, 403R8 means 403_8.

```
> TYPE 403R8 ↵
259
```

There is one restriction for the TYPE command: automatic and parameter variables used in the expression must have a defined value within the breakpoint environment. Other variables are not subject to this restriction. You must fully qualify references to variables that are not directly accessible from the working environment.

Setting a Variable's Value

Sometimes you'll find it helpful to change the value of a variable to observe the effect the new value has on execution. The SET command lets you do this. Changing a variable's value can alter the program flow, modify results, or test the effectiveness of an algorithm or the ability of your program to handle unexpected input.

You can assign a value to a variable known within the breakpoint environment. The assigned value can be a single variable, constant, or an expression. When using variables in an expression, be sure that each carries a defined value within the breakpoint environment. The elements of an expression must have compatible data types. The expression result must have the same data type, storage class, and size as the receiving variable.

For example, we can change the values of exchange rates in the rate table.

```
> TYPE DAILY_RATES.RATES(6) ↵
1.186500
> SET DAILY_RATES.RATES(6) = DAILY_RATES.RATES(14) ↵
Old value: 1.186500
New value: 0.843400
```

The assigned value must be compatible with the data type and size of the receiving variable. For example:

```
> SET DAILY_RATES.RATES(6) = "abcd"↵
Old value: 0.843400

ILLEGAL DATA TYPE abcd
```

Displaying Information About Program Symbols

The DESCRIBE command returns information about a user-defined symbol. It is useful for distinguishing between similarly named symbols, and for displaying a variable's type or an array's bounds. The format of the response varies from one programming language to another, using appropriate terminology.

```
> DESCRIBE SELECTION ↵
SELECTION automatic at +8 words, fixed binary (1 word)

> DESCRIBE SCREEN ↵
SCREEN constant (at 000447R8) file constant (1 word)

> DESCRIBE DAILY_RATES ↵
DAILY_RATES automatic at +20 bytes, structure (102 byte)
    2 RATE_DATE at +0 bytes, character (6 byte)
    2 RATES at +6 bytes, fixed decimal (10,6) array (1:16)

> DESCRIBE :CONVERSION ↵
CONVERSION line 3 to line 33
```

If the argument refers to a program symbol that is unknown in the working environment, you must fully qualify the reference. You cannot display information about reserved words.

When You Need Help

If, as you are debugging, you need information about a SWAT command, keyword, or concept, the on-line help file can assist you. Just type HELP; a menu of topics appears. To obtain a concise explanation of a topic, type HELP followed by the topic name. The SWAT debugger gives you a summary of what you need to know.

```
> HELP ↵

topics are:

  @ALL          @BIT          @BREAK        @CHARACTER
  @FLOAT        @INTEGER      @MAIN         @POINTER
  ACTION        AT            AUDIT         BREAKPOINT
  BYE           CLEAR         CLI           COMMANDS
  COMMENTS      CONTINUE      COUNT         DEBUG
  DESCRIBE      DIRECTORY     ENVIRONMENT   EXECUTE
  EXPRESSIONS   HELP          KEYWORDS      LIST
  LOCATORS      MESSAGE       OFF           ON
  OPERATORS     PREFIX        PROMPT        SEARCHLIST
  SET           SPECIFIERS    TYPE          UNIQUENESS
  WALKBACK
```

```
> HELP BREAKPOINT ↓

  Command: BREAKPOINT
  or: BREAKPOINT locator [COUNT=expression] [ACTION="string"]
  BREAKPOINT lists all the breakpoints currently set,
  their environment, their counts (if any), their action
  (if any), and the current location (if any).
  BREAKPOINT locator sets a breakpoint at locator. If the
  optional COUNT=expression is supplied, the breakpoint will
  be given an initial count of expression. If the optional
  ACTION="string" is supplied, the SWAT command(s) in the
  string will be executed when the breakpoint is signalled.
  To change the count or action on a breakpoint already set,
  repeat this command with the new count expression and/or
  the new action string.

> HELP LOCATORS ↓

  Locators are identifiers used for naming a location in the
  user program. They can be in the form of a line number, a
  label, or a numeric label preceded by a "*". Examples are:
  19, START or *100. If there is more than one statement on
  a single line, they are given clause numbers of the form
  19.1, 19.2, 19.3, etc.
```

Interpreting Error Codes

The MESSAGE command is useful if your program includes parameters that return system error code values. If you use the PL/I ONCODE function or the FORTRAN 77 IOSTAT option, for example, you can use this command to obtain the error message for an error code.

Our program doesn't use error code parameters, but we can supply arbitrary values to show how this command works. The argument can be any expression that resolves to a positive integer value. You can provide more than one argument.

```
> MESSAGE 34 ↓
000042 CONSOLE DEVICE SPECIFICATION ERROR
```

The SWAT debugger evaluates the expression, then looks up and displays the corresponding system error message. You can use the R8 suffix to specify an octal argument value.

```
> MESSAGE 34R8 ↓
000034 APPEND AND/OR WRITE ACCESS DENIED

> MESSAGE SELECTION SELECTION+15 ↓
000002 CHANNEL NOT OPEN
000021 FILE SPACE EXHAUSTED
```

The SWAT Debugger and the CLI

During a debugging session, you may find it helpful to perform one or more CLI functions. To do this, include the /NOCONSOLE switch when calling the SWAT software. Then, in the debugging session, you can type CLI followed by a string of one or more CLI commands. Enclose the string in quotation marks or single apostrophes. The SWAT software creates a CLI process and passes your command string to it.

We did not use the /NOCONSOLE switch, so we cannot use the CLI command. A simple example, however, might look like this:

```
> CLI "TIME" ↵
14:03:32
```

If you omit the argument, the SWAT debugger creates a son CLI process and passes control to it. After receiving the CLI prompt, you can use the CLI directly. When you terminate this CLI process (by typing BYE, for example), you return to the SWAT session. To use this command, your user profile must allow you to create at least four son processes. You'll need to create more than four sons, however, if the CLI generates additional processes.

The Directory and Search List

To make source files accessible to the SWAT debugger, you may need to change your working directory or search list. The DIRECTORY and SEARCHLIST commands help you do this. These commands let you display and set the working directory and search list for the debugging session.

If you type DIRECTORY with no argument, the SWAT debugger displays the working directory. Initially, this is the directory you were in when you invoked the debugger. For example:

```
> DIRECTORY ↵
:UDD:TOM
```

To change the working directory, supply a pathname enclosed in quotation marks or single apostrophes. The new directory pertains to the SWAT debugging session only. When the session ends, you return to the directory from which you invoked the SWAT software. The working directory does not affect your program execution in any way.

The SEARCHLIST command works in the same way. Typing it alone displays the current search list, which is the same as the one in effect when you invoked the SWAT debugger.

```
> SEARCHLIST ↵
:UDD:TOM,:UTIL,:LINKS,:MACROS
```

To set a new search list, provide directory arguments enclosed in quotation marks or apostrophes. You must enter each directory that you want to appear on the search list.

```
> SEARCHLIST ":UDD:TOM" ":UTIL" ":COMMON" ↵
:UDD:TOM,:UTIL,:COMMON
```

SWAT Command Files

The SWAT debugger lets you set up files containing SWAT commands. These command files can serve as debugger utilities. The command file may contain any number of SWAT commands with predefined arguments. When you are in a debugging session, you can call a command file by using the EXECUTE command.

If, for example, we built a file called DBUG.LIB that contained the statements

```
ENVIRONMENT
LIST @BREAK
```

we could execute these commands within a debugging session by typing

```
> EXECUTE "DBUG.LIB" )
```

As the SWAT software executes each command in the file, it displays the command preceded by a double prompt.

```
> EXECUTE "DBUG.LIB" )
>> ENVIRONMENT
:EXCHANGE
>> LIST @BREAK
54              PUT FILE(SCREEN) SKIP(2) EDIT
55                  ("1","Belgian francs","2","W. German marks","3",
56

.

73          IF RESPONSE = "Y" THEN INDX = SELECTION;
74          ELSE IF RESPONSE = "y" THEN INDX = SELECTION;
```

The SWAT debugger then displays the working environment, and lists the lines around the current breakpoint trap (or, if none has occurred, lines 1 through 21 of the main procedure). After executing the command file, the SWAT prompt appears.

The Null Command Response

A null command is NEW LINE pressed alone. How the debugger responds to a null command depends on the current null command response setting. By default, the SWAT debugger performs the LIST @BREAK command when you enter NEW LINE only. (If a breakpoint trap has not yet occurred, the debugger lists the first 21 lines of the main procedure.)

You can set the null command response by using the PROMPT command. Follow this command with a string containing one or more SWAT commands (with any necessary arguments). Enclose the string in quotation marks or apostrophes. Separate each command in the string with a semicolon.

If you simply type PROMPT, the debugger displays the current null command response setting.

```
> PROMPT ↵
"LIST @BREAK"

> PROMPT "CONTINUE" ↵
> ↵

Do you want to convert US$ into Foreign currency?
Enter 'Y' or 'N': Y ↵

ENTER US$: 250 ↵

Breakpoint trap at :CONVERSION:16
```

The null response command lets you define your own command sequence. Then, simply press NEW LINE to initiate the predefined sequence. The SWAT PROMPT command differs from the CLI PROMPT command in that the prompt string is executed only in response to a null command, not every time you enter a command.

Ending the Session

We've shown you all the SWAT commands but one: the BYE command. The BYE command terminates your program, then terminates the SWAT session, and returns you to the CLI (or other calling process). Use this command only to terminate the SWAT session *before* your program completes execution.

If a CONTINUE command results in your program running to completion or terminating abnormally, the debugging session automatically terminates afterwards. You may find it helpful to place a breakpoint at each exit statement in the program. This lets you to decide whether to continue the session by redirecting program flow, or to let the program terminate.

```
> BYE ↵

SWAT TERMINATED

)
```

Debugging the Exchange Program

Now that we've shown you how each SWAT command works, let's take the same program and actually debug it. The operator has reported that the program does strange things when trying to exchange American dollars with Japanese yen. At times it fails to print a dollar sign with the dollar amount when displaying the results of an exchange. Otherwise the program appears to work fine.

First we will run through the program performing a currency exchange that apparently works. We'll arbitrarily choose Swiss francs as our guinea pig. We'll exchange Swiss francs for American dollars while observing the values held by key variables. Next we'll perform the exchange in the other direction (changing American dollars to Swiss francs) and check the variables again. If everything appears normal (we don't expect a problem with anything but Japanese yen), we'll continue by doing the same exchanges with Japanese yen. The SWAT debugger should help us locate a discrepancy.

Before we begin debugging the EXCHANGE program, you may want to review the description of the program at the beginning of the chapter.

We'll start by invoking the SWAT debugger as we did earlier in this chapter. So that we don't add information to the current audit file (called EXCHANGE.AU), we'll specify a unique name for a new audit file. In the CLI we enter the command

```
XEQ SWAT/AUDIT=AUDIT.YEN EXCHANGE ↵
```

The SWAT software begins execution, and creates the file AUDIT.YEN, where it will record our debugging session. Remember that the audit file does not record any I/O performed by the program. Messages written by the program to the operator and the operator's responses do not appear in the audit file. Figure 3-5 lists the AUDIT.YEN file.

```
SWAT REVISION 02.00 ON 11/24/81 AT 10:41:49
PROGRAM – :UDD:TOM:EXCHANGE
>
```

The SWAT debugger is now ready for the first command. We are in the main program module EXCHANGE. First we set breakpoints at key statements. (Refer to the program listing in Figure 3-1.) Breakpoints at statements 28 and 43 allow us to observe the state of the program when an ON unit occurs, in this case an ON ERROR or ON ENDFILE. We don't set the breakpoint at the ON statement itself because the program encounters this statement sequentially. The program executes statements within the BEGIN block only when the specified condition occurs.

We also set breakpoints at statements 64 and 77. Statement 64 calls the subroutine CHECK. We can verify that the parameter passed to the subroutine carries the same value we entered from the keyboard. Similarly, by setting a breakpoint at statement 77, we can observe the parameters passed to the :CONVERSION routine.

In each case, we'll include an action string to display the breakpoint statement. This will clarify where execution has trapped.

```
> % Set breakpoints at each ON unit and each subroutine call. ↵
> BREAKPOINT 28 ACTION="LIST 28" ↵
Set at :EXCHANGE:27:28 action="LIST 28"

> BREAKPOINT 43 ACTION="LIST 43" ↵
Set at :EXCHANGE:42:43 action="LIST 43"

> BREAKPOINT 64 ACTION="LIST 64" ↵
Set at :EXCHANGE:64 action="LIST 64"

> BREAKPOINT 77 ACTION="LIST 77" ↵
Set at :EXCHANGE:77 action="LIST 77"
```

Now we set breakpoints at statements 13, 16, 24, and 26 in the external routine :CONVERSION. At statement 13 we can observe the received parameters and determine which of the two DO blocks will execute. Statement 16 performs the exchange calculation from dollars to foreign currency. We can display the operands and perform the calculation before the program does. The results should match.

Statement 24 allows us to check the type of currency requested by the program. We can also verify that the calculated array subscript is within proper range. Finally, we set a breakpoint at statement 26, which performs the exchange calculation from foreign currency to American dollars. Again we can display the operands and verify our result with the program's.

```
> % Set breakpoints in the subroutine CONVERSION. ⤶

> BREAKPOINT :CONVERSION:13 ACTION="LIST 13" ⤶
Set at :CONVERSION:13 action="LIST 13"

> BREAKPOINT :CONVERSION:16 ACTION="LIST 16" ⤶
Set at :CONVERSION:16 action="LIST 16"

> BREAKPOINT :CONVERSION:24 ACTION="LIST 24" ⤶
Set at :CONVERSION:24 action="LIST 24"

> BREAKPOINT :CONVERSION:26 ACTION="LIST 26" ⤶
Set at :CONVERSION:26 action="LIST 26"
```

Now we're ready to go. We begin execution of the EXCHANGE program with the SWAT command:

```
> CONTINUE ⤶
```

The program starts execution in the main program module and displays the exchange menu:

```
Select currency code for US$ exchange:

    1  Belgian francs
    2  W. German marks
    3  Dutch guilders
    4  Italian lire
    5  Swiss francs
    6  Canadian dollars
    7  Saudi riyals
    8  Japanese yen

(Type 0 to end the program)
Enter the currency code:
```

We enter 5 to make a test run with Swiss francs.

```
Enter the currency code: 5 ⤶
```

The program encounters a breakpoint; the debugger reports its location and, because of the action string, lists the source code statement.

```
Breakpoint trap at :EXCHANGE:64 action="LIST 64"
64C     CALL CHECK(SELECTION);
```

Because the audit file does not record any program I/O, we enter comments to document our interactions with the program. Otherwise the audit file will contain no record of the values we enter or the results output by the program.

```
> % We entered a currency code of 5 -- the code for Swiss francs. ⤶
```

The variable SELECTION should contain the value 5, which we entered from the keyboard.

```
> TYPE SELECTION ↵
5
> % SELECTION is the currency code variable. ↵
```

So far everything appears as it should. We continue execution of the program:

```
> CONTINUE ↵

Do you want to convert US$ into Foreign currency?
Enter 'Y' or 'N':
```

We answer N (No), to convert Swiss francs into American dollars. (After we check the results, we will go through the program again, but convert American dollars into Swiss francs.)

```
Do you want to convert US$ into Foreign currency?
Enter 'Y' or 'N': N ↵

Breakpoint trap at :EXCHANGE:77 action= "LIST 77"
77C CALL CONVERSION(INDX, RATES(INDX));

> % We responded No to the exchange direction question. ↵
```

We can display the contents of the parameters INDX and RATES(INDX). INDX points to the exchange rate in the table. Because we're converting Swiss francs into dollars, we want the rate in the second half of the table. Therefore, INDX should equal 5 (the currency code) plus 8. Using this subscript, we can also observe the exchange rate to be used in the calculation.

```
> TYPE INDX ↵
13

> TYPE DAILY_RATES.RATES(13) ↵
0.556600
> % The value of INDX is correct. We displayed the exchange rate. ↵
```

Remember that you must fully qualify a structure member when using the SET or TYPE command.

We continue the EXCHANGE program:

```
> CONTINUE ↵

Breakpoint trap at :CONVERSION:13 action= "LIST 13"
13C     IF CODE < 8 THEN DO;
```

We are now in the CONVERSION subroutine. This routine performs the currency exchange calculation.

```
> TYPE CODE ↵
13
```

The variable CODE in the CONVERSION routine corresponds to INDX in the main program. The program should execute the second DO block, which converts foreign currency to American dollars. The next breakpoint we should encounter is statement 24 (skipping the block containing the breakpoint at statement 16).

```
> CONTINUE ↵

Breakpoint trap at :CONVERSION:24 action= "LIST 24"
24C     PUT FILE(SCREEN) SKIP LIST("ENTER ",CURRENCY(CODE-8),": ");
```

The program is executing the correct block. It is about to request the number of Swiss francs we want to convert to American dollars. We can check the calculated subscript value to ensure that it is within the bounds of the array.

```
> TYPE CODE-8 ⤶
5
> TYPE CURRENCY(5) ⤶
"SF"
```

The subscript 5 points to the array entry carrying the symbol for Swiss francs.

```
> CONTINUE ⤶

ENTER SF :
```

We enter the value 250 for the number of Swiss francs.

```
ENTER SF : 250 ⤶

Breakpoint trap at :CONVERSION:26 action= "LIST 26"
26C     DVAL = FVAL * RATE;

> % We entered 250 Swiss francs for the exchange. ⤶
```

This statement performs the calculation. We can display the contents of each operand and have the SWAT debugger perform the calculation. The debugger's result should be the same as the value the program displays.

```
> TYPE FVAL ⤶
250.00
> % This is the number of Swiss francs we entered. ⤶

> TYPE RATE ⤶
0.556600
> % This is the same rate we observed earlier. ⤶

> TYPE 250.00 * .556600 ⤶
139.15
> % The program should output $139.15 as its result. ⤶
```

Now we continue execution of the program. EXCHANGE should display an exchange value of $139.15 for 250 Swiss francs.

```
> CONTINUE ⤶

    250.00 SF equivalent to: 139.00 US
```

We have a problem. The program's result is off by 15 cents, a small but important amount to the company's accounting department. At the moment it's hard to explain what happened; all the variables we examined held the expected values. If we keep this error in mind, other evidence may turn up as we exchange currency in the other direction.

The program did not display a dollar sign with the output dollar amount. If we look at the output statement beginning on line 27 of the :CONVERSION module, we find the answer to the problem.

Although the program declares FVAL as a picture variable with a floating dollar sign, the PUT FILE statement beginning on line 27 of the :CONVERSION module does not provide a picture for output, only a floating-point designation: F(11,2). Therefore, we must make the following change to this statement:

```
PUT FILE(SCREEN) EDIT(FVAL, CURRENCY(CODE-8),
"equivalent to: ",DVAL," US")
(SKIP(2),X(5),F(11,2),X(1),A(4),A(17),P"$$$$$$$$9V.99",A(3),
SKIP(2));
```

The program gives us the currency menu again. This time through we again specify Swiss francs, but perform the exchange in the other direction, that is, we convert American dollars into Swiss francs. As before, we examine the contents of the key variables.

Select currency code for US$ exchange:

1 Belgian francs
2 W. German marks
3 Dutch guilders
4 Italian lire
5 Swiss francs
6 Canadian dollars
7 Saudi riyals
8 Japanese yen

(Type 0 to end the program)
Enter the currency code: 5 ⟩

Breakpoint trap at :EXCHANGE:64 action= "LIST 64"
64C CALL CHECK(SELECTION);

Before continuing, we insert a comment in the audit file to explain what just happened.

```
> % The output value was $139.00, but should have been $139.15. ⟩
> % Change the output statement, line 29 of the :CONVERSION module. ⟩
> % -- The second F(11,2) should read P"$$$$$$$$9V.99" ⟩
> % The program displayed the menu. We entered code 5 again. ⟩

> TYPE SELECTION ⟩
5
> % This corresponds to the currency code we input at the keyboard. ⟩

> CONTINUE ⟩
```

Do you want to convert US$ into Foreign currency?
Enter 'Y' or 'N': Y ⟩

Breakpoint trap at :EXCHANGE:77 action= "LIST 77"
77C CALL CONVERSION(INDX, RATES(INDX));

```
> % We responded Yes to the exchange direction question. (US$ to SF) ⟩

> TYPE INDX ⟩
5
> % INDX points to exchange rate for SF in first half of table. ⟩

> TYPE DAILY_RATES.RATES(5) ⟩
1.772500
> % This rate should also appear as RATE in the CONVERSION routine. ⟩
```

```
> CONTINUE ↵

Breakpoint trap at :CONVERSION:13 action= "LIST 13"
13C     IF CODE < 8 THEN DO;

> % CODE should equal 5 and correspond to INDX in the calling routine. ↵

> TYPE CODE ↵
5
```

The program should execute the first DO group in the CONVERSION routine. We expect the next breakpoint to be at statement 16 in this module.

```
> CONTINUE ↵

ENTER US$: 250 ↵

Breakpoint trap at :CONVERSION:16 action= "LIST 16"
16C     FVAL = DVAL * RATE;

> % The program asked for the US dollar amount; we entered 250. ↵
```

Again we can display the values of the operands prior to the program's completion of this statement.

```
> % DVAL holds the number of American dollars to be converted. ↵
> TYPE DVAL ↵
" $2.50"
```

Another unexpected problem! We entered 250 dollars but the program has stored it as $2.50. The program has incorrectly built the character string for the picture variable DVAL. Chances are that we have not given an accurate picture description. Let's look at the declaration for DVAL.

```
> LIST 1 10 ↵
1
2     CONVERSION:
3         PROCEDURE(CODE, RATE);
4
5         DECLARE CODE FIXED BINARY;
6         DECLARE RATE FIXED DECIMAL(10,6);
7         DECLARE CURRENCY(8) CHARACTER(4) VARYING STATIC INTERNAL
8             INIT("BF","DM","gld","Lit","SF","Can$","SRI","Y");
9         DECLARE FVAL FIXED DECIMAL(11,2);
10        DECLARE DVAL PICTURE "$$$$$$$$9.99";
```

The declaration of DVAL inserts a decimal point in a fixed position, but ignores the decimal point we entered. To remedy this we must change the declaration to read:

```
DECLARE DVAL PICTURE "$$$$$$$$9V.99";
```

This picture error explains why we lost the 15 cents in the previous exchange operation. The program truncated the digits that fell to the right of the imposed decimal place. We can't make the change to the declaration within the SWAT session. For now, we must interpret the program results with this flaw in mind.

```
> TYPE RATE ↵
1.772500
> % The rate matches the rate we observed in the EXCHANGE module. ↵

> TYPE 1.7725 * 250. ↵
443.125

> CONTINUE ↵

    250.00 US equivalent to: 443.12 SF
```

The output American dollar value is correct, as is the output value for Swiss francs. Although the program stored the value incorrectly, in this case it performed the calculation correctly.

We're ready to go through the program again. We found two small problems, but they don't explain why there should be difficulty when working with Japanese yen. Our strategy remains as before, but this time we work with yen instead of Swiss francs.

```
Select currency code for US$ exchange:

    1  Belgian francs
    2  W. German marks
    3  Dutch guilders
    4  Italian lire
    5  Swiss francs
    6  Canadian dollars
    7  Saudi riyals
    8  Japanese yen

(Type 0 to end the program)
Enter the currency code: 8 ↵

Breakpoint trap at :EXCHANGE:64 action= "LIST 64"
64C     CALL CHECK(SELECTION);

> % We entered currency code 8 for Japanese yen. ↵

> TYPE SELECTION ↵
8
> % SELECTION holds the correct value. ↵

> CONTINUE ↵

Do you want to convert US$ into Foreign currency?
Enter 'Y' or 'N':
```

We respond N (No) and observe what happens. The program uses the eighth entry in the second half of the exchange rate table. Therefore, the index value (INDX) should be 16.

```
Do you want to convert US$ into Foreign currency?
Enter 'Y' or 'N': N ↵

Breakpoint trap at :EXCHANGE:77 action= "LIST 77"
77C     CALL CONVERSION(INDX, RATES(INDX));

> % We entered N to the exchange direction question. ↵

> TYPE INDX ↵
16
```

The index value is within the bounds of the array; in fact, it is at the uppermost limit of the array. Because we entered the largest possible currency code, and the index value does not exceed the upper limits of the array, we can feel confident that this is not the problem.

```
> TYPE DAILY_RATES.RATES(INDX) ↵
0.004471
> % This should be the rate we see in the calculation statement. ↵

> CONTINUE ↵

Breakpoint trap at :CONVERSION:13 action="LIST 13"
13C     IF CODE < 8 THEN DO;

> TYPE CODE ↵
16
> % This corresponds to INDX; it is correct. ↵
```

We expect the program to execute the second DO group (foreign to American dollars). The next breakpoint should occur at statement 24 of the CONVERSION routine.

```
> CONTINUE ↵

Breakpoint trap at :CONVERSION:24 action="LIST 24"
24C     PUT FILE(SCREEN) SKIP LIST("ENTER ",CURRENCY(CODE-8),":");

> TYPE CODE-8 ↵
8
> % The subscript is correct; it points to the symbol for yen. ↵

> CONTINUE ↵

ENTER Y : 560 ↵

Breakpoint trap at :CONVERSION:26 action="LIST 26"
26C     DVAL = FVAL * RATE;

> % We entered the value 560 for the number of yen. ↵
```

We display the values of the operands.

```
> TYPE FVAL ↵
560.00

> TYPE RATE ↵
0.004471
> % This value corresponds to the rate displayed earlier. ↵

> TYPE 560. * .004471 ↵
2.50376
```

The result of the program should be $2.50. Let's compare our result with the program's.

```
> CONTINUE ↵

    560.00 Y equivalent to: 2.00 US
```

Again we lost the cents because of the problem with the declaration of DVAL. We lost the dollar sign because of the error in the output statement.

HELP

Displays information about SWAT topics.

Format

HELP *[topic] ...*

where:

topic is an entry from the help menu.

Description

To get quick, on-line information when you're in a SWAT debugging session, simply type HELP. The SWAT debugger displays a menu of topics you can obtain specific information for.

For information about a particular topic, follow the HELP command with the name of the topic. You can abbreviate any help topic. If the abbreviation is not unique, the debugger displays the topics that match it. You can then select from these (or any other) topics.

You can append more than one topic to the HELP command.

Figure 4-1 illustrates a sample help menu. Later revisions of the SWAT debugger may list additional topics. Simply type HELP to display the current menu.

```
topics are:

      @ALL            @BIT            @BREAK          @CHARACTER
      @FLOAT          @INTEGER        @MAIN           @POINTER
      ACTION          AT              AUDIT           BREAKPOINT
      BYE             CLEAR           CLI             COMMANDS
      COMMENTS        CONTINUE        COUNT           DEBUG
      DESCRIBE        DIRECTORY       ENVIRONMENT     EXECUTE
      EXPRESSIONS     HELP            KEYWORDS        LIST
      LOCATORS        MESSAGE         OFF             ON
      OPERATORS       PREFIX          PROMPT          SEARCHLIST
      SET             SPECIFIERS      TYPE            UNIQUENESS
      WALKBACK
```

Figure 4-1. The Help Menu

Errors

UNKNOWN TOPIC topic

The specified topic is not listed on the help menu.

The program will execute the second DO group. This is an error. The second calculation exchanges foreign currency into American dollars. But we want to convert American dollars into yen. The statement on line 13 should direct execution to the first block when CODE is equal to 8.

```
> % Change statement 13 in :CONVERSION. It should read: ↓
> % IF CODE < 9 THEN DO; ↓
```

By continuing the program we should see why the operator received unexpected output.

```
> CONTINUE ↓

Breakpoint trap at :CONVERSION:24 action= "LIST 24"
24C      PUT FILE(SCREEN) SKIP LIST("ENTER ",CURRENCY(CODE-8),": ");

> TYPE CODE-8 ↓
0
```

Here is the reason for the runaway output. The subscript becomes 0 causing the program to use the wrong storage area when executing the output statement on line 24.

It appears that we have found the cause of the problem reported by the operator. We can continue the program by redirecting execution to the correct algorithm.

```
> CONTINUE AT 14 ↓

ENTER US$: 250 ↓

Breakpoint trap at :CONVERSION:16 action= "LIST 16"
16C      FVAL = DVAL * RATE;

> TYPE DVAL ↓
"     $2.50"
> TYPE RATE ↓
225.900000

> TYPE 250. * 225.90 ↓
56475.

> CONTINUE ↓

     250.00 US equivalent to: 56475.00 Y

Select currency code for US$ exchange:

     1  Belgian francs
     2  W. German marks
     3  Dutch guilders
     4  Italian lire
     5  Swiss francs
     6  Canadian dollars
     7  Saudi riyals
     8  Japanese yen

(Type 0 to end the program)
Enter the currency code:
```

Examples

```
SWAT REVISION 02.00 ON 10/04/81 AT 14:45:16
PROGRAM – :UDD:TOM:EXCHANGE
> ENVIRONMENT ⤸
:EXCHANGE

> ENV CHECK ⤸
:EXCHANGE:CHECK

> ENV :CONVERSION ⤸
:CONVERSION

> env :EXCHANGE:27 ⤸
:EXCHANGE:27

> ENV @MAIN ⤸
:EXCHANGE

> ENVIRONMENT ⤸
:.MAIN

> ENV :PAYMENT ⤸
:PAYMENT

> ENV @MAIN ⤸
:.MAIN
```

```
> BREAKPOINT
Set at :EXCHANGE:24
Set at :EXCHANGE:34
Set at :EXCHANGE:50
Set at :EXCHANGE:62
Set at :EXCHANGE:64
Set at :EXCHANGE:CHECK:87
Set at :CONVERSION:13
Set at :CONVERSION:16
Set at :CONVERSION:26
> ENVIRONMENT @MAIN
:EXCHANGE
> BREAKPOINT 28
Set at :EXCHANGE:27:28
> ENVIRONMENT 42
:EXCHANGE:42
> BREAKPOINT 43
Set at :EXCHANGE:42:43
> BREAKPOINT 71 COUNT=5
Set at :EXCHANGE:71 count=5
> BREAKPOINT 71 COUNT=4
Reset at :EXCHANGE:71 count=4
> BREAKPOINT 71 ACTION = "LIST 71"
Reset at :EXCHANGE:71 count=4 action="LIST 71"
> CLEAR 71
Cleared at :EXCHANGE:71 count=4 action="LIST 71"
> CLEAR 28 43 62 87
Cleared at :EXCHANGE:27:28
Cleared at :EXCHANGE:42:43
Cleared at :EXCHANGE:62
Cleared at :EXCHANGE:CHECK:87
> BREAKPOINT
Set at :EXCHANGE:24
Set at :EXCHANGE:34
Set at :EXCHANGE:50
Set at :EXCHANGE:64
Set at :CONVERSION:13
Set at :CONVERSION:16
Set at :CONVERSION:26
> CLEAR :CONVERSION:13
Cleared at :CONVERSION:13
> CLEAR @ALL
Cleared at :EXCHANGE:24
Cleared at :EXCHANGE:34
Cleared at :EXCHANGE:64
Cleared at :EXCHANGE:50
Cleared at :CONVERSION:16
Cleared at :CONVERSION:26
> BREAKPOINT
```

Figure 3-4. The Audit File EXCHANGE.AU. (continued)

DIRECTORY

Displays or sets the current directory.

Format

DIRECTORY *["pathname"]*

where:

pathname identifies the directory to become the working directory.

Description

Use this command to set or display the working directory for the SWAT debugging session. Initially, the working directory is the one from which you called the SWAT debugger. (This directory applies only to the current debugging session. When you terminate the SWAT process, you return to the directory in which you invoked the SWAT debugger.)

To display the current working directory, enter the command without an argument.

To set the working directory, follow the DIRECTORY command with a pathname. Enclose the pathname in quotation marks or apostrophes.

A confirmation message appears after you set the working directory.

This command does not affect your program's directory.

Errors

COMMAND ENDS ILLEGALLY

You probably omitted the final quotation mark or apostrophe to set off the string argument.

INVALID EXTRA ARGUMENTS ON COMMAND

You can specify only one directory as the working directory. Use the SEARCHLIST command if you need access to other directories.

NONUNIQUE COMMAND command

Your command abbreviation is not unique. The shortest acceptable abbreviation for the DIRECTORY command is DI.

UNEXPECTED OPERATOR operator

You probably used a pathname prefix without enclosing the pathname in quotation marks or apostrophes.

Examples

```
> DIRECTORY ↵
:UDD:TOM

> dir "TEST_PROGRAMS" ↵
:UDD:TOM:TEST_PROGRAMS

> DIR "^" ↵
:UDD:TOM
```

```
> LIST 20 28
20              OPEN FILE(RATE_FILE) STREAM INPUT;
21
22      /* Read in today's exchange rates.  Compare date with system date. */
23
24C             GET FILE(RATE_FILE) LIST(DAILY_RATES);
25
26              ON ENDFILE(RATE_FILE)
27              BEGIN;
28                      PUT FILE(SCREEN) SKIP LIST("NO RATES AVAILABLE.");
> LIST @BREAK
14                      2 RATES(16) FIXED DECIMAL(10,6);
15              DECLARE RESPONSE CHARACTER(1);
16              DECLARE INDX FIXED BINARY;
17
18              OPEN FILE(SCREEN) STREAM OUTPUT PRINT TITLE("@OUTPUT");
19              OPEN FILE(KEYBOARD) STREAM INPUT TITLE("@INPUT");
20              OPEN FILE(RATE_FILE) STREAM INPUT;
21
22      /* Read in today's exchange rates.  Compare date with system date. */
23
24C             GET FILE(RATE_FILE) LIST(DAILY_RATES);
25
26              ON ENDFILE(RATE_FILE)
27              BEGIN;
28                      PUT FILE(SCREEN) SKIP LIST("NO RATES AVAILABLE.");
29                      STOP;
30              END; /*BEGIN Block for End of File condition */
31
32      /* Compare file's date with the system date. */
33
34B             IF RATE_DATE ^= DATE() THEN DO;
> CONTINUE

Breakpoint trap at :EXCHANGE:34
```

Figure 3-4. The Audit File EXCHANGE.AU. (continued)

DESCRIBE

Displays information about a program symbol.

Format

DESCRIBE symbol ...

where:

symbol is a user-defined program symbol.

Description

Use the DESCRIBE command to obtain information about a user-defined program symbol such as a variable, label, constant, or structure. To obtain information about a symbol in another environment, precede the symbol name with the appropriate environment specifier.

To display information about a program symbol that is known within the working environment, simply supply that symbol reference as an argument to the command. If the symbol is not defined in the working environment, you must fully qualify the reference.

The terminology used in the description depends on the programming language you are using. The format of the description differs slightly between the AOS and AOS/VS versions.

Errors

COMMAND REQUIRES ARGUMENT(S)

You must provide at least one symbol as an argument to this command.

UNDEFINED SYMBOL symbol

The specified symbol does not exist.

Examples

```
> DESCRIBE :CONVERSION ↵
CONVERSION line 3 to line 33

> desc SELECTION ↵
SELECTION automatic at +8 words, fixed binary (1 word)

> describe DAILY_RATES ↵
DAILY_RATES automatic at +20 bytes, structure (102 byte)
     2 RATE_DATE at +0 bytes, character (6 byte)
     2 RATES at +6 bytes, fixed decimal (10,6) array (1:16)

> desc SKREEN ↵

UNDEFINED SYMBOL SKREEN
```

```
> TYPE SELECTION + 8
10
> TYPE SELECTION + INDX
2
> TYPE INDX - SELECTION
-2
> %Integer division produces integer result.
> TYPE 34/2
17
> TYPE 35/2
17
> TYPE 999999999 * 10
1410065398
>% Overflow causes a meaningless result.
> TYPE 999999999. * 10.
9999999990.
> TYPE 403R8
259
> TYPE DAILY_RATES.RATES(6)
1.186500
> SET DAILY_RATES.RATES(6) = DAILY_RATES.RATES(14)
Old value: 1.186500
New Value: 0.843400
> SET DAILY_RATES.RATES(6) = "abcd"
Old value: 0.843400

ILLEGAL DATA TYPE  abcd
> DESCRIBE SELECTION
SELECTION automatic at +8 words, fixed binary (1 word)
> DESCRIBE SCREEN
SCREEN constant (at 000447R8) file constant (1 word)
> DESCRIBE DAILY_RATES
DAILY_RATES automatic at +20 bytes, structure (102 byte)
  2 RATE_DATE at +0 bytes, character (6 byte)
  2 RATES at +6 bytes, fixed decimal (10,6) array (1:16)
> DESCRIBE :CONVERSION
CONVERSION line 3 to line 33
> HELP

topics are:

        @ALL            @BIT            @BREAK          @CHARACTER
        @FLOAT          @INTEGER        @MAIN           @POINTER
        ACTION          AT              AUDIT           BREAKPOINT
        BYE             CLEAR           CLI             COMMANDS
        COMMENTS        CONTINUE        COUNT           DEBUG
        DESCRIBE        DIRECTORY       ENVIRONMENT     EXECUTE
        EXPRESSIONS     HELP            KEYWORDS        LIST
        LOCATORS        MESSAGE         OFF             ON
        OPERATORS       PREFIX          PROMPT          SEARCHLIST
        SET             SPECIFIERS      TYPE            UNIQUENESS
        WALKBACK
```

Figure 3-4. The Audit File EXCHANGE.AU. (continued)

DEBUG

Calls the AOS or AOS/VS user debugger.

Format

DEBUG

Description

Use this command to call the AOS or AOS/VS user debugger within a SWAT session. The DEBUG command assumes that the program you are debugging has a terminal assigned to it. If the SWAT invocation command included the /NOCONSOLE switch, do not use the DEBUG command; a fatal error will occur.

To grant the user debugger write access to the program, include the /DEBUG switch in the SWAT invocation command. (If you use this switch, your debugging session begins in the user debugger.)

To exit the user debugger, enter the appropriate command:

P ↵	Exits the AOS user debugger
ESC R	Exits the AOS/VS user debugger

When you exit the user debugger, the SWAT prompt appears.

NOTE: Auditing does not record any I/O performed by a program called from the SWAT debugger. Your interactions with the user debugger will not appear in an opened audit file.

Errors

COMMAND DOES NOT ACCEPT ARGUMENT(S)

There is no valid argument to this command. Enter only DEBUG (or its abbreviation).

NONUNIQUE COMMAND command

Your command abbreviation is not unique. The shortest acceptable abbreviation for DEBUG is DEB.

Examples

```
) X SWAT EXCHANGE ↵
SWAT REVISION 02.00 ON 10/05/81 AT 12:57:10
PROGRAM -- :UDD:TOM:EXCHANGE
> DEBUG ↵
AOS USER DEBUGGER, REV 3.40
WRITE ACCESS DENIED
+
      .
      .
      .

+P ↵
>
```

```
>>LIST @BREAK
54                      PUT FILE(SCREEN) SKIP(2) EDIT
55                          ("1","Belgian francs","2","W. German marks","3",
56                          "Dutch guilders","4","Italian lire","5","Swiss francs",
57                          "6","Canadian dollars","7","Saudi riyals","8",
58                          "Japanese yen","(Type 0 to end the program)",
59                          "Enter the currency code: ")
60                          (8(X(5),A(1),X(3),A(16),SKIP),SKIP,A(27),SKIP,A(26));
61
62                      GET FILE(KEYBOARD) LIST(SELECTION);
63
64C                     CALL CHECK(SELECTION);
65
66      /* Request type of exchange */
67
68              PUT FILE(SCREEN) SKIP LIST("Do you want to convert US$ into"
69                      !!" Foreign currency?");
70              PUT FILE(SCREEN) SKIP LIST("Enter 'Y' or 'N': ");
71              GET FILE(KEYBOARD) LIST(RESPONSE);
72
73              IF RESPONSE = "Y" THEN INDX = SELECTION;
74              ELSE IF RESPONSE = "y" THEN INDX = SELECTION;
> PROMPT
"LIST @BREAK"
> PROMPT "CONTINUE"
>

Breakpoint trap at :CONVERSION:16
> BYE

SWAT TERMINATED
```

Figure 3-4. The Audit File EXCHANGE.AU. (concluded)

Errors

CANNOT CONTINUE AT locator

The specified locator does not point to an executable statement.

INVALID EXTRA ARGUMENTS ON COMMAND

You supplied unnecessary arguments or omitted the equal sign after the COUNT keyword.

INVALID NUMBER number

The specified count number is not an unsigned integer value in the range 1 through 32767.

NONUNIQUE COMMAND command

Your command abbreviation is not unique. The shortest acceptable abbreviation for CONTINUE is CO.

NOT AT A BREAKPOINT

Either program execution has not yet begun, or you used the COUNT or AT option inappropriately. You cannot use the COUNT option if you've cleared the current breakpoint, nor can you use the AT option with the first CONTINUE command.

NOT IN BREAKPOINT ENVIRONMENT

The specified locator points to a statement outside the breakpoint environment. You can continue execution only within the breakpoint environment.

SOME DATA MAY NOT BE CURRENT

This is a warning message. One or more accumulators held data values when the trap occurred.

NOTE: If a trap occurs at a breakpoint that has an attached action string, additional errors may result from the commands contained in the string.

Examples

```
> CONTINUE ↵

Breakpoint trap at :EXCHANGE:34

> co ↵

Breakpoint trap at :EXCHANGE:50 action= "LIST 50"
50C     LOOP:     DO WHILE("1"B);

> CON COUNT=4 ↵

Breakpoint trap at :EXCHANGE:64

> C ↵

NONUNIQUE COMMAND C

> CONTINUE AT 50 ↵

Breakpoint trap at :EXCHANGE:64
```

```
> CONTINUE

Breakpoint trap at :CONVERSION:26 action="LIST 26"
26C             DVAL = FVAL * RATE;
> % We entered 250 Swiss francs for the exchange.
> TYPE FVAL
250.00
> % This is the number of Swiss francs we entered.
> TYPE RATE
0.556600
> % This is the same rate we observed earlier.
> TYPE 250.00 * .556600
139.15
> % The program should output $139.15 as its result.
> CONTINUE

Breakpoint trap at :EXCHANGE:64 action="LIST 64"
64C                     CALL CHECK(SELECTION);
> % The output value was $139.00, but should have been $139.15.
> % Change the output statement, line 29 of the :CONVERSION module.
> % -- The second F(11,2) should read P"$$$$$$$$9V.99"
> % The program displayed the menu.  We entered code 5 again.
> TYPE SELECTION
5
> % This corresponds to the currency code we input at the keyboard.
> CONTINUE

Breakpoint trap at :EXCHANGE:77 action="LIST 77"
77C             CALL CONVERSION(INDX, RATES(INDX));
> % We responded Yes to the exchange direction question.  (US$ to SF)
> TYPE INDX
5
> % INDX points to exchange rate for SF in first half of table.
> TYPE DAILY_RATES.RATES(5)
1.772500
> % This rate should also appear as RATE in the CONVERSION routine.
> CONTINUE

Breakpoint trap at :CONVERSION:13 action="LIST 13"
13C             IF CODE < 8 THEN DO;
> % CODE should equal 5 and correspond to INDX in the calling routine.
> TYPE CODE
5
> CONTINUE

Breakpoint trap at :CONVERSION:16 action="LIST 16"
16C                             FVAL = DVAL * RATE;
> % The program asked for the US dollar amount; we entered 250.
> % DVAL holds the number of American dollars to be converted.
> TYPE DVAL
       $2.50"
```

Figure 3-5. The Audit File AUDIT.YEN. (continued)

Examples

```
> CL ↓
NONUNIQUE COMMAND CL
> CLI "FILESTATUS/AS/S" ↓
        .
        .
        .
        .
> cli ↓
AOS CLI     REV 3.40     21-OCT-81     9:11:45
)       .
        .
        .
) BYE ↓
AOS CLI     TERMINATING  21-OCT-81     9:34:19
>
```

```
> % We entered the value 560 for the number of yen.
> TYPE FVAL
560.00
> TYPE RATE
0.004471
> % This value corresponds to the rate displayed earlier.
> TYPE 560. * .004471
2.50376
> CONTINUE

Breakpoint trap at :EXCHANGE:64 action="LIST 64"
64C                     CALL CHECK(SELECTION);
> % The result was $2.00.  It should have been $2.50.
> % The errors we've already found explain these problems.
> % We again entered currency code 8.
> TYPE SELECTION
8
> CONTINUE

Breakpoint trap at :EXCHANGE:77 action="LIST 77"
77C             CALL CONVERSION(INDX, RATES(INDX));
> % We responded Yes to the exchange direction question. (US$ to yen)
> TYPE INDX
8
> % The index is correct.
> % It points to the entry for yen in the first half of the table.
> TYPE DAILY_RATES.RATES(8)
225.900000
> CONTINUE

Breakpoint trap at :CONVERSION:13 action="LIST 13"
13C             IF CODE < 8 THEN DO;
> TYPE CODE
8
> % Change statement 13 in :CONVERSION.  It should read:
> %     IF CODE < 9 THEN DO;
> CONTINUE

Breakpoint trap at :CONVERSION:24 action="LIST 24"
24C             PUT FILE(SCREEN) SKIP LIST("ENTER ",CURRENCY(CODE-8),":  ");
> TYPE CODE-8
0
> CONTINUE AT 14

Breakpoint trap at :CONVERSION:16 action="LIST 16"
16C                             FVAL = DVAL * RATE;
> TYPE DVAL
"       $2.50"
> TYPE RATE
225.900000
> TYPE 250. * 225.90
56475.
```

Figure 3-5. The Audit File AUDIT.YEN. (continued)

NOT A BREAKPOINT locator

There is no breakpoint at the specified location.

UNDEFINED SYMBOL symbol

The specified locator does not exist within the working environment. Check your spelling or the environment.

Examples

```
> CLEAR, *5 ↓
Cleared at :.MAIN:10

> ENVIRONMENT :FULL; CLE 20 ↓
:FULL
Cleared at :FULL:20

> CLEAR :EXCHANGE:62 ↓
Cleared at :EXCHANGE:62

> cle 45 ↓

NOT A BREAKPOINT 45

> clear 27 ↓

NOT A BREAKPOINT 27

> Clear @all ↓
Cleared at :EXCHANGE:34
Cleared at :EXCHANGE:50
Cleared at :EXCHANGE:CHECK:87
Cleared at :CONVERSION:13
Cleared at :CONVERSION:16
Cleared at :EXCHANGE:42:43
Cleared at :EXCHANGE:71 count=4
```

Chapter 4
The SWAT Commands

This chapter describes all of the commands you can use in a SWAT debugging session. We have arranged them alphabetically to help you locate each command quickly.

Remember that while in a debugging session you can obtain information about SWAT commands by using the HELP command. The debugger displays a brief description about the selected command, its format and functions. Table 4-1 lists all the SWAT commands and their functions.

Table 4-1. SWAT Commands

Command	Use
AUDIT	Reports or sets the audit status.
BREAKPOINT	Displays or sets a breakpoint.
BYE	Terminates the SWAT debugger.
CLEAR	Clears a breakpoint.
CLI	Performs a CLI function.
CONTINUE	Initiates or resumes program execution.
DEBUG	Calls the AOS or AOS/VS user debugger.
DESCRIBE	Displays information about a program symbol.
DIRECTORY	Displays or sets the current directory.
ENVIRONMENT	Displays or sets the working environment.
EXECUTE	Executes a file of SWAT commands.
HELP	Displays information about SWAT topics.
LIST	Displays one or more lines from the program source file.
MESSAGE	Displays an error message for an error code.
PREFIX	Displays or sets the SWAT command prompt.
PROMPT	Displays or sets the null command response.
SEARCHLIST	Displays or sets the search list.
SET	Assigns a value to a variable.
TYPE	Displays the value of an expression.
WALKBACK	Displays the current location and calling locations.

Using Upper- and Lowercase

You can enter a SWAT command or keyword in any combination of uppercase or lowercase characters. The entries CLEAR, clear, and Clear are all equivalent.

When referring to a program symbol such as a variable, an array, a label, or a structure, however, you must follow the case-usage rules of your programming language. If the language is insensitive to character case (such as FORTRAN 77), you can use any combination of upper- and lowercase characters. But for case-sensitive languages such as PL/I, you must enter the symbol *exactly* as it appears in the source program. In this case, the SWAT debugger distinguishes between NAME, Name, and name.

The SWAT Prompt

When the debugger is ready to accept a command, it displays the prompt

>

followed by a space. You can change the prompt character by calling the PREFIX command. This command takes a string argument in which you specify the new prompt character(s).

Enter your command line, terminating it with a NEW LINE. If you make a syntax or format error in a command line, an error message appears. See Appendix B for a complete list of SWAT error messages. Following each command description is a list of errors related to the command.

Delimiting Commands and Arguments

To enter more than one command on a line, separate the commands with a semicolon. For example:

```
> ENVIRONMENT @BREAK; LIST 1 10 ↵
```

You can continue a command line over more than one typed line on the terminal. If you simply continue typing, your command line wraps around to the next line. You can also use an ampersand (&) to end an incomplete line, then continue it on the next line. In either case, the debugger acknowledges that you are continuing the command by preceding the the prompt on the next line with an ampersand. For example:

```
> AUDIT ON;BREAKPOINT 23,34,76,81,112;CONTINUE;& ↵
&>ENVIRONMENT;CLI "TIME";EXECUTE "DBUG" ↵
```

The complete command line cannot exceed 511 characters.

You can use a comma in place of a tab or space to separate arguments.

```
> CLEAR 53,76,104 ↵
```

is equivalent to

```
> CLEAR 53 76 104 ↵
```

At times, you may need to use a comma to avoid ambiguity or to separate complex arguments. Usually a comma must precede expressions that do not begin with either a digit or a name. The comma prevents the debugger from interpreting the argument(s) as an incomplete expression. For example:

```
TYPE , (A + B) * C
```

or

```
LIST , *5
```

When providing a series of expression arguments, you must use a comma to distinguish one from another.

```
TYPE A, -B * C, (D + E) * F
```

*

The Null Command

If you press just the NEW LINE key following a SWAT prompt, the debugger responds according to the current null command response setting. By default, the SWAT debugger performs a LIST @BREAK command when you enter a null command. To change the null command response setting, use the PROMPT command. You can specify one or more debugger operations to be performed in response to a null command.

Abbreviating SWAT Commands

You can abbreviate a SWAT command to its minimum uniqueness. For example, you can abbreviate the DESCRIBE command to DES, which distinguishes it from the DEBUG and DIRECTORY commands. If you enter only D or DE, however, the SWAT debugger cannot tell which of these commands you want to use.

You must spell out the CLI and SET commands completely to distinguish them from all other SWAT commands.

Locators

A locator identifies a program statement. The locator can be the statement's line number or label. If you specify a numeric label as a locator, precede it with an asterisk (*) to distinguish it from a line number. For example:

PROG1	label locator
250	line number locator
*250	numeric label locator

If the locator points to a line number outside the current program module, or to a label outside the working environment, you must fully qualify the locator by preceding it with an environment specifier. For example

:EXCHANGE:LOOP	label locator
:CONVERSION:24	line number locator
:.MAIN:*5	numeric label locator

Specifying Environments

Environment specifiers identify a specific procedure block. You use them with the ENVIRONMENT command to set the working environment, and with program statement locators to fully qualify them.

The general format for an environment specifier is

[:external-procedure] [[:] internal-procedure[:internal-procedure]...]

An external procedure is a block that is not contained within any other block. In other words, it is located directly below the root environment. For this reason, an external procedure identifier always begins with a colon. Do not begin an environment specifier with a colon unless the first element is an external procedure.

An internal procedure is an embedded block. Use a colon to separate an internal procedure identifier from an external procedure identifier.

An environment specifier can contain more than one internal procedure specifier, depending on the depth at which the target environment is embedded. Separate each internal procedure identifier from the previous one with a colon. Do not precede the first internal procedure identifier with a colon unless it follows another procedure identifier.

An external or internal procedure identifier is usually the block's label. The AOS SWAT debugger and SWAT16 identify an unnamed block by using the line number where the block begins.

The caret (^) symbol represents the next higher environment. You can use one or more carets alone, or as a prefix to one or more internal procedure identifiers.

There are two SWAT keywords that you can use to specify an environment. The @BREAK keyword identifies the block immediately containing the statement where execution is currently trapped. The @MAIN keyword represents the main program module.

The identifying statement of an embedded procedure block technically resides within the enclosing environment. To examine the program state when a block becomes active, set a breakpoint at an executable statement within the block. You cannot refer to symbols within a block unless execution is trapped within that block's environment.

Placing Comments in the Audit File

If you set up an audit file through the /AUDIT switch or the AUDIT command, you can keep a copy of your debugging session dialog. The debugger appends the dialog to the current audit file. The audit file is a useful tool for examining what occurred during a debugging session. To make this record more understandable, you can insert comments into the audit file as you debug the program.

To make a comment, simply type a percent sign (%) in the first space after the SWAT prompt, then enter your comment. The debugger ignores everything to the right of the percent sign. You can describe what you are doing, and document your debugging activity. Just remember to begin the comment with the percent sign, or the SWAT software will try to interpret the comment as a command.

Editing the Command Line

If you're working at a video display terminal, you can use various control characters to edit a command line. You are probably familiar with many of the control characters from your experience with one of Data General's text editors. Refer to your text editor manual and CLI User's Manual for details. Table 4-2 summarizes the control keys and functions you can use in a SWAT debugging session. Most of these control characters and keys do not apply to a hard-copy terminal.

Table 4-2. Control Characters and Keys for Editing

Key(s)	Use
CTRL-A	Duplicates the previous entry, or moves the cursor to the end of the current command line.
CTRL-B	Moves the cursor to the end of the previous word.
CTRL-E	Inserts the characters entered after the cursor position. (CTRL-E again closes insert mode.)
CTRL-F	Moves the cursor to the beginning of the next word.
CTRL-H	Moves the cursor to the first position on the line.
CTRL-I	Moves the cursor right to the next tab stop.
CTRL-K	Erases everything to the right of the cursor.
CTRL-L	Ends the command line and clears the screen, placing the cursor at the upper left position on the screen.
CTRL-O	Halts display at the terminal without stopping the output operation. (You can resume display of the output by typing CTRL-O again.)
CTRL-Q	Resumes display at the terminal (after CTRL-S).
CTRL-S	Halts display at the terminal until you type CTRL-Q.
CTRL-U	Clears the current line and places the cursor at the first position.
CTRL-X	Moves the cursor one position to the right.
CTRL-Y	Moves the cursor one position to the left.
DEL	Deletes one character to the left of the cursor, moving the cursor to that position, and shifting the rest of the line one position to the left.
ERASE EOL	Erases everything to the right of the cursor.
ERASE PAGE	Ends the command line and clears the screen, placing the cursor at the upper left position on the screen.
TAB	Moves the cursor right to the next tab stop.
→	Moves the cursor one position to the right.
←	Moves the cursor one position to the left.

Examples

```
) XEQ SWAT/AUDIT EXCHANGE ⤶
SWAT REVISION 02.00 ON 10/13/81 AT 14:23:18
PROGRAM — :UDD:TOM:EXCHANGE
> AUDIT ⤶
ON
.
.
> AUDIT ON "EXCHANGE.LOG" ⤶
.
.
> AU OFF ⤶
.
.
> audit on ⤶
```

The /AUDIT switch without a filename opens the audit file EXCHANGE.AU as the session begins. The AUDIT command confirms that auditing is on. Later in the session, a new audit file (EXCHANGE.LOG) is opened, thereby closing EXCHANGE.AU. Auditing is then turned off, closing EXCHANGE.LOG. The final AUDIT command turns auditing on, reopening the default audit file, EXCHANGE.AU.

Examples

```
) XEQ SWAT/AUDIT EXCHANGE ↵
SWAT REVISION 02.00 ON 10/13/81 AT 14:23:18
PROGRAM — :UDD:TOM:EXCHANGE
> AUDIT ↵
ON
.
.
> AUDIT ON "EXCHANGE.LOG" ↵
.
.
> AU OFF ↵
.
.
> audit on ↵
```

The /AUDIT switch without a filename opens the audit file EXCHANGE.AU as the session begins. The AUDIT command confirms that auditing is on. Later in the session, a new audit file (EXCHANGE.LOG) is opened, thereby closing EXCHANGE.AU. Auditing is then turned off, closing EXCHANGE.LOG. The final AUDIT command turns auditing on, reopening the default audit file, EXCHANGE.AU.

Editing the Command Line

If you're working at a video display terminal, you can use various control characters to edit a command line. You are probably familiar with many of the control characters from your experience with one of Data General's text editors. Refer to your text editor manual and CLI User's Manual for details. Table 4-2 summarizes the control keys and functions you can use in a SWAT debugging session. Most of these control characters and keys do not apply to a hard-copy terminal.

Table 4-2. Control Characters and Keys for Editing

Key(s)	Use
CTRL-A	Duplicates the previous entry, or moves the cursor to the end of the current command line.
CTRL-B	Moves the cursor to the end of the previous word.
CTRL-E	Inserts the characters entered after the cursor position. (CTRL-E again closes insert mode.)
CTRL-F	Moves the cursor to the beginning of the next word.
CTRL-H	Moves the cursor to the first position on the line.
CTRL-I	Moves the cursor right to the next tab stop.
CTRL-K	Erases everything to the right of the cursor.
CTRL-L	Ends the command line and clears the screen, placing the cursor at the upper left position on the screen.
CTRL-O	Halts display at the terminal without stopping the output operation. (You can resume display of the output by typing CTRL-O again.)
CTRL-Q	Resumes display at the terminal (after CTRL-S).
CTRL-S	Halts display at the terminal until you type CTRL-Q.
CTRL-U	Clears the current line and places the cursor at the first position.
CTRL-X	Moves the cursor one position to the right.
CTRL-Y	Moves the cursor one position to the left.
DEL	Deletes one character to the left of the cursor, moving the cursor to that position, and shifting the rest of the line one position to the left.
ERASE EOL	Erases everything to the right of the cursor.
ERASE PAGE	Ends the command line and clears the screen, placing the cursor at the upper left position on the screen.
TAB	Moves the cursor right to the next tab stop.
→	Moves the cursor one position to the right.
←	Moves the cursor one position to the left.

Errors

BREAKPOINT ALREADY SET AT locator

The specified line already has a breakpoint.

CANNOT SET BREAKPOINT AT locator

The specified line is a blank line, a comment line, a nonexecutable statement, a continuation line, or a nonexistent line.

COMMAND ENDS ILLEGALLY

You did not end the action string with a quotation mark or apostrophe.

ILLEGAL DATA TYPE symbol

Be sure to enclose the action string with quotation marks or apostrophes. Do not use these characters to set off a proceed count. This error also occurs if a symbol argument is not a label.

INVALID LOCATOR locator

The specified locator does not point to a line number or label in the working environment.

INVALID NUMBER number

The specified breakpoint count is not an integer value in the range 1 through 32767.

LINE NUMBER OUT OF RANGE

The specified line number does not exist within the current program module.

NONUNIQUE COMMAND command

Your command abbreviation is not unique. The shortest acceptable abbreviation for BREAKPOINT is BR.

TOO MANY BREAKPOINTS *

You can set a maximum of 20 breakpoints throughout the program. Your request exceeds that number.

UNDEFINED SYMBOL symbol

The specified locator does not exist within the working environment.

The debugger interprets the command

```
LIST 5 *20
```

as LIST 5*20 (i.e., LIST 100). A comma, however, separates the arguments so that the debugger lists the statements from line 5 to label 20.

```
LIST 5, *20
```

To list only the label, you must separate the command from the argument with a comma:

```
LIST, *20
```

The Null Command

If you press just the NEW LINE key following a SWAT prompt, the debugger responds according to the current null command response setting. By default, the SWAT debugger performs a LIST @BREAK command when you enter a null command. To change the null command response setting, use the PROMPT command. You can specify one or more debugger operations to be performed in response to a null command.

Abbreviating SWAT Commands

You can abbreviate a SWAT command to its minimum uniqueness. For example, you can abbreviate the DESCRIBE command to DES, which distinguishes it from the DEBUG and DIRECTORY commands. If you enter only D or DE, however, the SWAT debugger cannot tell which of these commands you want to use.

You must spell out the CLI and SET commands completely to distinguish them from all other SWAT commands.

Locators

A locator identifies a program statement. The locator can be the statement's line number or label. If you specify a numeric label as a locator, precede it with an asterisk (*) to distinguish it from a line number. For example:

PROG1	label locator
250	line number locator
*250	numeric label locator

If the locator points to a line number outside the current program module, or to a label outside the working environment, you must fully qualify the locator by preceding it with an environment specifier. For example

:EXCHANGE:LOOP	label locator
:CONVERSION:24	line number locator
:.MAIN:*5	numeric label locator

BYE

Terminates the SWAT debugger.

Format

BYE

Description

Use the BYE command to end a debugging session. The SWAT debugger terminates your program before ending the session. (If your program runs to completion following a CONTINUE command, the debugger terminates automatically, returning you to the calling process.)

When the debugging session ends, this message appears:

SWAT TERMINATED

Errors

COMMAND DOES NOT ACCEPT ARGUMENT(S)

There is no valid argument to this command. Enter only BYE (or its abbreviation).

NONUNIQUE COMMAND command

Your command abbreviation is not unique. The shortest acceptable abbreviation for BYE is BY.

Examples

> B ↵

NONUNIQUE COMMAND B

> BYE ↵

SWAT TERMINATED

Chapter 4
The SWAT Commands

This chapter describes all of the commands you can use in a SWAT debugging session. We have arranged them alphabetically to help you locate each command quickly.

Remember that while in a debugging session you can obtain information about SWAT commands by using the HELP command. The debugger displays a brief description about the selected command, its format and functions. Table 4-1 lists all the SWAT commands and their functions.

Table 4-1. SWAT Commands

Command	Use
AUDIT	Reports or sets the audit status.
BREAKPOINT	Displays or sets a breakpoint.
BYE	Terminates the SWAT debugger.
CLEAR	Clears a breakpoint.
CLI	Performs a CLI function.
CONTINUE	Initiates or resumes program execution.
DEBUG	Calls the AOS or AOS/VS user debugger.
DESCRIBE	Displays information about a program symbol.
DIRECTORY	Displays or sets the current directory.
ENVIRONMENT	Displays or sets the working environment.
EXECUTE	Executes a file of SWAT commands.
HELP	Displays information about SWAT topics.
LIST	Displays one or more lines from the program source file.
MESSAGE	Displays an error message for an error code.
PREFIX	Displays or sets the SWAT command prompt.
PROMPT	Displays or sets the null command response.
SEARCHLIST	Displays or sets the search list.
SET	Assigns a value to a variable.
TYPE	Displays the value of an expression.
WALKBACK	Displays the current location and calling locations.

NOT A BREAKPOINT locator

There is no breakpoint at the specified location.

UNDEFINED SYMBOL symbol

The specified locator does not exist within the working environment. Check your spelling or the environment.

Examples

```
> CLEAR, *5 ↓
Cleared at :.MAIN:10

> ENVIRONMENT :FULL; CLE 20 ↓
:FULL
Cleared at :FULL:20

> CLEAR :EXCHANGE:62 ↓
Cleared at :EXCHANGE:62

> cle 45 ↓

NOT A BREAKPOINT 45

> clear 27 ↓

NOT A BREAKPOINT 27

> Clear @all ↓
Cleared at :EXCHANGE:34
Cleared at :EXCHANGE:50
Cleared at :EXCHANGE:CHECK:87
Cleared at :CONVERSION:13
Cleared at :CONVERSION:16
Cleared at :EXCHANGE:42:43
Cleared at :EXCHANGE:71 count=4
```

CLI

Performs a CLI function.

Format

CLI *["string"]*

where:

string is a character string of one or more CLI commands.

Description

Use this command to perform a CLI function within the SWAT session.

If you type CLI with no argument, the SWAT software creates a CLI process, and passes control to it. You remain in the CLI until it terminates.

To perform one or more CLI functions in a debugging session, follow the SWAT CLI command with a command string enclosed in quotation marks or single apostrophes. Use the ASCII notation <042> or <047> respectively, for an embedded quotation mark or apostrophe. If the string contains more than one command, use a semicolon to separate one command from another.) The debugger passes the string to the CLI process for execution.

This command has no unique abbreviation; you must use its full name.

IMPORTANT: You can use this command only if you grant the SWAT debugger exclusive use of the terminal through the /CONSOLE or /NOCONSOLE switch.

If your user profile does not grant the privilege to create unlimited sons, you may also need to use the /SONS switch to ensure that SWAT can create a minimum of two son processes.

Errors

COMMAND ENDS ILLEGALLY

You did not end the string argument with a quotation mark or apostrophe.

DEVICE ALREADY IN USE

(System error): The SWAT debugger does not have exclusive use of this terminal. Use the /CONSOLE or /NOCONSOLE switch.

INVALID EXTRA ARGUMENTS ON COMMAND

You can provide only one string argument. The string can contain more than one CLI command, provided that you separate each command with a semicolon.

NONUNIQUE COMMAND command

There is no acceptable abbreviation for this command; enter CLI.

TOO MANY SUBORDINATE PROCESSES

(System error): The SWAT debugger cannot create the CLI process because it has already created the maximum number of sons allowed it.

Examples

```
> CL ↵
NONUNIQUE COMMAND CL
> CLI "FILESTATUS/AS/S" ↵
        .
        .
        .
        .
> cli ↵
AOS CLI      REV 3.40      21-OCT-81      9:11:45
)       .
        .
        .
) BYE ↵
AOS CLI      TERMINATING   21-OCT-81      9:34:19
>
```

```
> CONTINUE

Breakpoint trap at :CONVERSION:26 action="LIST 26"
26C             DVAL = FVAL * RATE;
> % We entered 250 Swiss francs for the exchange.
> TYPE FVAL
250.00
> % This is the number of Swiss francs we entered.
> TYPE RATE
0.556600
> % This is the same rate we observed earlier.
> TYPE 250.00 * .556600
139.15
> % The program should output $139.15 as its result.
> CONTINUE

Breakpoint trap at :EXCHANGE:64 action="LIST 64"
64C                     CALL CHECK(SELECTION);
> % The output value was $139.00, but should have been $139.15.
> % Change the output statement, line 29 of the :CONVERSION module.
> % -- The second F(11,2) should read P"$$$$$$$$9V.99"
> % The program displayed the menu.  We entered code 5 again.
> TYPE SELECTION
5
> % This corresponds to the currency code we input at the keyboard.
> CONTINUE

Breakpoint trap at :EXCHANGE:77 action="LIST 77"
77C             CALL CONVERSION(INDX, RATES(INDX));
> % We responded Yes to the exchange direction question.  (US$ to SF)
> TYPE INDX
5
> % INDX points to exchange rate for SF in first half of table.
> TYPE DAILY_RATES.RATES(5)
1.772500
> % This rate should also appear as RATE in the CONVERSION routine.
> CONTINUE

Breakpoint trap at :CONVERSION:13 action="LIST 13"
13C             IF CODE < 8 THEN DO;
> % CODE should equal 5 and correspond to INDX in the calling routine.
> TYPE CODE
5
> CONTINUE

Breakpoint trap at :CONVERSION:16 action="LIST 16"
16C                             FVAL = DVAL * RATE;
> % The program asked for the US dollar amount; we entered 250.
> % DVAL holds the number of American dollars to be converted.
> TYPE DVAL
-       $2.50"
```

Figure 3-5. The Audit File AUDIT.YEN. (continued)

Errors

CANNOT CONTINUE AT locator

The specified locator does not point to an executable statement.

INVALID EXTRA ARGUMENTS ON COMMAND

You supplied unnecessary arguments or omitted the equal sign after the COUNT keyword.

INVALID NUMBER number

The specified count number is not an unsigned integer value in the range 1 through 32767.

NONUNIQUE COMMAND command

Your command abbreviation is not unique. The shortest acceptable abbreviation for CONTINUE is CO.

NOT AT A BREAKPOINT

Either program execution has not yet begun, or you used the COUNT or AT option inappropriately. You cannot use the COUNT option if you've cleared the current breakpoint, nor can you use the AT option with the first CONTINUE command.

NOT IN BREAKPOINT ENVIRONMENT

The specified locator points to a statement outside the breakpoint environment. You can continue execution only within the breakpoint environment.

SOME DATA MAY NOT BE CURRENT

This is a warning message. One or more accumulators held data values when the trap occurred.

NOTE: If a trap occurs at a breakpoint that has an attached action string, additional errors may result from the commands contained in the string.

Examples

```
> CONTINUE ↵

Breakpoint trap at :EXCHANGE:34

> co ↵

Breakpoint trap at :EXCHANGE:50 action= "LIST 50"
50C     LOOP:       DO WHILE("1"B);

> CON COUNT=4 ↵

Breakpoint trap at :EXCHANGE:64

> C ↵

NONUNIQUE COMMAND C

> CONTINUE AT 50 ↵

Breakpoint trap at :EXCHANGE:64
```

```
>>LIST @BREAK
 54                     PUT FILE(SCREEN) SKIP(2) EDIT
 55                         ("1","Belgian francs","2","W. German marks","3",
 56                         "Dutch guilders","4","Italian lire","5","Swiss francs",
 57                         "6","Canadian dollars","7","Saudi riyals","8",
 58                         "Japanese yen","(Type 0 to end the program)",
 59                         "Enter the currency code:  ")
 60                         (8(X(5),A(1),X(3),A(16),SKIP),SKIP,A(27),SKIP,A(26));
 61
 62                     GET FILE(KEYBOARD) LIST(SELECTION);
 63
 64C                    CALL CHECK(SELECTION);
 65
 66      /* Request type of exchange */
 67
 68             PUT FILE(SCREEN) SKIP LIST("Do you want to convert US$ into"
 69                     !!" Foreign currency?");
 70             PUT FILE(SCREEN) SKIP LIST("Enter 'Y' or 'N':  ");
 71             GET FILE(KEYBOARD) LIST(RESPONSE);
 72
 73             IF RESPONSE = "Y" THEN INDX = SELECTION;
 74             ELSE IF RESPONSE = "y" THEN INDX = SELECTION;
> PROMPT
"LIST @BREAK"
> PROMPT "CONTINUE"
>

Breakpoint trap at :CONVERSION:16
> BYE

SWAT TERMINATED
```

Figure 3-4. The Audit File EXCHANGE.AU. (concluded)

DEBUG

Calls the AOS or AOS/VS user debugger.

Format

DEBUG

Description

Use this command to call the AOS or AOS/VS user debugger within a SWAT session. The DEBUG command assumes that the program you are debugging has a terminal assigned to it. If the SWAT invocation command included the /NOCONSOLE switch, do not use the DEBUG command; a fatal error will occur.

To grant the user debugger write access to the program, include the /DEBUG switch in the SWAT invocation command. (If you use this switch, your debugging session begins in the user debugger.)

To exit the user debugger, enter the appropriate command:

P ⤶	Exits the AOS user debugger
ESC R	Exits the AOS/VS user debugger

When you exit the user debugger, the SWAT prompt appears.

NOTE: Auditing does not record any I/O performed by a program called from the SWAT debugger. Your interactions with the user debugger will not appear in an opened audit file.

Errors

COMMAND DOES NOT ACCEPT ARGUMENT(S)

There is no valid argument to this command. Enter only DEBUG (or its abbreviation).

NONUNIQUE COMMAND command

Your command abbreviation is not unique. The shortest acceptable abbreviation for DEBUG is DEB.

Examples

```
) X SWAT EXCHANGE ⤶
SWAT REVISION 02.00 ON 10/05/81 AT 12:57:10
PROGRAM -- :UDD:TOM:EXCHANGE
> DEBUG ⤶
AOS USER DEBUGGER, REV 3.40
WRITE ACCESS DENIED
+
.
.
.
+P ⤶
>
```

```
> TYPE SELECTION + 8
10
> TYPE SELECTION + INDX
2
> TYPE INDX - SELECTION
-2
> %Integer division produces integer result.
> TYPE 34/2
17
> TYPE 35/2
17
> TYPE 999999999 * 10
1410065398
>% Overflow causes a meaningless result.
> TYPE 999999999. * 10.
9999999990.
> TYPE 403R8
259
> TYPE DAILY_RATES.RATES(6)
1.186500
> SET DAILY_RATES.RATES(6) = DAILY_RATES.RATES(14)
Old value: 1.186500
New Value: 0.843400
> SET DAILY_RATES.RATES(6) = "abcd"
Old value: 0.843400

ILLEGAL DATA TYPE  abcd
> DESCRIBE SELECTION
SELECTION automatic at +8 words, fixed binary (1 word)
> DESCRIBE SCREEN
SCREEN constant (at 000447R8) file constant (1 word)
> DESCRIBE DAILY_RATES
DAILY_RATES automatic at +20 bytes, structure (102 byte)
  2 RATE_DATE at +0 bytes, character (6 byte)
  2 RATES at +6 bytes, fixed decimal (10.6) array (1:16)
> DESCRIBE :CONVERSION
CONVERSION line 3 to line 33
> HELP

topics are:

        @ALL            @BIT            @BREAK          @CHARACTER
        @FLOAT          @INTEGER        @MAIN           @POINTER
        ACTION          AT              AUDIT           BREAKPOINT
        BYE             CLEAR           CLI             COMMANDS
        COMMENTS        CONTINUE        COUNT           DEBUG
        DESCRIBE        DIRECTORY       ENVIRONMENT     EXECUTE
        EXPRESSIONS     HELP            KEYWORDS        LIST
        LOCATORS        MESSAGE         OFF             ON
        OPERATORS       PREFIX          PROMPT          SEARCHLIST
        SET             SPECIFIERS      TYPE            UNIQUENESS
        WALKBACK
```

Figure 3-4. The Audit File EXCHANGE.AU. (continued)

DESCRIBE

Displays information about a program symbol.

Format

DESCRIBE symbol ...

where:

symbol is a user-defined program symbol.

Description

Use the DESCRIBE command to obtain information about a user-defined program symbol such as a variable, label, constant, or structure. To obtain information about a symbol in another environment, precede the symbol name with the appropriate environment specifier.

To display information about a program symbol that is known within the working environment, simply supply that symbol reference as an argument to the command. If the symbol is not defined in the working environment, you must fully qualify the reference.

The terminology used in the description depends on the programming language you are using. The format of the description differs slightly between the AOS and AOS/VS versions.

Errors

COMMAND REQUIRES ARGUMENT(S)

You must provide at least one symbol as an argument to this command.

UNDEFINED SYMBOL symbol

The specified symbol does not exist.

Examples

```
> DESCRIBE :CONVERSION ↵
CONVERSION line 3 to line 33

> desc SELECTION ↵
SELECTION automatic at +8 words, fixed binary (1 word)

> describe DAILY_RATES ↵
DAILY_RATES automatic at +20 bytes, structure (102 byte)
     2 RATE_DATE at +0 bytes, character (6 byte)
     2 RATES at +6 bytes, fixed decimal (10,6) array (1:16)

> desc SKREEN ↵

UNDEFINED SYMBOL SKREEN
```

```
> LIST 20 28
 20              OPEN FILE(RATE_FILE) STREAM INPUT;
 21
 22      /* Read in today's exchange rates.  Compare date with system date. */
 23
 24C             GET FILE(RATE_FILE) LIST(DAILY_RATES);
 25
 26              ON ENDFILE(RATE_FILE)
 27              BEGIN;
 28                      PUT FILE(SCREEN) SKIP LIST("NO RATES AVAILABLE.");
> LIST @BREAK
 14                      2 RATES(16) FIXED DECIMAL(10,6);
 15              DECLARE RESPONSE CHARACTER(1);
 16              DECLARE INDX FIXED BINARY;
 17
 18              OPEN FILE(SCREEN) STREAM OUTPUT PRINT TITLE("@OUTPUT");
 19              OPEN FILE(KEYBOARD) STREAM INPUT TITLE("@INPUT");
 20              OPEN FILE(RATE_FILE) STREAM INPUT;
 21
 22      /* Read in today's exchange rates.  Compare date with system date. */
 23
 24C             GET FILE(RATE_FILE) LIST(DAILY_RATES);
 25
 26              ON ENDFILE(RATE_FILE)
 27              BEGIN;
 28                      PUT FILE(SCREEN) SKIP LIST("NO RATES AVAILABLE.");
 29                      STOP;
 30              END; /*BEGIN Block for End of File condition */
 31
 32      /* Compare file's date with the system date. */
 33
 34B             IF RATE_DATE ^= DATE() THEN DO;
> CONTINUE

Breakpoint trap at :EXCHANGE:34
```

Figure 3-4. The Audit File EXCHANGE.AU. (continued)

DIRECTORY

Displays or sets the current directory.

Format

DIRECTORY *["pathname"]*

where:

pathname identifies the directory to become the working directory.

Description

Use this command to set or display the working directory for the SWAT debugging session. Initially, the working directory is the one from which you called the SWAT debugger. (This directory applies only to the current debugging session. When you terminate the SWAT process, you return to the directory in which you invoked the SWAT debugger.)

To display the current working directory, enter the command without an argument.

To set the working directory, follow the DIRECTORY command with a pathname. Enclose the pathname in quotation marks or apostrophes.

A confirmation message appears after you set the working directory.

This command does not affect your program's directory.

Errors

COMMAND ENDS ILLEGALLY

You probably omitted the final quotation mark or apostrophe to set off the string argument.

INVALID EXTRA ARGUMENTS ON COMMAND

You can specify only one directory as the working directory. Use the SEARCHLIST command if you need access to other directories.

NONUNIQUE COMMAND command

Your command abbreviation is not unique. The shortest acceptable abbreviation for the DIRECTORY command is DI.

UNEXPECTED OPERATOR operator

You probably used a pathname prefix without enclosing the pathname in quotation marks or apostrophes.

Examples

```
> DIRECTORY ↵
:UDD:TOM

> dir "TEST_PROGRAMS" ↵
:UDD:TOM:TEST_PROGRAMS

> DIR "^" ↵
:UDD:TOM
```

ENVIRONMENT

Displays or sets the working environment.

Format

ENVIRONMENT [*@ALL* | *@BREAK* | *@MAIN* | *specifier*]

where:

@ALL	represents all the external environments that the SWAT debugger can examine.
@BREAK	represents the environment of the current breakpoint trap.
@MAIN	represents the main program module.
specifier	identifies a procedure block to become the working environment.

Description

If you type only ENVIRONMENT, the debugger displays the current working environment; i.e., the area of the program that you can immediately refer to.

If you use the @ALL keyword, the debugger displays the name of each program module that it can debug. This listing is helpful when only certain modules were compiled and linked with the /DEBUG switch.

You can set the working environment by supplying an argument to the command. If you use the keyword @BREAK, the SWAT debugger sets the working environment to that of the current breakpoint trap. The @BREAK keyword provides a convenient return to the breakpoint environment.

Use the @MAIN keyword to set the working environment to that of the main procedure block.

You can also use an environment specifier to identify an environment to which you want to relocate. Specify the procedure name of the block (or, for the AOS SWAT debugger and SWAT16, you can use the line number of an unnamed block). Refer to the section called "Specifying Environments" earlier in this chapter.

Whenever you use this command, the debugger displays the working environment. The working environment remains unchanged until either you explicitly reset it, or program execution traps in another environment.

Errors

ENVIRONMENT NOT FOUND specifier

The specifier does not identify an existing environment. To refer to a superior environment or an external environment, be sure to begin the specifier at the root environment, then work your way down. If the environment is embedded within the working environment, do not begin the specifier with a colon.

NONUNIQUE COMMAND command

Your command abbreviation is not unique. The shortest acceptable abbreviation for ENVIRONMENT is EN.

NOT AT A BREAKPOINT

You used the @BREAK keyword, which refers to the environment of the current breakpoint trap. Program execution, however, has not yet begun; you must first issue a CONTINUE command.

Examples

```
SWAT REVISION 02.00 ON 10/04/81 AT 14:45:16
PROGRAM – :UDD:TOM:EXCHANGE
> ENVIRONMENT ↵
:EXCHANGE

> ENV CHECK ↵
:EXCHANGE:CHECK

> ENV :CONVERSION ↵
:CONVERSION

> env :EXCHANGE:27 ↵
:EXCHANGE:27

> ENV @MAIN ↵
:EXCHANGE

> ENVIRONMENT ↵
:.MAIN

> ENV :PAYMENT ↵
:PAYMENT

> ENV @MAIN ↵
:.MAIN
```

EXECUTE

Executes a series of SWAT commands stored in a file.

Format

EXECUTE "filename"

where:

filename identifies a file that consists of one or more debugger commands.

Description

Use this command to execute SWAT commands stored in a file. With the EXECUTE command, you can employ user-built debugger utilities.

Name the command file by providing its filename enclosed in quotation marks. You cannot use a filename template.

The SWAT debugger executes the commands in the order they appear in the file. The commands must carry all necessary arguments; you cannot pass arguments at runtime.

Errors

COMMAND ENDS ILLEGALLY

The string argument does not end with a quotation mark or apostrophe.

COMMAND REQUIRES ARGUMENT(S)

You must provide an argument that is a filename or an expression that resolves to a filename.

FILE DOES NOT EXIST

(System error): The specified command file does not exist, or does not reside in a directory on the current search list.

FILENAME TOO LONG

(System error): The filename you provided (either explicitly or implicitly) contains an illegal number of characters.

NONUNIQUE COMMAND command

Your command abbreviation is not unique. The shortest acceptable abbreviation for EXECUTE is EX.

Examples

```
> E "DBUG" ↓
```

NONUNIQUE COMMAND E

```
> EXECUTE "DBUG" ↓
>> command_1
response_1
>> command_2
response_2
        .
        .
        .
>
```

HELP

Displays information about SWAT topics.

Format

HELP *[topic] ...*

where:

topic is an entry from the help menu.

Description

To get quick, on-line information when you're in a SWAT debugging session, simply type HELP. The SWAT debugger displays a menu of topics you can obtain specific information for.

For information about a particular topic, follow the HELP command with the name of the topic. You can abbreviate any help topic. If the abbreviation is not unique, the debugger displays the topics that match it. You can then select from these (or any other) topics.

You can append more than one topic to the HELP command.

Figure 4-1 illustrates a sample help menu. Later revisions of the SWAT debugger may list additional topics. Simply type HELP to display the current menu.

```
topics are:

      @ALL          @BIT          @BREAK        @CHARACTER
      @FLOAT        @INTEGER      @MAIN         @POINTER
      ACTION        AT            AUDIT         BREAKPOINT
      BYE           CLEAR         CLI           COMMANDS
      COMMENTS      CONTINUE      COUNT         DEBUG
      DESCRIBE      DIRECTORY     ENVIRONMENT   EXECUTE
      EXPRESSIONS   HELP          KEYWORDS      LIST
      LOCATORS      MESSAGE       OFF           ON
      OPERATORS     PREFIX        PROMPT        SEARCHLIST
      SET           SPECIFIERS    TYPE          UNIQUENESS
      WALKBACK
```

Figure 4-1. The Help Menu

Errors

UNKNOWN TOPIC topic

The specified topic is not listed on the help menu.

HELP (continued)

Examples

> HELP ↵

topics are:

@ALL	*@BIT*	*@BREAK*	*@CHARACTER*
@FLOAT	*@INTEGER*	*@MAIN*	*@POINTER*
ACTION	*AT*	*AUDIT*	*BREAKPOINT*
BYE	*CLEAR*	*CLI*	*COMMANDS*
COMMENTS	*CONTINUE*	*COUNT*	*DEBUG*
DESCRIBE	*DIRECTORY*	*ENVIRON-*	*EXECUTE*
EXPRESSIONS	*HELP*	*MENT*	*LIST*
LOCATORS	*MESSAGE*	*KEYWORDS*	*ON*
OPERATORS	*PREFIX*	*OFF*	*SEARCHLIST*
SET	*SPECIFIERS*	*PROMPT*	*UNIQUENESS*
WALKBACK		*TYPE*	

> help @all ↵

Keyword: *CLEAR @ALL*
ENVIRONMENT @ALL
LIST @ALL

CLEAR @ALL will clear all breakpoints. ENVIRONMENT @ALL will report all debugger external procedures. LIST @ALL will list all the source lines in the current environment.

> H DEB ↵

Command: DEBUG

Enter the user debugger. To return to SWAT, either type P newline (AOS), or escape R (AOS/VS).
Note: This command is usable only if the user program has a console, i.e. the /NOCONSOLE switch was not set on SWAT. Also, the user debugger will have write access only if the /DEBUG switch was set on SWAT.

> Help program ↵

UNKNOWN TOPIC program

LIST

Displays one or more lines from the program source file.

Format

LIST { @ALL | @BREAK | locator *[locator]* }

where:

@ALL — represents the entire contents of the current program module.

@BREAK — represents the statement where program execution is currently trapped, and a range of up to 10 lines to either side of that statement.

locator — is a line number or statement label.

Description

The LIST command requires access to the program source file that you compiled and linked to produce the program. If this file is not available, the SWAT debugger cannot list source code.

You can request the debugger to display a particular line by supplying a line locator to the command. To specify a range of lines, give two locators with the command. The first locator identifies where the display is to begin and the second locator specifies where it ends. The largest range you can display is 1 through 8191 (for AOS) or 32767 (for AOS/VS). If you specify a range of lines that exceeds the range of the current module, the SWAT debugger ignores your error and displays the available source code.

Fully qualify a locator if it refers to a line number outside of the current program module or to a label outside of the working environment. When specifying a range of lines, both locators must point to lines in the same program module.

Type LIST @ALL to display the entire source file for the current program module.

Use the @BREAK keyword to display a range of up to 10 lines on either side of the statement where the last breakpoint trap occurred. If a trap has not yet occurred, the first 21 lines of the main procedure appear. Unless you change the default null command response through the PROMPT command, the debugger performs the LIST @BREAK command when you enter a NEW LINE only.

When listing lines from the source program, the debugger appends the letter B to the line number of each statement that carries a breakpoint. The letter C, however, identifies the statement where the last breakpoint trap occurred.

Errors

CANNOT LIST LINE

The line numbers of the source file do not match those of the program file. The source file may have been altered since compilation.

COMMAND REQUIRES ARGUMENT(S)

You must provide at least one locator or keyword as an argument to this command.

FILE DOES NOT EXIST filename

(System error): The source file does not exist within the working directory or any directory on the current search list.

LIST (continued)

FILE NOT CURRENT filename

The source file may have been modified. If so, the LIST command cannot accurately display source lines.

ILLEGAL DATA TYPE symbol

You supplied a locator that cannot be used as a label.

LOCATORS MUST BE IN SAME ENVIRONMENT

You can display a range of lines within a single module only.

UNDEFINED SYMBOL symbol

The specified symbol does not exist within the working environment.

Examples

```
> LIST 1 15 ↓
1     EXCHANGE:
2           PROCEDURE;
3
4     /* This program calculates currency exchange values based on
5        daily international exchange rates. The operator selects
6        the type of exchange then enters the incoming amount. The
7        program performs the calculation and displays the result. */
8
9           DECLARE (SCREEN, KEYBOARD, RATE_FILE) FILE;
10          DECLARE CONVERSION ENTRY (FIXED BINARY, FIXED
              DECIMAL(10,6));
11          DECLARE SELECTION FIXED BINARY;
12          DECLARE 1 DAILY_RATES,
13                2 RATE_DATE CHARACTER(6),
14                2 RATES(16) FIXED DECIMAL(10,6);
15          DECLARE RESPONSE(1) CHARACTER;

> LIST CHECK 100 ↓
81    CHECK:
82          PROCEDURE(SEL);
83          DECLARE SEL FIXED BINARY;
84
85          IF SEL > 0 THEN
86                IF SEL < 9 THEN RETURN;
87          PUT FILE(SCREEN) SKIP LIST("Exchange program ended.");
88          STOP;
89    END;/* CHECK SUBROUTINE */
90
91    END;/* EXCHANGE */

> list LOOP ↓
50B   LOOP:     DO WHILE("1"B);
```

```
> ENVIRONMENT ↓
:EXCHANGE
> LIST :CONVERSION:16 ↓
16        FVAL = DVAL * RATE;

> list, *5 ↓
10     5      print *, "Enter principal amount:   $"

> list @BREAK ↓
12
13           print *, "Enter interest rate. % = "
14           read *, RATE
15B         RATE = RATE / 100
16
17           print *, "Enter the number of years: "
18           read *, IYEARS
19
20      C Call the PAYMENT subroutine, which calculates the monthly payment.
21
22C             call PAYMENT(AMOUNT, RATE, IYEARS, PAY)
23
24      C Does user want full schedule?
25
26           print *, "Do you want a full schedule? Enter YES or NO: "
27           read (*, '(A)') ANSWER
28
29B         if ( (ANSWER(1:1) .eq. 'Y') .or. (ANSWER(1:1) .eq. 'y') ) then
30
31B         call FULL(AMOUNT, RATE, IYEARS, PAY)
32
```

MESSAGE

Displays an error message for an error code.

Format

MESSAGE expression ...

where:

expression resolves to a positive integer value.

Description

Use this command to display the system error message for an error code. Follow the MESSAGE command with one or more expressions that resolve to a numeric value. The SWAT debugger looks up the error message that corresponds to the octal equivalent of the expression result. It then displays the octal value and message text.

This command is useful for interpreting error codes that your program receives as parameters, such as an error status variable assigned through the FORTRAN 77 IOSTAT option, or returned by the PL/I ONCODE function.

Errors

ILLEGAL DATA TYPE symbol

The expression must resolve to a positive integer value. The specified symbol is incompatible.

INVALID NUMBER number

The expression does not resolve to a positive integer value.

Examples

```
> MESSAGE 23R8 ↵
000023 DIRECTORY DOES NOT EXIST

> me IER ↵
000067 LINE TOO LONG

> M IER + 3
000072 NOT A DIRECTORY
```

PREFIX

Displays or sets the SWAT command prompt.

Format

PREFIX *["string"]*

where:

string is a character string to become the SWAT prompt.

Description

Use this command to set or display the SWAT command prompt, which the debugger displays when it is ready to accept a command. The default prompt symbol is a right angle bracket (>). The debugger always outputs a space following the prompt.

If you type this command without an argument, the debugger displays the current prompt within quotation marks.

To change the prompt, supply a character string enclosed in quotation marks or single apostrophes. Use the ASCII notation <042> or <047> for an embedded quotation mark or apostrophe, respectively. The prompt may contain up to 32 characters. If you provide a longer string, the debugger ignores the extra characters.

Errors

COMMAND ENDS ILLEGALLY

You did not end the argument with a quotation mark or apostrophe.

NONUNIQUE COMMAND command

Your command abbreviation is not unique. The shortest acceptable abbreviation for PREFIX is PRE.

Examples

```
> PREFIX ↵
">"

> PREFIX "What next?" ↵
What next? ENVIRONMENT ↵
:EXCHANGE
What next? PREFIX "?" ↵
?
```

PROMPT

Displays or sets the null command response.

Format

PROMPT *["string"]*

where:

string is a character string containing one or more SWAT commands.

Description

Use this command to set or display the current null command response. By default, the null command response is LIST @BREAK.

If you enter the PROMPT command with no argument, the debugger displays the current null command response within quotation marks.

To set the null command response, supply a command string. Use a semicolon to separate commands within the string. Begin and end the string with a quotation mark or apostrophe. Use the ASCII notation <042> or <047>, respectively, for an embedded quotation mark or apostrophe.

To clear the prompt setting completely, supply a null string as the argument.

The only limit to the length of the string argument is the 511-character maximum for any SWAT command.

Errors

COMMAND ENDS ILLEGALLY

You did not end the argument with a quotation mark or apostrophe.

NONUNIQUE COMMAND command

Your command abbreviation is not unique. The shortest acceptable abbreviation for PROMPT is PRO.

Examples

```
> PROMPT "ENVIRONMENT" ↵
> ↵
:CONVERSION
> pro ↵
"ENVIRONMENT"

> PRO "" ↵

> ↵
> PROMPT "ENVIRONMENT; CLI <042>TIME<042>;EXECUTE <042>DBUG<042>"
↵
> ↵
:CONVERSION
11:43:23
        .
        .
        .
>
```

SEARCHLIST

Displays or sets the search list.

Format

SEARCHLIST *["pathname"]* ...

where:

pathname identifies a directory for the search list.

Description

Use this command to display or set the current search list for the SWAT debugging session. This search list applies only for the debugging session. After the session ends, your search list is the same as it was when you invoked the SWAT debugger (unless you changed it via the CLI command).

To display the current search list, enter the command without an argument.

To set a new search list, supply one or more directories as arguments to the command. Be sure to enclose each pathname in quotation marks or apostrophes. You can use pathname prefixes. See the *CLI User's Manual* for information about pathname prefixes.

The pseudo-macro [!SEARCHLIST] is not an acceptable argument.

This command does not affect your program's search list.

Errors

COMMAND ENDS ILLEGALLY

You did not end the argument with a quotation mark or apostrophe.

FILE DOES NOT EXIST

(System error): The SWAT debugger cannot locate a specified file.

ILLEGAL FILENAME CHARACTER

(System error): One or more of the arguments contains an illegal character. Make sure that you enclose each filename in quotation marks or apostrophes. You cannot use the [!SEARCHLIST] pseudo-macro.

NONUNIQUE COMMAND command

Your command abbreviation is not unique. The shortest acceptable abbreviation for SEARCHLIST is SEA.

UNEXPECTED OPERATOR operator

You used a pathname prefix, but did not enclose the argument in quotation marks or apostrophes.

SEARCHLIST (continued)

Examples

```
> SEARCHLIST ⤸
:MACROS :BASIC

> SEAR ":UTIL" ":LINKS" ⤸
:UTIL :LINKS

> search ":UDD:TOM:TEST_PROGRAMS" ⤸
:UDD:TOM:TEST_PROGRAMS

> sea "^" ⤸
:UDD:TOM
```

SET

Assigns a value to a variable.

Format

SET variable = expression ...

where:

variable	is either a stack variable defined in the breakpoint environment, or a static variable.
expression	resolves to a value that has the same data type as the variable.

Description

Use this command to assign a value to a variable defined within the breakpoint environment, or to a static variable. You can use this command for variables of the following data types: bit, Boolean, character, complex, packed or unpacked decimal, entry, file, float, integer, label, picture, or pointer. (Not every programming language supports all of these types. Terminology may differ from one language to another.)

The expression you supply represents the value you want to assign to the variable. This argument must agree in data type with the receiving variable. The expression may comprise any proper combination of variables, numeric constants, array references, based references, fully qualified structure references, strings, mathematical operators, and subexpressions. The SWAT debugger evaluates the argument and assigns the result to the variable.

If the receiving variable is integer or float, the expression can be an integer or float variable, constant, or arithmetic expression.

If the variable's data type is file, entry, or label, the debugger interprets it as one or more pointer values, depending on its size. You can use the SET command to assign such a variable to another of the same data type.

If the variable is an arbitrary length item such as a decimal, character, bit, or picture variable, the debugger performs a simple assignment. It pads or truncates character and bit strings, as necessary. The SWAT debugger uses spaces to pad character variables. When assigning decimal or picture values, the receiving variable and the assignment variable must carry the same declared extents.

A bit expression can contain a logical operator such as & (logical AND) or ! (logical inclusive OR).

If the variable's data type is bit or character, you can assign a string constant to the variable. Enclose the string in quotation marks or single apostrophes. (To include an embedded quotation mark or apostrophe in a character string, use the ASCII notation <042> or <047>, respectively). A bit string can consist of the characters 0 and 1 only.

The SWAT debugger can handle strings of up to 2048 characters. If you use the SET command with a string that is longer, the debugger modifies only the first 2048 characters, and sends you a warning message.

If the variable is a fixed-length item, such as a complex value, the debugger makes an assignment from a variable of the same data type and precision as the receiving variable.

For Boolean (F77 logical) variables, a nonzero integer value represents .TRUE.; if equal to zero, the value signifies .FALSE..

SET (continued)

When you alter a variable's value, the debugger confirms the change with the message:

Old value: old-n
New value: new-n

Errors

COMMAND REQUIRES =

You must separate the variable and expression with an equal sign.

COMMAND REQUIRES ARGUMENT(S)

You must specify at least one variable and the value it is to receive.

ILLEGAL DATA TYPE symbol

You tried to set a value to a variable whose data type is not one the SWAT debugger will change.

ILLEGAL PARENTHESIZATION

The expression has mismatched parentheses or square brackets.

INCORRECT NUMBER OF SUBSCRIPTS array

You did not supply the correct number of subscripts for this array reference.

NONUNIQUE COMMAND command

There is no acceptable abbreviation for this command; enter SET.

NOT A BASED VARIABLE symbol

The expression refers to a variable as if it were a based variable, but the variable is declared with another data type.

NOT A MEMBER VARIABLE symbol

The expression refers to a variable as if it were a member of a structure, but the variable is declared with another data type.

NOT A STRUCTURED VARIABLE symbol

The expression refers to a variable as if it were a structure, but the variable is declared with another data type.

NOT A VARIABLE symbol

The expression refers to a constant as if it were a variable.

NOT AN ARRAY VARIABLE symbol

The expression refers to a variable as if it were an array, but the variable is declared with another data type.

NOT AT A BREAKPOINT

Program execution has not begun. You cannot use this command for a stack variable until a breakpoint trap occurs.

NOT IN BREAKPOINT ENVIRONMENT

You tried to set a value for a variable that is not recognized within the breakpoint environment.

STRING TOO LONG

The string argument is longer than 2048 characters. The extra characters are ignored.

SUBSCRIPT OUT OF RANGE array

The subscript for the specified array exceeds the array's declared bounds.

UNTERMINATED QUOTED STRING

You did not end the string argument with a quotation mark or apostrophe.

Examples

```
> SET SELECTION=4 ↵
Old value: 0
New value: 4

> SET SELECTION = SELECTION + 2 ↵
Old value: 4
New value: 6

> SET RESPONSE = "N" ↵
Old value: "Y"
New value: "N"
```

TYPE

Displays the value of an expression.

Format

TYPE expression [@BIT | @CHARACTER | @FLOAT | @INTEGER | @POINTER | @Rn] ...

where:

expression	is an expression containing one or more elements.
@BIT	displays the expression result as a bit string.
@CHARACTER	displays the expression result as a character string.
@FLOAT	displays the expression result as a floating-point value.
@INTEGER	displays the expression result as an integer value.
@POINTER	dislays the expression result as one or more pointer values.
@Rn	(AOS/VS only): displays the expression result with radix n, where n can range from 2 through 16.

Description

You can use the TYPE command to display the result of an expression. The expression can consist of any valid combination of program symbols, including constants and variables.

To examine the contents of a variable, the variable must be a scalar, a subscripted array element, or a constant of one of the following data types: bit, Boolean, character, complex, decimal, entry, file, float, integer, label, picture, or pointer. (Not every programming language provides all of these data types. Terminology may differ from one programming language to another.)

To examine the contents of a variable that is unknown in the working environment, you must fully qualify the reference. The only restriction is that an automatic or parameter variable can be examined only if it is known within the breakpoint environment.

If the variable's data type is integer or float, the debugger displays the variable as either a single- or double-precision value, depending on its declaration. Float variables are expressed in scientific notation.

If the variable's data type is pointer, file, entry, or label, the debugger displays the variable as one or more pointer values depending on the item's size. Each octal value consists of 6 digits (in the 16-bit SWAT debugger) or 11 digits (in the 32-bit SWAT debugger) followed by the R8 suffix.

Bit and character string results appear enclosed in quotation marks.

Enclose all subscripts in parentheses or square brackets. You can use an expression as a subscript. All variables in a subscript must have a defined value within the environment. A subscript expression must resolve to an integer value within the bounds of the array.

Arithmetic expressions can contain integer, float, or pointer values. Integer and float expressions can contain both single- and double-precision elements. The result of such an expression is a double-precision value.

By default, the SWAT debugger displays the expression result according to the data conversion rules described in Chapter 1. If no conversion is necessary, the debugger displays the result according to the declared or implicit data type of the value.

You can specify the display format for any expression by following the expression argument with a display format keyword. For example, let's assume KEYBOARD is a file constant. We enter the command

```
> TYPE KEYBOARD ↵
000500R8
```

To change the display format we can append the appropriate keyword:

```
> TYPE KEYBOARD @BIT ↵
"0000000101000000"

> TYPE KEYBOARD @CHARACTER ↵
"<001>@"

> TYPE KEYBOARD @FLOAT ↵
3.46462E-77

> TYPE KEYBOARD @INTEGER
320

> TYPE KEYBOARD @POINTER ↵
000500R8
```

When displaying a character string, the debugger represents nonprintable characters as octal values enclosed in angle brackets.

Floating-point values are expressed in scientific notation.

Pointer values appear as one or more octal pointer values.

If your program runs under the 32-bit SWAT debugger, you can display the expression result in a specific base. Use the keyword @Rn, where n ranges from 2 through 16.

You can provide more than one expression argument to the command. Each argument can have one data format keyword following it. To display a single expression in two or more formats, repeat the expression argument as often as necessary.

The SWAT debugger can display a string of up to 2048 characters. If you try to type a longer string, a warning message appears, then the debugger displays the first 2048 characters of the string.

Errors

COMMAND DOES NOT ACCEPT =

The expression cannot include an equal sign.

COMMAND REQUIRES ARGUMENT(S)

You must provide at least one expression as an argument to this command.

TYPE (continued)

ILLEGAL DATA TYPE symbol

You tried to display a variable whose data type is not one the SWAT debugger displays.

ILLEGAL PARENTHESIZATION

You did not enclose a subscript in matching parentheses or square brackets.

MISSING OPERATOR

The expression does not contain an arithmetic operator where the debugger expects one.

NOT A BASED VARIABLE symbol

The expression refers to a variable as if it were a based variable, but the variable is declared with another data type.

NOT A MEMBER VARIABLE symbol

The expression refers to a variable as if it were a member of a structure, but the variable is declared with another data type.

NOT A STRUCTURED VARIABLE symbol

The expression refers to a variable as if it were a structure, but the variable is declared with another data type.

NOT A VARIABLE symbol

The expression refers to a constant as if it were a variable.

NOT AN ARRAY VARIABLE symbol

The expression refers to a variable as if it were an array, but the variable is declared with another data type.

NOT AT A BREAKPOINT symbol

Program execution has not begun. You cannot display the contents of a stack variable until storage has been allocated.

NOT IN BREAKPOINT ENVIRONMENT

You tried to display a stack variable that has no currently defined value.

SUBSCRIPT OUT OF RANGE array

The specified subscript exceeds the declared bounds of the array.

STRING TOO LONG

The result of the expression is a string of more than 2048 characters. Only the first 2048 characters are displayed.

UNDEFINED SYMBOL symbol

The specified symbol does not exist within the current environment.

UNEXPECTED CHARACTER

The expression contains an illegal character.

UNEXPECTED OPERATOR operator

The expression includes an extra operator, or an argument must be preceded by a comma.

Examples

```
> TYPE SELECTION ↓
4

> type SELECTION @BIT ↓
"0000000000000100"

> ty RESPONSE ↓
"Y"

> ty DAILY_RATES.RATES(3) ↓
2.454300

> TY NAME ↓

Undefined symbol NAME

> TYPE CURRENCY(CODE) CURRENCY(CODE) @BIT ↓
"SRI"
"0101001101010010 01001001"
```

WALKBACK

Displays the current location and calling locations.

Format

WALKBACK *[COUNT=expression]*

where:

expression resolves to a positive integer value.

Description

Use the WALKBACK command to display the current location and the locators of calling statements.

If you enter the command with no argument, the debugger displays all calling locations. To display a specific number of calling locations, provide an expression that resolves to that number.

The format of the display is:

Current location is :environment:locator
[Called from :environment:locator]
[Undefined locator at nnnnnnR8]
[Begin block at :environment:locator]

When the calling procedure is one that cannot be debugged by the SWAT software, (it was not compiled and linked with the /DEBUG switch), the message

Undefined locator at nnnnnnR8

appears, indicating the location of the calling statement.

If a BEGIN block (as in PL/I) intervenes in the calling sequence, the 32-bit AOS/VS SWAT debugger displays the message

Begin block at :environment:locator

to report the location of the block. (The 16-bit SWAT debugger identifies a BEGIN block by including its line number in the environment specifier.)

Errors

INVALID EXTRA ARGUMENTS ON COMMAND

You included an unknown argument. (This error occurs if you omit the COUNT keyword.)

NOT AT A BREAKPOINT

The program has not begun execution. There is no invocation information to display.

Examples

```
> WALKBACK ⤸
Current location is :EXCHANGE:64

> walk COUNT=1 ⤸
Current location is :EXCHANGE:CHECK:85
Called from :EXCHANGE:64

> W
Current location is :CONVERSION:14
Called from :EXCHANGE:77
```

End of Chapter

Appendix A
Troubleshooting Tips
(Or What to Do if You're Having Trouble)

We provide this appendix to help you solve common problems more easily and quickly. If the suggestions outlined here don't give you an answer to your difficulties, consult your system manager or a Data General representative.

SWAT Check List

On the next page you'll find a list of requirements you must satisfy before you can successfully use the SWAT debugger. Go through the check list item by item making sure that you can answer YES to every question. If you can, you should have no trouble getting the SWAT software running.

If you run into problems after you begin debugging, refer to the next sections. They explain the differences between the AOS and AOS/VS SWAT debuggers, describe operations that are unique to the programming language you are using, and provide some helpful hints to get you over common obstacles.

	YES	NO	
	_____	_____	Does your system operate with the minimum software requirements described in the SWAT release notice? (Check with your system manager to ensure that your system has at least the minimum revisions of the operating system, the language compiler, and the Link utility that support the SWAT debugger, and that the software revisions are compatible.)
	_____	_____	Do you have access to the SWAT files? (If not, move a copy into a directory on your search list, or change your search list.)
	_____	_____	Has your system manager installed the SWAT error message and parameter files? (See the Release Notice.)
*	_____	_____	Does your user profile allow you to create a process without blocking?
	_____	_____	Does your user profile allow you to create at least three son processes?
*			(Your system operator can change your profile, if necessary.)
	_____	_____	Is your program written in a high-level language that the SWAT software can debug? (The SWAT Release Notice lists the programming languages that the SWAT debugger supports.)
	_____	_____	Did you compile at least one program module with the /DEBUG switch?
	_____	_____	Did you use the /DEBUG switch when linking the program modules together?
	_____	_____	Did you place the SWAT interface routine SWATI before the language library files in the Link command line?
	_____	_____	Did you link the program with the current (newest) version of the SWATI file?

Using the AOS/VS SWAT Debugger

The differences between the AOS/VS and AOS SWAT debuggers are few. The enhancements offered by the AOS/VS system mean that the AOS/VS SWAT debugger includes some additional features.

There are two versions of the SWAT software that you can run on an AOS/VS system: the 16-bit SWAT debugger (SWAT16) and the 32-bit SWAT debugger. You can use SWAT16 to debug 16-bit program running under AOS/VS. The differences between the AOS SWAT debugger and SWAT16 are noted in the manual.

The 32-bit AOS/VS SWAT Debugger

- supports decimal arithmetic.
- supports integer/floating-point mixed mode arithmetic.
- fully supports multitasking. Scheduling is disabled when the debugger encounters a breakpoint. When execution continues, scheduling returns to the state it was in just prior to the breakpoint trap. There is no additional overhead for single-task programs.
- includes the display format keyword @Rn where n can range from 2 through 16. This keyword specifies a radix for the display of numeric values.
- can be initiated by a process other than the Command Line Interpreter (CLI). This means that you can pass a non-CLI format initial IPC message to the SWAT debugger.
- supports the process switches /BREAK, /WSMAX, and /WSMIN. You can use these switches when invoking the SWAT debugger. (The AOS/VS 16-bit SWAT debugger also supports these switches.)

Other Considerations for the AOS/VS SWAT Debugger

- When using the AOS/VS user debugger, be aware that you cannot set a SWAT breakpoint where an assembly language breakpoint is already set. If an assembly language breakpoint and a SWAT breakpoint exist at the same instruction, you should clear the breakpoints in the opposite order that you set them.
- The debugger distinguishes between upper- and lowercase only when the programming language is case-sensitive.

Specific Language Considerations

If you are running PL/I:

In your Link command line, append the /START switch to the name of the main program module.

The keyword @MAIN refers to the highest level procedure of the first program module (.OB file) listed in the Link command line.

To build a locator to refer to a specific clause within a statement, use this format:

line-no.clause-no

The first clause on a line is clause 0, the next is clause 1, and so on. Note, however, that even if the clauses of a single statement actually appear on adjacent lines of the source code, the debugger treats them as if they were on the same line. So, use the locator that identifies the line where the statement begins, then append the clause number.

If you are running FORTRAN 77:

The identifier .MAIN represents an unnamed main program.

You can use the SET command to assign a value to a LOGICAL variable. To assign the value .FALSE., set the variable equal to 0. To assign the value .TRUE., set the variable equal to a nonzero integer value. For example:

```
SET L1 = 0      (L1 is .false.)
SET L1 = 1      (L1 is .true.)
SET L1 = -4     (L1 is .true.)
```

If you are running COBOL:

You can debug 32-bit COBOL programs with the AOS/VS SWAT debugger.

The program name is the PROGRAM-ID specified in the IDENTIFICATION DIVISION.

If you are running PASCAL:

The debugger recognizes type and constant identifiers, enumerated data types, and subrange data types; you can refer to these with SWAT commands in the appropriate context. You can also use the elementary items of packed and unpacked arrays and structures in an expression. You can refer to a packed array [1..n] of CHAR as an aggregate.

The DESCRIBE and TYPE commands let you work with set variables.

The postfix arrow operator (^) specifies a pointer or file buffer variable.

The debugger recognizes the PASCAL assignment operator (:=), as well as the use of apostrophes to identify a character string.

If you have external subprograms within a module compilation unit, the debugger treats them as internal environments within the external environment of the module.

If you are running C:

The debugger recognizes user-defined data types, unsigned integers, structures and unions, and C arrays; you can refer to these with SWAT commands in the appropriate context. You must fully qualify all references to structure and union members.

You cannot refer to unsubscripted array names. The debugger recognizes the following subscript notation for multidimension arrays:

array_name [1] [2] [n] ...

The debugger recognizes the star operator (*) for C pointers.

Common User Errors

The following section may provide some assistance when you're having trouble after the SWAT debugger begins executing.

The program begins execution, but does not pass control to the debugger:

Is your program written in a language that the SWAT software supports? See the SWAT Release Notice for details.

Did you place the SWAT Interface file (SWATI or SWATI16) before all language library files in the Link command? If not, relink the program modules. Be sure that you use the latest revision of the SWATI.OB or SWATI16.OB file.

The SWAT debugger doesn't recognize a symbol or locator:

Is the symbol or locator known within the working environment? If not, did you qualify the reference by preceding it with an environment specifier?

If the programming language is case-sensitive, did you enter the symbol exactly as it appears in the source text?

If you're specifying a structure member, did you fully qualify it?

If you're specifying a numeric label locator, did you precede it with an asterisk to distinguish it from a line number?

The SWAT debugger won't set a breakpoint:

Are you located in the correct environment? You must fully qualify a locator if it is a label outside the working environment, or a label outside the current module.

If you're setting a breakpoint at a clause, does the locator identify the line where the statement begins? If you append a clause number to the locator, does it point to the right statement?

Does the locator point to a nonexecutable statement or a line number that does not exist?

The SWAT debugger won't list a statement:

Does the locator identify a line within the current module?

Did you receive a warning message that indicated that the source file was not current?

The SWAT debugger won't assign a value to a variable:

Is the result of the expression argument compatible in data type and size with the receiving variable?

Does the expression contain incompatible data types?

If the expression contains an automatic or parameter variable, is the variable known within the breakpoint environment?

If the programming language is case-sensitive, did you enter the symbol name exactly as it appears in the source text?

Did you fully qualify a structure member reference?

The program (and the SWAT session) terminated after a CONTINUE statement:

You did not place a breakpoint within the path of execution (or the breakpoint carries an unexpired proceed count).

The SWAT debugger reports that a file is not current:

The debugger lines or debugger symbols files may not be compatible with the program file.

If you are using the LIST command, the source file has been modified since you compiled it.

The SWAT debugger does not let you display a structure member with the TYPE command:

Make sure that you have fully qualified the structure member.

Use the DESCRIBE command to display a description of the entire structure. This may explain why the debugger does not let you examine the member.

Helpful Hints

Help Information from the CLI

When you are in the CLI environment, you can use the command HELP *SWAT to obtain general information about the SWAT debugger. (The file CLI.TPC.SWAT must reside in the :HELP directory.)

Calling the User Debugger

If you intend to work with the AOS or AOS/VS user debugger, we suggest that you enable a second terminal for your program. This allows you to observe the program's interaction with the terminal, uninterrupted by SWAT dialog.

For an added margin of security when using the user debugger, you can call it through the SWAT debugger and work on a copy of the program. This allows you to keep the original program file intact.

Using Screenedit Features

If you are working at a DASHER® display terminal and have screen editing capabilities, you can use a multiple command line to great advantage. For example, the SWAT command line

```
> CONTINUE; TYPE ACCOUNT.CHECKING.CURRENT_BALANCE )
```

when combined with strategic breakpoints in the program and the CTRL-A key combination, allows you to monitor the value of a specific variable through program execution. This particular example also demonstrates the amount of rekeying time you can save.

Defining Extra Variables

You may find it helpful to define more variables than the program actually needs. The extra variables, such as statement labels, can aid your debugging activity. For example, you may want to assign meaningful labels to key statements throughout a large program. Using the LIST command, you can locate a specific block of code quickly. We suggest that you remove the extra variables after debugging a program; the compiler produces more efficient code if you eliminate unnecessary elements.

Using Pointer Arithmetic

By defining based variables of various types you can examine virtually any piece of storage that is accessible to your program. You can use pointer arithmetic to display the value of an offset storage location. You can use the pointer reference operator (^) by appending it to an expression. Thus, PTR -> based_var is equivalent to PTR^.

Simulating CLI Pseudo-Macros

You can simulate the CLI pseudo-macros [!OCTAL] and [!DECIMAL] with the TYPE command. To display the octal equivalent of a decimal value, use the @POINTER keyword with the command. To display the decimal equivalent of an octal value, append the R8 suffix to the octal argument.

End of Appendix

Appendix B
SWAT Error Messages

This appendix lists the error messages you may receive when using the SWAT debugger. Accompanying each error message is an explanation of what caused the error. If possible, we suggest how you can correct the error. *

SWAT Command Line Errors

BREAKPOINT ALREADY SET AT locator

You tried to set a breakpoint at a line that already has a breakpoint.

BREAKPOINT SIGNALLED INCORRECTLY

More than one user is executing the same program file.

CANNOT CONTINUE AT locator

The specified locator points to a nonexecutable or nonexistent statement.

CANNOT LIST LINE

The line numbers of the source file and program file do not match. The source file may have been altered since compilation.

CANNOT SET BREAKPOINT AT locator

You tried to set a breakpoint at a nonexecutable statement, a comment line, continuation line, or blank line.

COMMAND DOES NOT ACCEPT =

The expression argument cannot contain an equal sign.

COMMAND DOES NOT ACCEPT ARGUMENT(S)

There is no valid argument to the command. Enter only the command (or its abbreviation).

COMMAND ENDS ILLEGALLY

The debugger expected additional entries. You may have omitted a required argument.

COMMAND REQUIRES =

An equal sign must separate the variable and the expression.

COMMAND REQUIRES ARGUMENT(S)

You must provide at least one argument to the command.

COMMAND REQUIRES KEYWORD

You omitted a necessary keyword argument. Check the command syntax.

CONSOLE INTERRUPT

The debugger received an interrupt sequence (CTRL-C CTRL-A) from the keyboard.

ENVIRONMENT NOT FOUND specifier

The specified environment does not exist. Start at the root environment when referring to a block that is superior to the current environment. To specify an inferior environment, use only the label specifier.

FILE DOES NOT EXIST

(System error): The debugger cannot find the specified file.

ILLEGAL DATA TYPE symbol

The specified variable's data type prevents the debugger from setting or displaying its value.

ILLEGAL LANGUAGE SPECIFIER

The debugger cannot interpret your program. Contact your Data General representative.

ILLEGAL PARENTHESIZATION

The argument has missing or mismatched parentheses or square brackets.

INCORRECT NUMBER OF SUBSCRIPTS array

You specified either too many or too few subscripts for the array element. Use the DESCRIBE command to display information about the array and its dimensions.

INVALID COMMAND LINE command

You did not follow the syntax rules for the command. (This error appears only when the debugger cannot give a more detailed explanation.)

INVALID ENVIRONMENT SPECIFIER

You did not construct the environment specifier properly. Be sure that you correctly identify each procedure block and separate each element with a colon.

INVALID EXTRA ARGUMENTS ON COMMAND

You supplied more than the allowed number of arguments to the command.

INVALID LOCATOR locator

You specified a line number less than 1 or greater than 8191 (for AOS) or 32767 (for AOS/VS), or a nonexistent statement label. Be sure that you are in the correct environment, or have qualified the reference. To specify a numeric label, precede it with an asterisk (*990, for example). Clauses numbers cannot exceed 7 in AOS, or 32767 in AOS/VS.

INVALID NUMBER number

A constant is incorrect or out of range.

LOCATORS MUST BE IN SAME ENVIRONMENT

You cannot list a range of lines that extends beyond a single program module.

LINE NUMBER OUT OF RANGE

The specified line number lies outside the range of the current program module.

MISSING OPERATOR

The debugger tried to interpret your argument as an arithmetic expression, but could not locate a necessary operator.

NONUNIQUE COMMAND command

Your command or keyword abbreviation is too short. Expand the abbreviation so that the debugger can distinguish the command or keyword from all others.

NOT A BASED VARIABLE symbol

The variable you used with the pointer operator is not declared as a based variable. Use the DESCRIBE command to display information about the variable.

NOT A BREAKPOINT locator

The specified line does not have an attached breakpoint.

NOT A MEMBER VARIABLE symbol

The variable is not a member of a structure. Use the DESCRIBE command to display information about the variable.

NOT A STRUCTURED VARIABLE symbol

Your argument implies that a variable is a structured variable, but it is not declared that way.

NOT A VARIABLE symbol

The symbol you specified as a variable is not a variable.

NOT AN ARRAY VARIABLE symbol

You used a subscript with a scalar variable.

NOT AT A BREAKPOINT symbol

Program execution has not begun. You cannot display the contents of a stack variable until storage has been allocated for it, nor can you use the AT keyword when issuing the first CONTINUE command.

NOT IN BREAKPOINT ENVIRONMENT

You tried to display or set the contents of an automatic or parameter variable that does not have a defined value in the stack frame. This error also occurs if you try to redirect execution (CONTINUE AT) to a statement outside the breakpoint environment.

PROGRAM FILE ALREADY IN USE

The program file has been opened by another user.

SOME DATA MAY NOT BE CURRENT

This is a warning message. One or more system registers held data values when a trap occurred.

STRING TOO LONG

You provided a string argument longer than 2048 characters. The debugger performs the requested operation using only the first 2048 characters; the rest are ignored.

SUBSCRIPT OUT OF RANGE array

The specified subscript goes beyond the declared dimensions of the array. Use the DESCRIBE command to display information about the array.

TOO MANY BREAKPOINTS

You tried to set more than 20 breakpoints in the program.

UNDEFINED SYMBOL symbol

You specified a symbol that is not defined in the current environment. (If your programming language is case sensitive, be sure that you have entered the symbol correctly.)

UNEXPECTED CHARACTER

The debugger detected an illegal character in an argument.

UNEXPECTED OPERATOR operator

You used the indicated operator incorrectly, or did not separate multiple arguments with a comma.

UNKNOWN COMMAND

The command you entered is not a SWAT command. Check your spelling, or the menu of commands in Table 4-1.

UNKNOWN STORAGE CLASS symbol

You made an invalid reference through a PL/I VIRTUAL, %REPLACE, or EXTERNAL variable, or an F77 VIRTUAL, PARAMETER, or EXTERNAL variable.

UNKNOWN TOPIC topic

The specified topic is not on the help menu. Type HELP only to display the menu of available topics.

UNTERMINATED QUOTED STRING

You supplied a string argument that did not end with a quotation mark or apostrophe.

USER DATA BASE ERROR

You referred to a variable containing inconsistent data. Please report this error to a Data General Software Support Representative.

Start-Up and Termination Errors

You may receive error messages from the SWAT debugger as it begins execution or as it terminates. The following section describes error messages you may encounter.

Start-Up Errors

CALLER NOT PRIVILEGED FOR THIS ACTION

(System error): Your user profile does not allow you to create a process without blocking. The system operator can change your user profile.

FILE DOES NOT EXIST pathname

(System error): One or more necessary files do not exist or are not accessible. This is a fatal error.

FILE NOT CURRENT pathname

This is a warning message. The creation date of the specified file is incompatible with the creation date of related files. The original file may have been modified. The debugger will, however, try to use the existing file.

PROGRAM FILE ALREADY IN USE

You used the /NOCOPY switch to debug the original program file. But, another user has already opened the file. Unexpected results can occur when there are multiple users.

SWAT CANNOT START UP

Check your user profile to ensure that you have the necessary privileges listed in Chapter 2.

SWAT COMMAND LINE ERROR

The SWAT invocation command contained an error.

SWITCH DOES NOT ACCEPT ARGUMENT

The SWAT invocation command included one or more switches with illegal arguments. Chapter 2 describes all the command switches.

SWITCH REQUIRES ARGUMENT

The SWAT invocation command included one or more switches that did not have a required argument. Chapter 2 describes all the command switches and their arguments.

Termination Errors

SWAT TERMINATED

This message appears following program termination or a BYE command. The SWAT debugger terminates normally, unless one of the following messages also appears:

SYSTEM TRAP
TERMINATED BY CONSOLE INTERRUPT
TERMINATED BY A SUPERIOR PROCESS
TERMINATED BY SYSTEM
TERMINATION SIGNALLED INCORRECTLY
USER TRAP

If a trap occurs, the termination message specifies the condition that caused the trap.

SWAT Maintenance Errors

The SWAT debugger reports certain error messages for internal maintenance purposes. If you receive any of the following error messages, contact your Data General Software Support Representative as soon as possible.

***** *INTERNAL FATAL ERROR*

***** *INTERNAL NONFATAL ERROR*

* *MODULE CALLED INCORRECTLY*

System Error Messages

If a system related error occurs while you are debugging, the SWAT software relays the message to you. System errors can occur whenever the SWAT debugger makes a system call. Refer to the appropriate AOS or AOS/VS manual for information about specific system errors and their correction.

End of Appendix

Appendix C
More Examples

This appendix provides additional examples of PL/I, FORTRAN 77, COBOL, PASCAL, and C programs, and audit files that illustrate the use of SWAT commands with these programs.

The first example illustrates the same PL/I program used in the sample session in Chapter 3. In this case, however, the program is compiled and linked on an AOS/VS system, then run using the 32-bit AOS/VS SWAT debugger.

Figures C-1 and C-2 present the compilation listings of the EXCHANGE and CONVERSION modules, respectively. Figure C-3 presents the audit files of sample sessions similar to those described in Chapter 3.

```
Source file: EXCHANGE
Compiled on 24-Nov-81 at 14:40:38 by AOS/VS PL/I Rev 01.20.02.00
Options: PL1/DEBUG/L=EXLIST

    1  EXCHANGE:
    2          PROCEDURE;
    3
    4          /* This program calculates currency exchange values based on
    5  *       daily international exchange rates.  The operator selects
    6  *       the type of exchange then enters the incoming amount.  The
    7  *       program performs the calculation and displays the result. */
    8
    9          DECLARE (SCREEN, KEYBOARD, RATE_FILE) FILE;
   10          DECLARE CONVERSION ENTRY(FIXED BINARY,FIXED DECIMAL(10,6));
   11          DECLARE SELECTION FIXED BINARY;
   12          DECLARE 1 DAILY_RATES,
   13                  2 RATE_DATE CHARACTER(6),
   14                  2 RATES(16) FIXED DECIMAL(10,6);
   15          DECLARE RESPONSE CHARACTER(1);
   16          DECLARE INDX FIXED BINARY;
   17
   18          OPEN FILE(SCREEN) STREAM OUTPUT PRINT TITLE('@OUTPUT');
   19          OPEN FILE(KEYBOARD) STREAM INPUT TITLE('@INPUT');
   20          OPEN FILE(RATE_FILE) STREAM INPUT;
   21
   22  /* Read in today's exchange rates.  Compare date with system date. */
   23
   24          GET FILE(RATE_FILE) LIST(DAILY_RATES);
   25
   26          ON ENDFILE(RATE_FILE)
   27          BEGIN;
   28                  PUT FILE(SCREEN) SKIP LIST('NO RATES AVAILABLE.');
   29                  STOP;
   30          END; /*BEGIN Block for End of File condition */
   31
   32  /* Compare file's date with the system date. */
   33
```

Figure C-1. Compilation Listing of the Module EXCHANGE (AOS/VS) (continues)

```
34          IF RATE_DATE ^= DATE() THEN DO;
35                  PUT FILE(SCREEN) SKIP LIST("RATES NOT CURRENT");
36                  STOP;
37                  END; /*DO Block */
38
39  /* Set up ON ERROR condition for bad input */
40
41          ON ERROR
42                  BEGIN;
43                  PUT FILE(SCREEN) SKIP LIST("Invalid input.  "
44                                                  !!"Try again.");
45                  GO TO LOOP;
46                  END;
47
48  /* Display menu for exchanges */
49
50          LOOP:   DO WHILE("1"B);
51
52                  PUT FILE(SCREEN) SKIP(3) LIST("Select currency code for "!!
53                       "US$ exchange:");
54                  PUT FILE(SCREEN) SKIP(2) EDIT
55                       ("1","Belgian francs","2","W. German marks","3",
56                       "Dutch guilders","4","Italian lire","5","Swiss francs",
57                       "6","Canadian dollars","7","Saudi riyals","8",
58                       "Japanese yen","(Type 0 to end the program)",
59                       "Enter the currency code:  ")
60                       (8(X(5),A(1),X(3),A(16),SKIP),SKIP,A(27),SKIP,A(26));
61
62                  GET FILE(KEYBOARD) LIST(SELECTION);
63
64                  CALL CHECK(SELECTION);
65
66  /* Request type of exchange */
67
68          PUT FILE(SCREEN) SKIP LIST("Do you want to convert US$ into"
69                  !!" Foreign currency?");
70          PUT FILE(SCREEN) SKIP LIST("Enter 'Y' or 'N':  ");
71          GET FILE(KEYBOARD) LIST(RESPONSE);
72
73          IF RESPONSE = "Y" THEN INDX = SELECTION;
74          ELSE IF RESPONSE = "y" THEN INDX = SELECTION;
75                  ELSE INDX = SELECTION + 8;
76
77          CALL CONVERSION(INDX, RATES(INDX));
78
79  END;/* LOOP */
80
81  CHECK:
82          PROCEDURE(SEL);
83          DECLARE SEL FIXED BINARY;
84
85          IF SEL > 0 THEN
86                  IF SEL < 9 THEN RETURN;
87          PUT FILE(SCREEN) SKIP LIST("Exchange program ended.");
88          STOP;
89  END;/* CHECK SUBROUTINE */
90
91  END;/* EXCHANGE */
```

Figure C-1. Compilation Listing of the Module EXCHANGE (AOS/VS) (continued)

```
Source file: EXCHANGE
Compiled on 24-Nov-81 at 14:40:44 by AOS/VS PL/I Rev 01.20.02.00

                          Allocation Map


EXTERNAL ENTRY POINTS

NAME          CLASS       SIZE   LOC    ATTRIBUTES

EXCHANGE      CONSTANT                  ENTRY EXTERNAL


PROCEDURE EXCHANGE ON LINE 1

NAME          CLASS       SIZE   LOC    ATTRIBUTES

SCREEN        CONSTANT                  FILE EXTERNAL
KEYBOARD      CONSTANT                  FILE EXTERNAL
RATE_FILE     CONSTANT                  FILE EXTERNAL
CONVERSION    CONSTANT                  ENTRY EXTERNAL
SELECTION     AUTOMATIC   1W     12W    FIXED BIN(15,0)
DAILY_RATES   AUTOMATIC   102C   14W    STRUCTURE
RATE_DATE     MEMBER      6C     0C     CHAR(6)
RATES         MEMBER      96C    6C     FIXED DEC(10,6) DIMENSION
RESPONSE      AUTOMATIC   1C     65W    CHAR(1)
INDX          AUTOMATIC   1W     13W    FIXED BIN(15,0)
LOOP          CONSTANT                  LABEL
CHECK         CONSTANT                  ENTRY INTERNAL
DATE          CONSTANT                  BUILTIN


BEGIN BLOCK (ON UNIT) ON LINE 26

NO DECLARED NAMES


BEGIN BLOCK (ON UNIT) ON LINE 41

NO DECLARED NAMES


PROCEDURE CHECK ON LINE 81

NAME          CLASS       SIZE   LOC    ATTRIBUTES

SEL           PARAMETER   1W             FIXED BIN(15,0)
```

Figure C-1. Compilation Listing of the Module EXCHANGE (AOS/VS) (concluded)

```
Source file: CONVERSION
Compiled on 24-Nov-81 at 14:41:05 by AOS/VS PL/I Rev 01.20.02.00
Options: PL1/DEBUG/L=CONLIST

    1
    2   CONVERSION:
    3           PROCEDURE(CODE, RATE);
    4
    5           DECLARE CODE FIXED BINARY;
    6           DECLARE RATE FIXED DECIMAL(10,6);
    7           DECLARE CURRENCY(8) CHARACTER(4) VARYING STATIC INTERNAL
    8                   INIT("BF","DM","gld","Lit","SF","Can$","SRI","Y");
    9           DECLARE FVAL FIXED DECIMAL(11,2);
   10           DECLARE DVAL PICTURE "$$$$$$$$9.99";
   11           DECLARE (SCREEN, KEYBOARD) FILE;
   12
   13           IF CODE < 8 THEN DO;
   14                           PUT FILE(SCREEN) SKIP LIST("ENTER US$: ");
   15                           GET FILE(KEYBOARD) LIST(DVAL);
   16                           FVAL = DVAL * RATE;
   17                           PUT FILE(SCREEN) EDIT(DVAL,
   18           " US equivalent to:  ",FVAL, CURRENCY(CODE))
   19           (SKIP(2),X(5),F(11,2),A(20),F(11,2),X(1),A(4),SKIP(2));
   20                           RETURN;
   21                           END; /* DO GROUP FOR DOLLAR CONVERSION */
   22
   23           ELSE DO;
   24           PUT FILE(SCREEN) SKIP LIST("ENTER ",CURRENCY(CODE-8),":  ");
   25           GET FILE(KEYBOARD) LIST(FVAL);
   26           DVAL = FVAL * RATE;
   27           PUT FILE(SCREEN) EDIT(FVAL, CURRENCY(CODE-8),
   28           " equivalent to:  ",DVAL," US")
   29           (SKIP(2), X(5),F(11,2),X(1),A(4),A(17),F(11,2),A(3),SKIP(2));
   30           RETURN;
   31           END; /* DO GROUP FOR FOREIGN CURRENCY CONVERSION */
   32
   33   END; /* CONVERSION */
```

Figure C-2. Compilation Listing of the Module CONVERSION (AOS/VS) (continues)

```
Source file: CONVERSION
Compiled on 24-Nov-81 at 14:41:10 by AOS/VS PL/I Rev 01.20.02.00

                        Allocation Map


EXTERNAL ENTRY POINTS

NAME          CLASS       SIZE   LOC    ATTRIBUTES

CONVERSION    CONSTANT                  ENTRY EXTERNAL


PROCEDURE CONVERSION ON LINE 2

NAME          CLASS       SIZE   LOC    ATTRIBUTES

CODE          PARAMETER   1W             FIXED BIN(15,0)
RATE          PARAMETER   6C             FIXED DEC(10,6)
CURRENCY      STATIC      24W    0W      CHAR(4) VARYING DIMENSION INTERNAL
                                         INITIAL
FVAL          AUTOMATIC   6C     24W     FIXED DEC(11,2)
DVAL          AUTOMATIC   12C    28W     PICTURE
SCREEN        CONSTANT                  FILE EXTERNAL
KEYBOARD      CONSTANT                  FILE EXTERNAL
```

Figure C-2. Compilation Listing of the Module CONVERSION (AOS/VS) (concluded)

```
------------------------------------------------------------
USER PROGRAM exchange SWAT AUDIT ON 11/24/81 AT 10:09:13

AOS/VS SWAT Revision 02.00.00.00 ON 11/24/81 AT 10:09:14
PROGRAM -- :UDD:TOM:EXCHANGE
> PREFIX "NEXT COMMAND?"
NEXT COMMAND? PREFIX ">"
> AUDIT
ON
> AUDIT OFF

------------------------------------------------------------
USER PROGRAM :UDD:TOM:EXCHANGE SWAT AUDIT ON 11/24/81 AT 10:10:34

ON
> %This is a comment line.
> ENVIRONMENT
:EXCHANGE
> ENVIRONMENT :CONVERSION
:CONVERSION
> ENVIRONMENT @MAIN
:EXCHANGE
> BREAKPOINT 24
Set at :EXCHANGE:24
> BREAKPOINT 34 62 64 87
Set at :EXCHANGE:34
Set at :EXCHANGE:62
Set at :EXCHANGE:64
Set at :EXCHANGE:CHECK:87
> BREAKPOINT :CONVERSION:13
Set at :CONVERSION:13
> ENVIRONMENT :CONVERSION
:CONVERSION
> BREAKPOINT 16 26
Set at :CONVERSION:16
Set at :CONVERSION:26
> BREAKPOINT
Set at :EXCHANGE:24
Set at :EXCHANGE:34
Set at :EXCHANGE:62
Set at :EXCHANGE:64
Set at :EXCHANGE:CHECK:87
Set at :CONVERSION:13
Set at :CONVERSION:16
Set at :CONVERSION:26
> ENVIRONMENT @MAIN
:EXCHANGE
> BREAKPOINT 28
Set at :EXCHANGE:28
```

Figure C-3. Audit File of the EXCHANGE Program SWAT Session (continues)

```
> BREAKPOINT 43
Set at :EXCHANGE:43
> BREAKPOINT 71 COUNT=5
Set at :EXCHANGE:71 count=5
> BREAKPOINT 71 COUNT=4
Reset at :EXCHANGE:71 count=4
> BREAKPOINT 71 ACTION = "LIST 71"
Reset at :EXCHANGE:71 count=4 action="LIST 71"
> CLEAR 71
Cleared at :EXCHANGE:71 count=4 action="LIST 71"
> CLEAR 28 43 62 87
Cleared at :EXCHANGE:28
Cleared at :EXCHANGE:43
Cleared at :EXCHANGE:62
Cleared at :EXCHANGE:CHECK:87
> BREAKPOINT
Set at :EXCHANGE:24
Set at :EXCHANGE:34
Set at :EXCHANGE:64
Set at :CONVERSION:13
Set at :CONVERSION:16
Set at :CONVERSION:26
> CLEAR :CONVERSION:13
Cleared at :CONVERSION:13
> CLEAR @ALL
Cleared at :EXCHANGE:24
Cleared at :EXCHANGE:34
Cleared at :EXCHANGE:64
Cleared at :CONVERSION:16
Cleared at :CONVERSION:26
> BREAKPOINT
> LIST 26
26              ON ENDFILE(RATE_FILE)
> LIST 26 30
26              ON ENDFILE(RATE_FILE)
27              BEGIN;
28                      PUT FILE(SCREEN) SKIP LIST("NO RATES AVAILABLE.");
29                      STOP;
30              END; /*BEGIN Block for End of File condition */
> BREAKPOINT 24 34 64
Set at :EXCHANGE:24
Set at :EXCHANGE:34
Set at :EXCHANGE:64
```

Figure C-3. Audit File of the EXCHANGE Program SWAT Session (continued)

```
> LIST 20 35
20              OPEN FILE(RATE_FILE) STREAM INPUT;
21
22      /* Read in today's exchange rates.  Compare date with system date. */
23
24B             GET FILE(RATE_FILE) LIST(DAILY_RATES);
25
26              ON ENDFILE(RATE_FILE)
27              BEGIN;
28                      PUT FILE(SCREEN) SKIP LIST("NO RATES AVAILABLE.");
29                      STOP;
30              END; /*BEGIN Block for End of File condition */
31
32      /* Compare file's date with the system date. */
33
34B             IF RATE_DATE ^= DATE() THEN DO;
35                      PUT FILE(SCREEN) SKIP LIST("RATES NOT CURRENT");
> CONTINUE

Breakpoint trap at :EXCHANGE:24
> ENVIRONMENT
:EXCHANGE
> BREAKPOINT
Set at :EXCHANGE:24
Set at :EXCHANGE:34
Set at :EXCHANGE:64
Current location is :EXCHANGE:24
> ENVIRONMENT
:EXCHANGE
> ENVIRONMENT :CONVERSION
:CONVERSION
> BREAKPOINT 16 26
Set at :CONVERSION:16
Set at :CONVERSION:26
> ENVIRONMENT @BREAK
:EXCHANGE
> LIST 20 28
20              OPEN FILE(RATE_FILE) STREAM INPUT;
21
22      /* Read in today's exchange rates.  Compare date with system date. */
23
24C             GET FILE(RATE_FILE) LIST(DAILY_RATES);
25
26              ON ENDFILE(RATE_FILE)
27              BEGIN;
28                      PUT FILE(SCREEN) SKIP LIST("NO RATES AVAILABLE.");
```

Figure C-3. Audit File of the EXCHANGE Program SWAT Session (continued)

```
> LIST @BREAK
14                      2 RATES(16) FIXED DECIMAL(10.6);
15              DECLARE RESPONSE CHARACTER(1);
16              DECLARE INDX FIXED BINARY;
17
18              OPEN FILE(SCREEN) STREAM OUTPUT PRINT TITLE("@OUTPUT");
19              OPEN FILE(KEYBOARD) STREAM INPUT TITLE("@INPUT");
20              OPEN FILE(RATE_FILE) STREAM INPUT;
21
22      /* Read in today's exchange rates.  Compare date with system date. */
23
24C              GET FILE(RATE_FILE) LIST(DAILY_RATES);
25
26              ON ENDFILE(RATE_FILE)
27              BEGIN;
28                      PUT FILE(SCREEN) SKIP LIST("NO RATES AVAILABLE.");
29                      STOP;
30              END; /*BEGIN Block for End of File condition */
31
32      /* Compare file's date with the system date. */
33
34B             IF RATE_DATE ^= DATE() THEN DO;
> CONTINUE

Breakpoint trap at :EXCHANGE:34
>
24B             GET FILE(RATE_FILE) LIST(DAILY_RATES);
25
26              ON ENDFILE(RATE_FILE)
27              BEGIN;
28                      PUT FILE(SCREEN) SKIP LIST("NO RATES AVAILABLE.");
29                      STOP;
30              END; /*BEGIN Block for End of File condition */
31
32      /* Compare file's date with the system date. */
33
34C             IF RATE_DATE ^= DATE() THEN DO;
35                      PUT FILE(SCREEN) SKIP LIST("RATES NOT CURRENT");
36                      STOP;
37                      END; /*DO Block */
38
39      /* Set up ON ERROR condition for bad input */
40
41              ON ERROR
42                      BEGIN;
43                      PUT FILE(SCREEN) SKIP LIST("Invalid input.  "
44                                                      !!"Try again.");
> CONTINUE COUNT=2
Reset at :EXCHANGE:34 count=2

Breakpoint trap at :EXCHANGE:64
```

Figure C-3. Audit File of the EXCHANGE Program SWAT Session (continued)

```
> CLEAR 34
Cleared at :EXCHANGE:34 count=2
> CONTINUE AT LOOP

Breakpoint trap at :EXCHANGE:64
> TYPE DAILY_RATES.RATE_DATE
"811124"
> TYPE SELECTION
2
> TYPE RESPONSE
"<000>"
> TYPE INDX
0
> TYPE DAILY_RATES.RATES(4)
+1194.
> TYPE SELECTION @BIT
"0000000000000010"B
> TYPE SELECTION @CHARACTER
"<000><002>"
> TYPE SELECTION @FLOAT
6.747007E-80
> TYPE SELECTION @INTEGER
2
> TYPE SELECTION @POINTER
400000R8
> TYPE SELECTION @R4
2R4
> TYPE SELECTION + 8
10
> TYPE SELECTION + INDX
2
> TYPE INDX - SELECTION
-2
> % Integer division produces integer results.
> TYPE 34/2
17
> TYPE 35/2
17
> TYPE 999999999 * 10
1410065398
>% Overflow causes a meaningless result.
> TYPE 999999999. * 10.
 9.999999990000000E+09
> TYPE 403R8
259
> TYPE DAILY_RATES.RATES(6)
+1.1865
> SET DAILY_RATES.RATES(6) = DAILY_RATES.RATES(14)
Old value: +1.1865
New Value: +.8434
```

Figure C-3. Audit File of the EXCHANGE Program SWAT Session (continued)

```
> SET DAILY_RATES.RATES(6) = "abcd"
Old value: +.8434

ILLEGAL DATA TYPE
> DESCRIBE SELECTION
SELECTION (1 word at +14R8 words) fixed binary(15) aligned automatic
> DESCRIBE SCREEN
SCREEN (40 words at 16000000726R8 words) file constant external
> DESCRIBE DAILY_RATES
DAILY_RATES (102 bytes at +16R8 words) automatic
    2 RATE_DATE (6 bytes at +0R8 bytes) character(6)
    2 RATES (96 bytes at +6R8 bytes) (1:16) fixed decimal(10,6) aligned
> DESCRIBE :CONVERSION
CONVERSION (at 16001746130R8 words) entry line 5 to 33 constant external
> HELP

topics are:

        @ALL            @BIT            @BREAK          @CHARACTER
        @FLOAT          @INTEGER        @MAIN           @POINTER
        ACTION          AT              AUDIT           BREAKPOINT
        BYE             CLEAR           CLI             COMMANDS
        COMMENTS        CONTINUE        COUNT           DEBUG
        DESCRIBE        DIRECTORY       ENVIRONMENT     EXECUTE
        EXPRESSIONS     HELP            KEYWORDS        LIST
        LOCATORS        MESSAGE         OFF             ON
        OPERATORS       PREFIX          PROMPT          SEARCHLIST
        SET             SPECIFIERS      TYPE            UNIQUENESS
        WALKBACK
> HELP BREAKPOINT

        Command: BREAKPOINT
             or: BREAKPOINT locator [COUNT=expression] [ACTION="string"]

        BREAKPOINT lists all the breakpoints currently set,
        their environment, their counts (if any), their action
        (if any), and the current location (if any).
        BREAKPOINT locator sets a breakpoint at locator. If the
        optional COUNT=expression is supplied, the breakpoint will
        be given an initial count of expression. If the optional
        ACTION="string" is supplied, the SWAT command(s) in the
        string will be executed when the breakpoint is signalled.
        To change the count or action on a breakpoint already set,
        repeat this command with the new count expression and/or
        the new action string.
```

Figure C-3. Audit File of the EXCHANGE Program SWAT Session (continued)

```
> HELP LOCATORS

        Locators are identifiers used for naming a location in the
        user program. They can be in the form of a line number, a
        label, or a numeric label preceded by a "*". Examples are:
        19, START or *100. If there is more than one statement on
        a single line, they are given clause number of the form
        19.1, 19.2, 19.3, etc.

> MESSAGE 34
000042 CONSOLE DEVICE SPECIFICATION ERROR

> MESSAGE 34R8
000034 APPEND AND/OR WRITE ACCESS DENIED

> MESSAGE SELECTION SELECTION+15
000002 CHANNEL NOT OPEN
000021 FILE SPACE EXHAUSTED

> DIRECTORY
:UDD:TOM
> SEARCHLIST
:UDD:TOM :UTIL :LINKS :MACROS
> SEARCHLIST ":UDD:TOM" ":UTIL" ":COMMON"
:UDD:TOM :UTIL :COMMON
> EXECUTE "DBUG.LIB"
>>ENVIRONMENT
:EXCHANGE
>>LIST @BREAK
54              PUT FILE(SCREEN) SKIP(2) EDIT
55                      ("1","Belgian francs","2","W. German marks","3",
56                      "Dutch guilders","4","Italian lire","5","Swiss francs",
57                      "6","Canadian dollars","7","Saudi riyals","8",
58                      "Japanese yen","(Type 0 to end the program)",
59                      "Enter the currency code: ")
60                      (8(X(5),A(1),X(3),A(16),SKIP),SKIP,A(27),SKIP,A(26));
61
62              GET FILE(KEYBOARD) LIST(SELECTION);
63
64C             CALL CHECK(SELECTION);
65
66      /* Request type of exchange */
67
68              PUT FILE(SCREEN) SKIP LIST("Do you want to convert US$ into"
69                      !!" Foreign currency?");
70              PUT FILE(SCREEN) SKIP LIST("Enter 'Y' or 'N': ");
71              GET FILE(KEYBOARD) LIST(RESPONSE);
72
73              IF RESPONSE = "Y" THEN INDX = SELECTION;
74              ELSE IF RESPONSE = "y" THEN INDX = SELECTION;
```

Figure C-3. Audit File of the EXCHANGE Program SWAT Session (continued)

```
> PROMPT
"LIST @BREAK"
> PROMPT "CONTINUE"
>

Breakpoint trap at :CONVERSION:16
> BYE

SWAT TERMINATED

------------------------------------------------------------
USER PROGRAM exchange SWAT AUDIT ON 11/24/81 AT 12:44:38

AOS/VS SWAT Revision 02.00.00.00 ON 11/24/81 AT 12:44:41
PROGRAM -- :UDD:TOM:EXCHANGE
> % Set breakpoints at each ON unit and each subroutine call.
> BREAKPOINT 28 ACTION="LIST 28"
Set at :EXCHANGE:28 action="LIST 28"
> BREAKPOINT 43 ACTION="LIST 43"
Set at :EXCHANGE:43 action="LIST 43"
> BREAKPOINT 64 ACTION="LIST 64"
Set at :EXCHANGE:64 action="LIST 64"
> BREAKPOINT 77 ACTION="LIST 77"
Set at :EXCHANGE:77 action="LIST 77"
> % Set breakpoints in the subroutine CONVERSION.
> BREAKPOINT :CONVERSION:13 ACTION="LIST 13"
Set at :CONVERSION:13 action="LIST 13"
> BREAKPOINT :CONVERSION:16 ACTION="LIST 16"
Set at :CONVERSION:16 action="LIST 16"
> BREAKPOINT :CONVERSION:24 ACTION="LIST 24"
Set at :CONVERSION:24 action="LIST 24"
> BREAKPOINT :CONVERSION:26 ACTION="LIST 26"
Set at :CONVERSION:26 action="LIST 26"
> CONTINUE

Breakpoint trap at :EXCHANGE:64 action="LIST 64"
64C                     CALL CHECK(SELECTION);
> % We entered a currency code of 5 -- the code for Swiss francs.
> TYPE SELECTION
5
> % SELECTION is the currency code variable.
> CONTINUE

Breakpoint trap at :EXCHANGE:77 action="LIST 77"
77C             CALL CONVERSION(INDX, RATES(INDX));
> % We responded No to the exchange direction question.
> TYPE INDX
13
>TYPE DAILY_RATES.RATES(13)
+.5566
```

Figure C-3. Audit File of the EXCHANGE Program SWAT Session (continued)

```
> % The value of INDX is correct.  We displayed the exchange rate.
> CONTINUE

Breakpoint trap at :CONVERSION:13 action="LIST 13"
13C             IF CODE < 8 THEN DO:
> TYPE CODE
13
> CONTINUE

Breakpoint trap at :CONVERSION:24 action="LIST 24"
24C             PUT FILE(SCREEN) SKIP LIST("ENTER ",CURRENCY(CODE-8),":  ");
> TYPE CODE-8
5
> TYPE CURRENCY(5)
"SF"
> CONTINUE

Breakpoint trap at :CONVERSION:26 action="LIST 26"
26C             DVAL = FVAL * RATE;
> % We entered 250 Swiss francs for the exchange.
> TYPE FVAL
+250.
> % This is the number of Swiss francs we entered.
> TYPE RATE
+.5566
> % This is the same rate we observed earlier.
> TYPE 250.00 * .556600
1.39150000000000000E+02
> % The program should output $139.15 as its result.
> CONTINUE

Breakpoint trap at :EXCHANGE:64 action="LIST 64"
64C                     CALL CHECK(SELECTION);
> % The output value was $139.00. but should have been $139.15.
> % Change the output statement. line 29 of the :CONVERSION module.
> % -- The second F(11.2) should read P"$$$$$$$$9V.99"
> % The program displayed the menu.  We entered code 5 again.
> TYPE SELECTION
5
> % This corresponds to the currency code we input at the keyboard.
> CONTINUE

Breakpoint trap at :EXCHANGE:77 action="LIST 77"
77C             CALL CONVERSION(INDX, RATES(INDX));
> % We responded Yes to the exchange direction question.  (US$ to SF)
> TYPE INDX
5
> % INDX points to exchange rate for SF in first half of table.
> TYPE DAILY_RATES.RATES(5)
+1.7725
```

Figure C-3. Audit File of the EXCHANGE Program SWAT Session (continued)

```
> % This rate should also appear as RATE in the CONVERSION routine.
> CONTINUE

Breakpoint trap at :CONVERSION:13 action="LIST 13"
13C             IF CODE < 8 THEN DO;
> % CODE should equal 5 and correspond to INDX in the calling routine.
> TYPE CODE
5
> CONTINUE

Breakpoint trap at :CONVERSION:16 action="LIST 16"
16C                             FVAL = DVAL * RATE;
> % The program asked for the US dollar amount; we entered 250.
> % DVAL holds the number of American dollars to be converted.
> TYPE DVAL
"       $2.50"
> LIST 1 10
1
2       CONVERSION:
3               PROCEDURE(CODE, RATE);
4
5               DECLARE CODE FIXED BINARY;
6               DECLARE RATE FIXED DECIMAL(10,6);
7               DECLARE CURRENCY(8) CHARACTER(4) VARYING STATIC INTERNAL
8                       INIT("BF","DM","gld","Lit","SF","Can$","SRI","Y");
9               DECLARE FVAL FIXED DECIMAL(11,2);
10              DECLARE DVAL PICTURE "$$$$$$$$9.99";
> TYPE RATE
+1.7725
> % The rate matches the rate we observed in the EXCHANGE module.
> TYPE 1.7725 * 250.
4.43125000000000000E+02
> CONTINUE

Breakpoint trap at :EXCHANGE:64 action="LIST 64"
64C                     CALL CHECK(SELECTION);
> % We entered currency code 8 for Japanese yen.
> TYPE SELECTION
8
> % SELECTION holds the correct value.
> CONTINUE

Breakpoint trap at :EXCHANGE:77 action="LIST 77"
77C             CALL CONVERSION(INDX, RATES(INDX));
> % We entered N to the exchange direction question.
> TYPE INDX
16
> TYPE DAILY_RATES.RATES(INDX)
+.004471
```

Figure C-3. Audit File of the EXCHANGE Program SWAT Session (continued)

```
> % This should be the rate we see in the calculation statement.
> CONTINUE

Breakpoint trap at :CONVERSION:13 action="LIST 13"
13C             IF CODE < 8 THEN DO;
> TYPE CODE
16
> % This corresponds to INDX: it is correct.
> CONTINUE

Breakpoint trap at :CONVERSION:24 action="LIST 24"
24C             PUT FILE(SCREEN) SKIP LIST("ENTER ",CURRENCY(CODE-8),":  ");
> TYPE CODE-8
8
> % This subscript is correct: it points to the symbol for yen.
> CONTINUE

Breakpoint trap at :CONVERSION:26 action="LIST 26"
26C             DVAL = FVAL * RATE;
> % We entered the value 560 for the number of yen.
> TYPE FVAL
+560.
> TYPE RATE
+.004471
> % This value corresponds to the rate displayed earlier.
> TYPE 560. * .004471
2.503760000000000E+00
> CONTINUE

Breakpoint trap at :EXCHANGE:64 action="LIST 64"
64C                     CALL CHECK(SELECTION);
> % The result was $2.00.  It should have been $2.50.
> % The errors we've already found explain these problems.
> % We again entered currency code 8.
> TYPE SELECTION
8
> CONTINUE

Breakpoint trap at :EXCHANGE:77 action="LIST 77"
77C             CALL CONVERSION(INDX, RATES(INDX));
> % We responded Yes to the exchange direction question. (US$ to yen)
> TYPE INDX
8
> % The index is correct.
> % It points to the entry for yen in the first half of the table.
> TYPE DAILY_RATES.RATES(8)
+225.9
> CONTINUE

Breakpoint trap at :CONVERSION:13 action="LIST 13"
13C             IF CODE < 8 THEN DO;
```

Figure C-3. Audit File of the EXCHANGE Program SWAT Session (continued)

```
> TYPE CODE
8
> % Change statement 13 in :CONVERSION.  It should read:
> %     IF CODE < 9 THEN DO;
> CONTINUE

Breakpoint trap at :CONVERSION:24 action="LIST 24"
24C             PUT FILE(SCREEN) SKIP LIST("ENTER ",CURRENCY(CODE-8),":  ");
> TYPE CODE-8
0
> CONTINUE AT 14

Breakpoint trap at :CONVERSION:16 action="LIST 16"
16C                             FVAL = DVAL * RATE;
> TYPE DVAL
"       $2.50"
> TYPE RATE
+225.9
> TYPE 250. * 225.90
5.6475000000000000E+04
> CONTINUE

Breakpoint trap at :EXCHANGE:64 action="LIST 64"
64C                     CALL CHECK(SELECTION);
> BYE

SWAT TERMINATED
```

Figure C-3. Audit File of the EXCHANGE Program SWAT Session (concluded)

A Sample FORTRAN 77 Program and Audit File

This F77 program calculates and displays a mortgage payment information. The program consists of three external modules: the main program (called LOAN), and the subroutines PAYMENT and FULL. The main program interacts with the operator, requesting principal, interest rate, and the number of years. The subroutine PAYMENT calculates the monthly payment for the specified loan. The main program asks the operator if a full schedule is desired. If so, the main program calls the subroutine FULL, which calculates the amount of interest and principal paid in each monthly payment, the remaining principal, and the interest paid to date. This routine writes the information to the current LIST file.

The compilation commands are:

```
F77/DEBUG/L=LOAN.LS LOAN.F77

F77/DEBUG/L=PAYMENT.LS PAYMENT.F77

F77/DEBUG/L=FULL.LS FULL.F77
```

The link command line is:

```
F77LINK/DEBUG LOAN PAYMENT FULL
```

(The F77LINK macro includes the SWATI file automatically, so you do not specify it as an argument.)

Figures C-4, C-5, and C-6 present the compilation listings of the main program, PAYMENT subroutine, and FULL subroutine, respectively. Figure C-7 lists the audit file of a sample SWAT session. The audit file includes comments to describe the debugging activity.

```
Source file: LOAN.F77
Compiled on 24-Nov-81 at 11:32:22 by AOS/VS F77 Rev 01.31.00.02
Options: F77/DEBUG/L=LOAN.LS

    1
    2   C This F77 program computes mortgage payments: summary or full schedule.
    3
    4         double precision  AMOUNT, RATE, PAY
    5
    6         character*10  ANSWER
    7
    8   C  Request the principal, interest rate, and number of years.
    9
   10   5     print *, "Enter principal amount:  $"
   11         read *,  AMOUNT
   12
   13         print *, "Enter interest rate.  % = "
   14         read *,  RATE
   15         RATE = RATE / 100
   16
   17         print *, "Enter the number of years:  "
   18         read *,  IYEARS
   19
   20   C  Call the PAYMENT subroutine, which calculates the monthly payment.
   21
   22         call PAYMENT(AMOUNT, RATE, IYEARS, PAY)
   23
   24   C  Does user want full schedule?
   25
   26         print *, "Do you want a full schedule?  Enter YES or NO:  "
   27         read (*, '(A)')  ANSWER
   28
   29         if ( (ANSWER(1:1) .eq. 'Y') .or. (ANSWER(1:1) .eq. 'y') ) then
   30
   31            call FULL(AMOUNT, RATE, IYEARS, PAY)
   32
   33         end if
   34
   35   C  User can repeat the program or end it.
   36
   37         print *, "Do you want to repeat the program?  Enter YES or NO:  "
   38         read (*, '(A)')   ANSWER          ! Formatted "A" read.
   39         if ( (ANSWER(1:1) .eq. 'Y') .or. (ANSWER(1:1) .eq. 'y') ) go to  5
   40         end
```

Figure C-4. Compilation Listing of the Loan Program (Main Routine) (continues)

```
EXTERNAL ENTRY POINTS

Entry Point Program Unit        Line    Type

.MAIN.                          4       ENTRY


Main Program .MAIN. on line 4

Name        Storage     Size    Loc     Attributes

5           Constant                    Executable Label Line 10
AMOUNT      Automatic   4W      12      REAL*8
RATE        Automatic   4W      16      REAL*8
PAY         Automatic   4W      20      REAL*8
ANSWER      Automatic   10C     26      CHARACTER*10
IYEARS      Automatic   2W      24      INTEGER*4
PAYMENT     Constant                    SUBROUTINE
FULL        Constant                    SUBROUTINE
```

Figure C-4. Compilation Listing of the Loan Program (Main Routine) (concluded)

```
Source file: PAYMENT.F77
Compiled on 24-Nov-81 at 11:32:42 by AOS/VS F77 Rev 01.31.00.02
Options: F77/DEBUG/L=PAYMENT.LS

    1
    2   C This routine computes the monthly payment for the loan.
    3
    4         SUBROUTINE PAYMENT(AMOUNT, RATE, IYEARS, PAY)
    5
    6         double precision  AMOUNT, RATE, R, PAY
    7
    8         R = RATE / 12
    9         MONTHS = IYEARS * 12
   10
   11   C  Calculate the monthly payment and write it to the console.
   12
   13         PAY = AMOUNT*R*(1+R)**MONTHS  / ((1+R)**MONTHS -1)
   14
   15         print 110,  AMOUNT, RATE, IYEARS, PAY
   16   110   format ("0", "Amount     = $", F9.2 ,/, " Interest Rate =",F7.4,/,
   17        X  " Loan Life is ",I3," Years",/, " Monthly Payment = $", F9.2, /)
   18
   19         return
   20         end


EXTERNAL ENTRY POINTS

Entry Point Program Unit        Line    Type

PAYMENT                         4       SUBROUTINE


Subroutine PAYMENT on line 4

Name        Storage     Size    Loc     Attributes

110         Constant                    FORMAT Label Line 16
AMOUNT      Dummy Arg   4W      Pos 1   REAL*8
RATE        Dummy Arg   4W      Pos 2   REAL*8
IYEARS      Dummy Arg   2W      Pos 3   INTEGER*4
PAY         Dummy Arg   4W      Pos 4   REAL*8
R           Automatic   4W      14      REAL*8
MONTHS      Automatic   2W      18      INTEGER*4
```

Figure C-5. Compilation Listing of the Subroutine PAYMENT

```
Source file: FULL.F77
Compiled on 24-Nov-81 at 11:32:59 by AOS/VS F77 Rev 01.31.00.02
Options: F77/DEBUG/L=FULL.LS

    1
    2   C This routine computes the full schedule.
    3
    4         SUBROUTINE FULL(AMOUNT, RATE, IYEARS, PAY)
    5
    6         double precision  AMOUNT, RATE, R, PAY
    7         double precision  BAL, INTEREST, ITD, PRIN
    8
    9         R = RATE / 12             ! Set the monthly rate of interest
   10         MONTHS = IYEARS * 12      ! Set the number of payments
   11
   12   C  Set the initial values for principal, months, and interest-to-date
   13
   14            BAL = AMOUNT           ! Balance starts as the original amount.
   15            MM = MONTHS            ! Save original number of months in MM.
   16            ITD = 0                ! Interest To Date starts as 0.
   17
   18   C   Write summary information to the current LISTFILE.
   19
   20            write (12,110)  AMOUNT, RATE, IYEARS, PAY
   21   110      format ("0", "Amount   = $", F9.2,/," Interest Rate =",F7.4,/
   22        X   " Loan Life is ",I3," Years",/," Monthly Payment = $",F9.2, /)
   23
   24            write (12,120)
   25   120      format (1X," Num",8X,"Interest",6X,"Prin. Pay", 6X,
   26        X   "Prin. Bal.", 4X,"Interest Paid to Date",/)
   27
   28   C  Calculate the interest and principal amounts for each month.
   29
   30            do 200  I = 1, MM              ! DO for all the the months..
   31
   32               PRIN = BAL*R / ((R+1)**MONTHS-1)  ! Calculate principal paid.
   33               MONTHS = MONTHS - 1         ! Decrement month.
   34               INTEREST = PAY - PRIN       ! Calculate interest paid.
   35               BAL = BAL - PRIN            ! Update loan balance.
   36               ITD = ITD + INTEREST        ! Update interest paid to date.
   37
   38               write (12,130) I, INTEREST, PRIN, BAL, ITD
   39   130         format (1X,I3, 7X,"$",F9.2, 5X,"$",F9.2, 5X,"$",F9.2,
   40        X      8X,"$",F9.2,/)
   41
   42   200      continue
   43
   44         return
   45         end
```

Figure C-6. Compilation Listing of the Subroutine FULL (continues)

```
EXTERNAL ENTRY POINTS

Entry Point Program Unit        Line    Type

FULL                            4       SUBROUTINE


Subroutine FULL on line 4

Name        Storage     Size    Loc     Attributes

110         Constant                    FORMAT Label Line 21
120         Constant                    FORMAT Label Line 25
200         Constant                    Executable Label Line 42
130         Constant                    FORMAT Label Line 39
AMOUNT      Dummy Arg   4W      Pos 1   REAL*8
RATE        Dummy Arg   4W      Pos 2   REAL*8
IYEARS      Dummy Arg   2W      Pos 3   INTEGER*4
PAY         Dummy Arg   4W      Pos 4   REAL*8
R           Automatic   4W      14      REAL*8
BAL         Automatic   4W      18      REAL*8
INTEREST    Automatic   4W      22      REAL*8
ITD         Automatic   4W      26      REAL*8
PRIN        Automatic   4W      30      REAL*8
MONTHS      Automatic   2W      34      INTEGER*4
MM          Automatic   2W      36      INTEGER*4
I           Automatic   2W      38      INTEGER*4
```

Figure C-6. Compilation Listing of the Subroutine FULL (concluded)

```
------------------------------------------------------------------
USER PROGRAM loan SWAT AUDIT ON 11/24/81 AT 08:19:14

AOS/VS SWAT Revision 02.00.00.00 ON 11/24/81 AT 08:19:15
PROGRAM -- :UDD:TOM:LOAN
> % This session illustrates the use of SWAT commands
> % with a program written in FORTRAN 77.
> %
> % When invoking the SWAT debugger, we used the /AUDIT switch
> % to open the audit file LOAN.AU.
> %
> % Confirm that auditing is on.
> AUDIT
ON
> % Display the working environment, which should be the main procedure.
> ENVIRONMENT
:.MAIN
> % List the source code off this module.
```

Figure C-7. Audit File of the LOAN Program SWAT Session (continues)

```
> LIST @ALL
1
2       C This F77 program computes mortgage payments: summary or full schedule.
3
4             double precision  AMOUNT, RATE, PAY
5
6             character*10  ANSWER
7
8       C  Request the principal, interest rate, and number of years.
9
10      5     print *, "Enter principal amount:  $"
11            read *,  AMOUNT
12
13            print *, "Enter interest rate.  % = "
14            read *,  RATE
15            RATE = RATE / 100
16
17            print *, "Enter the number of years:  "
18            read *,  IYEARS
19
20      C  Call the PAYMENT subroutine, which calculates the monthly payment.
21
22            call PAYMENT(AMOUNT, RATE, IYEARS, PAY)
23
24      C  Does user want full schedule?
25
26            print *, "Do you want a full schedule?  Enter YES or NO:  "
27            read (*, '(A)')  ANSWER
28
29            if ( (ANSWER(1:1) .eq. 'Y') .or. (ANSWER(1:1) .eq. 'y') ) then
30
31               call FULL(AMOUNT, RATE, IYEARS, PAY)
32
33            end if
34
35      C  User can repeat the program or end it.
36
37            print *, "Do you want to repeat the program?  Enter YES or NO:  "
38            read (*, '(A)')   ANSWER          ! Formatted "A" read.
39            if ( (ANSWER(1:1) .eq. 'Y') .or. (ANSWER(1:1) .eq. 'y') ) go to  5
40            end
> % Display the working directory.
> DIRECTORY
:UDD:TOM
> % Display the current search list.
> SEARCHLIST
:UDD:TOM :UTIL :SWAT :LANG_RT :F77
> % Obtain information about program symbols: constants and variables.
> DESCRIBE AMOUNT
AMOUNT (4 Words at +14R8 Words) REAL*8
```

Figure C-7. Audit File of the LOAN Program SWAT Session (continued)

```
> DESCRIBE ANSWER
ANSWER (10 Characters at +32R8 Words) CHARACTER*10
> DESCRIBE 5
5 Executable label  line 10 at 16001770642R8 Bits
> % Set breakpoints at key statements.
> % First, at the label 5, which is line 10.
> BREAKPOINT,*5
Set at :.MAIN:10
> % Then at other line numbers.
> BREAKPOINT 15 22 29 31 39
Set at :.MAIN:15
Set at :.MAIN:22
Set at :.MAIN:29
Set at :.MAIN:31
Set at :.MAIN:39
> % Display the breakpoints currently in effect.
> BREAKPOINT
Set at :.MAIN:10
Set at :.MAIN:15
Set at :.MAIN:22
Set at :.MAIN:29
Set at :.MAIN:31
Set at :.MAIN:39
> % Set the environment to that of the routine called PAYMENT.
> ENVIRONMENT :PAYMENT
:PAYMENT
> % List the source code for this module.
> LIST @ALL
1
2       C This routine computes the monthly payment for the loan.
3
4             SUBROUTINE PAYMENT(AMOUNT, RATE, IYEARS, PAY)
5
6             double precision  AMOUNT, RATE, R, PAY
7
8             R = RATE / 12
9             MONTHS = IYEARS * 12
10
11      C  Calculate the monthly payment and write it to the console.
12
13            PAY = AMOUNT*R*(1+R)**MONTHS  / ((1+R)**MONTHS -1)
14
15            print 110,  AMOUNT, RATE, IYEARS, PAY
16      110   format ("0", "Amount    = $", F9.2 ,/, " Interest Rate =",F7.4,/,
17           X  " Loan Life is ",I3," Years",/, " Monthly Payment = $", F9.2, /)
18
19            return
20            end
> % Obtain information about symbols in this routine.
```

Figure C-7. Audit File of the LOAN Program SWAT Session (continued)

```
> DESCRIBE R
R (4 Words at +16R8 Words) REAL*8
> DESCRIBE MONTHS
MONTHS (2 Words at +22R8 Words) INTEGER*4
> DESCRIBE IYEARS
IYEARS (2 Words) INTEGER*4 Parameter
> % Set breakpoints in this routine.
> BREAKPOINT 13 19
Set at :PAYMENT:13
Set at :PAYMENT:19
> % Display the breakpoints set so far.
> BREAKPOINT
Set at :.MAIN:10
Set at :.MAIN:15
Set at :.MAIN:22
Set at :.MAIN:29
Set at :.MAIN:31
Set at :.MAIN:39
Set at :PAYMENT:13
Set at :PAYMENT:19
> % Set the environment to that of the routine called FULL.
> ENVIRONMENT :FULL
:FULL
> % List the source code for this module.
```

Figure C-7. Audit File of the LOAN Program SWAT Session (continued)

```
> LIST @ALL
 1
 2       C This routine computes the full schedule.
 3
 4             SUBROUTINE FULL(AMOUNT, RATE, IYEARS, PAY)
 5
 6             double precision  AMOUNT, RATE, R, PAY
 7             double precision  BAL, INTEREST, ITD, PRIN
 8
 9             R = RATE / 12             ! Set the monthly rate of interest
10             MONTHS = IYEARS * 12      ! Set the number of payments
11
12       C  Set the initial values for principal, months, and interest-to-date
13
14                BAL = AMOUNT           ! Balance starts as the original amount.
15                MM = MONTHS            ! Save original number of months in MM.
16                ITD = 0                ! Interest To Date starts as 0.
17
18       C   Write summary information to the current LISTFILE.
19
20                write (12,110)  AMOUNT, RATE, IYEARS, PAY
21       110      format ("0", "Amount   = $", F9.2,/," Interest Rate =",F7.4,/
22            X   " Loan Life is ",I3," Years",/," Monthly Payment = $",F9.2, /)
23
24                write (12,120)
25       120      format (1X," Num",8X,"Interest",6X,"Prin. Pay", 6X,
26            X   "Prin. Bal.", 4X,"Interest Paid to Date",/)
27
28       C  Calculate the interest and principal amounts for each month.
29
30                do 200  I = 1, MM              ! DO for all the the months..
31
32                   PRIN = BAL*R / ((R+1)**MONTHS-1)  ! Calculate principal paid.
33                   MONTHS = MONTHS - 1         ! Decrement month.
34                   INTEREST = PAY - PRIN       ! Calculate interest paid.
35                   BAL = BAL - PRIN            ! Update loan balance.
36                   ITD = ITD + INTEREST        ! Update interest paid to date.
37
38                   write (12,130) I, INTEREST, PRIN, BAL, ITD
39       130         format (1X,I3, 7X,"$",F9.2, 5X,"$",F9.2, 5X,"$",F9.2,
40            X      8X,"$",F9.2,/)
41
42       200      continue
43
44             return
45             end
> % Obtain information about symbols in this routine.
> DESCRIBE ITD
ITD (4 Words at +32R8 Words) REAL*8
```

Figure C-7. Audit File of the LOAN Program SWAT Session (continued)

```
> DESCRIBE PAY
PAY (4 Words) REAL*8 Parameter
> DESCRIBE BAL
BAL (4 Words at +22R8 Words) REAL*8
> DESCRIBE 130
 (at 0R8 Words)
> DESCRIBE 200
 (at 0R8 Words)
> % Set breakpoints in this routine.
> BREAKPOINT 20 32 44
Set at :FULL:20
Set at :FULL:32
Set at :FULL:44
> % Display the breakpoints set so far.
> BREAKPOINT
Set at :.MAIN:10
Set at :.MAIN:15
Set at :.MAIN:22
Set at :.MAIN:29
Set at :.MAIN:31
Set at :.MAIN:39
Set at :PAYMENT:13
Set at :PAYMENT:19
Set at :FULL:20
Set at :FULL:32
Set at :FULL:44
> % Return to the main procedure.
> ENVIRONMENT @MAIN
:.MAIN
> % Initiate program execution.
> CONTINUE

Breakpoint trap at :.MAIN:10
> LIST.*5
10C     5     print *, "Enter principal amount:  $"
> CONTINUE

Breakpoint trap at :.MAIN:15
> % The program requested the principal amount: we entered $3000.
> % The program requested the interest rate: we entered 17 (%).
> %
> % Display the stored values.
> TYPE AMOUNT
 3.000000000000000E+03
> TYPE RATE
 1.700000000000000E+01
> LIST 15
15C           RATE = RATE / 100
> TYPE RATE/100
 1.700000000000000E-01
```

Figure C-7. Audit File of the LOAN Program SWAT Session (continued)

```
> CONTINUE

Breakpoint trap at :.MAIN:22
> % The program requested the number of years:  we entered 3.
> TYPE IYEARS
3
> LIST 22
22C           call PAYMENT(AMOUNT, RATE, IYEARS, PAY)
> TYPE PAY
 0.000000000000000E+00
> CONTINUE

Breakpoint trap at :PAYMENT:13
> LIST 13
13C           PAY = AMOUNT*R*(1+R)**MONTHS  / ((1+R)**MONTHS -1)
> TYPE AMOUNT
 3.000000000000000E+03
> TYPE R
 1.416666666666667E-02
> TYPE MONTHS
36
> ENVIRONMENT
:PAYMENT
> CONTINUE

Breakpoint trap at :PAYMENT:19
> % The program reports that our monthly payment is $106.96.
> % We can change the principal amount and recalculate.
> SET AMOUNT = 4000.00
Old value:  3.000000000000000E+03
New value:  4.000000000000000E+03
> CONTINUE AT 9

Breakpoint trap at :PAYMENT:13
> TYPE AMOUNT
 4.000000000000000E+03
> CONTINUE

Breakpoint trap at :PAYMENT:19
>
% Changing the amount from $3000 to $4000. the monthly payment is $142.61.
> CONTINUE

Breakpoint trap at :.MAIN:29
> % The program asked if we wanted a full schedule. we responded no.
```

Figure C-7. Audit File of the LOAN Program SWAT Session (continued)

```
> LIST @BREAK
19
20      C  Call the PAYMENT subroutine, which calculates the monthly payment.
21
22B           call PAYMENT(AMOUNT, RATE, IYEARS, PAY)
23
24      C  Does user want full schedule?
25
26            print *, "Do you want a full schedule?  Enter YES or NO:  "
27            read (*, '(A)')  ANSWER
28
29C           if ( (ANSWER(1:1) .eq. 'Y') .or. (ANSWER(1:1) .eq. 'y') ) then
30
31B              call FULL(AMOUNT, RATE, IYEARS, PAY)
32
33            end if
34
35      C  User can repeat the program or end it.
36
37            print *, "Do you want to repeat the program?  Enter YES or NO:  "
38            read (*, '(A)')   ANSWER          ! Formatted "A" read.
39B           if ( (ANSWER(1:1) .eq. 'Y') .or. (ANSWER(1:1) .eq. 'y') ) go to  5
> TYPE ANSWER
"NO        "
> % We can change our answer.  The program should call the FULL routine then.
> SET ANSWER = "YES"
Old value: "NO        "
New value: "YES       "
> CONTINUE

Breakpoint trap at :.MAIN:31
> LIST 31
31C             call FULL(AMOUNT, RATE, IYEARS, PAY)
> TYPE AMOUNT
 4.000000000000000E+03
> TYPE RATE
 1.700000000000000E-01
> TYPE IYEARS
3
> CONTINUE

Breakpoint trap at :FULL:20
```

Figure C-7. Audit File of the LOAN Program SWAT Session (continued)

```
>
10            MONTHS = IYEARS * 12      ! Set the number of payments
11
12      C  Set the initial values for principal, months, and interest-to-date
13
14              BAL = AMOUNT            ! Balance starts as the original amount.
15              MM = MONTHS             ! Save original number of months in MM.
16              ITD = 0                 ! Interest To Date starts as 0.
17
18      C   Write summary information to the current LISTFILE.
19
20C              write (12,110)  AMOUNT, RATE, IYEARS, PAY
21      110     format ("0", "Amount   = $", F9.2,/," Interest Rate =",F7.4,/
22           X   " Loan Life is ",I3," Years",/," Monthly Payment = $",F9.2, /)
23
24              write (12,120)
25      120     format (1X," Num",8X,"Interest",6X,"Prin. Pay", 6X,
26           X   "Prin. Bal.", 4X,"Interest Paid to Date",/)
27
28      C  Calculate the interest and principal amounts for each month.
29
30              do 200  I = 1, MM               ! DO for all the the months..
> TYPE AMOUNT
 4.000000000000000E+03
> TYPE RATE
 1.700000000000000E-01
> TYPE IYEARS
3
> TYPE PAY
 1.426109101069792E+02
> % We'll skip the full schedule calculation and return to the main program.
> CONTINUE AT 44

Breakpoint trap at :.MAIN:39
> LIST 39
39C           if ( (ANSWER(1:1) .eq. 'Y') .or. (ANSWER(1:1) .eq. 'y') ) go to  5
> % Our answer is No, so the program will end if we continue.
> % To end the debugging session at this point, we can also type:
> BYE

SWAT TERMINATED
```

Figure C-7. Audit File of the LOAN Program SWAT Session (concluded)

A Sample COBOL Program and Audit File

The program (MORTPROG) calculates mortgage payment information. The operator enters the principal amount, the interest rate, and the number of years to pay back. The program then calculates the monthly payment. If the operator requests, the program also calculates a full schedule, which lists the amount of interest and principal paid in each monthly payment, the principal balance, and the interest paid to date.

The compilation and link commands are as follows:

```
COBOL/DEBUG/L/X/A MORTPROG MORTPROG.LS/L

CLINK/DEBUG MORTPROG
```

(The CLINK macro automatically includes the SWATI file; do not specify it as an argument.)

Figure C-8 is the compilation listing. Following the listing is Figure C-9, the audit file of a sample SWAT session. This session illustrates the use of SWAT commands with a COBOL program. It also shows how certain SWAT entries and responses differ when you are working with a COBOL program.

```
Source file: MORTPROG
Compiled on 24-Nov-81 at 11:07:06 by AOS/VS COBOL Rev 01.00.00.00
OPTIONS: COBOL/DEBUG/L/X/A,MORTPROG
    1
    2    IDENTIFICATION DIVISION.
    3    PROGRAM-ID. MORTPROG.
    4    AUTHOR. TEK REITER.
    5    DATE-WRITTEN. 9 NOV 1981.
    6    ENVIRONMENT DIVISION.
    7    CONFIGURATION SECTION.
    8    SOURCE-COMPUTER.  MV-8000.
    9    OBJECT-COMPUTER.  MV-8000.
   10    INPUT-OUTPUT SECTION.
   11    FILE-CONTROL.
   12            SELECT OUTFILE, ASSIGN TO PRINTER.
   13    DATA DIVISION.
   14    FILE SECTION.
   15    FD      OUTFILE, BLOCK CONTAINS 512 CHARACTERS.
   16    01      OUTREC.
   17            02 OUT-PAYMT-NUM       PIC ZZZ9.
   18            02 FILLER              PIC X(6).
   19            02 OUT-MON-INT         PIC $(4)9.99.
   20            02 FILLER              PIC X(3).
   21            02 OUT-MON-PRIN                PIC $(6)9.99.
   22            02 FILLER              PIC X(3).
   23            02 OUT-BALANCE         PIC $(6)9.99.
   24            02 FILLER              PIC X(8).
   25            02 OUT-INT-TO-DATE     PIC $(6)9.99.
   26            02 FILLER              PIC X(10).
   27
   28    WORKING-STORAGE SECTION.
   29    01      CRT-INPUTS.
   30            02 PRINCIPAL           PIC 9(6)V99.
   31            02 PERCENT             PIC 99V99.
   32            02 YEARS               PIC 99.
   33            02 FUNCTION            PIC 9.
   34            02 REPEAT-FLAG         PIC 9.
   35
   36    01      TEMPS.
   37            02 MONTHLY-INT-RATE    USAGE COMP-1.
   38            02 MONTHS              PIC 9(4).
   39            02 MONTHS-LEFT         PIC 9(4).
   40            02 MONTHLY-PAYMT       PIC 9(4)V99.
   41            02 LOAN-BAL            PIC 9(6)V99.
   42            02 INT-TO-DATE         PIC 9(6)V99.
   43            02 PAYMT-NUM           PIC 9(4), USAGE COMP.
   44            02 INT-PAYMT           PIC 9(4)V99.
   45            02 PRIN-PAYMT          PIC 9(4)V99.
   46
```

Figure C-8. Compilation Listing of the MORTPROG Program (continues)

```
47    01      SUMMARY-LINE1.
48            02 FILLER      PIC X(16), VALUE "PRINCIPAL =      ".
49            02 SUMMARY-PRIN. PIC $(6)9.99.
50            02 FILLER      PIC X(50), VALUE SPACES.
51    01      SUMMARY-LINE2.
52            02 FILLER      PIC X(20), VALUE "INTEREST RATE =      ".
53            02 SUMMARY-RATE. PIC 9.9(4).
54            02 FILLER      PIC X(50), VALUE SPACES.
55    01      SUMMARY-LINE3.
56            02 FILLER      PIC X(18), VALUE "LOAN LIFE =       ".
57            02 SUMMARY-YEARS. PIC Z9.
58            02 FILLER      PIC X(6), VALUE " YEARS".
59            02 FILLER      PIC X(50), VALUE SPACES.
60    01      SUMMARY-LINE4.
61            02 FILLER      PIC X(18), VALUE "MONTHLY PAYMENT = ".
62            02 SUMMARY-PAYMT. PIC $(4)9.99.
63            02 FILLER      PIC X(50), VALUE SPACES.
64
65    01      HEADLINE      PIC X(80),
66                VALUE " NUM       INTEREST     PRIN. PAY  PRIN.
67    -               " BAL      INTEREST PAID TO DATE".
68
69    PROCEDURE DIVISION.
70    INIT.  OPEN OUTPUT OUTFILE.
71    OPERATOR.
72            DISPLAY "ENTER PRINCIPAL:  $" WITH NO ADVANCING.
73            ACCEPT PRINCIPAL.
74            DISPLAY "INTEREST RATE (%): " WITH NO ADVANCING.
75            ACCEPT PERCENT.
76            COMPUTE MONTHLY-INT-RATE = PERCENT / 100 / 12.
77            DISPLAY "YEARS TO PAY:       " WITH NO ADVANCING.
78            ACCEPT YEARS.
79            COMPUTE MONTHS = YEARS * 12.
80            DISPLAY "FUNCTION (0=SUMMARY, 1=FULL SCHEDULE): "
81                                    WITH NO ADVANCING.
82            ACCEPT FUNCTION.
83            COMPUTE MONTHLY-PAYMT ROUNDED =
84                  PRINCIPAL * MONTHLY-INT-RATE *
85                        (1 + MONTHLY-INT-RATE) ** MONTHS /
86    *             ----------------------------------------
87                  ((1 + MONTHLY-INT-RATE) ** MONTHS - 1).
88            PERFORM SUMMARY-OUTPUT.
89            IF FUNCTION NOT = 0.
90                  PERFORM DETAIL-OUTPUT.
91            DISPLAY "TYPE 1 TO REPEAT, 0 TO STOP: " WITH NO ADVANCING.
92            ACCEPT REPEAT-FLAG.
93            IF REPEAT-FLAG NOT = 0, GO TO OPERATOR.
94            CLOSE OUTFILE.
95            STOP RUN.
96
```

Figure C-8. Compilation Listing of the MORTPROG Program (continued)

```
 97   SUMMARY-OUTPUT.
 98           MOVE PRINCIPAL TO SUMMARY-PRIN.
 99           WRITE OUTREC FROM SUMMARY-LINE1 BEFORE ADVANCING 1.
100           COMPUTE SUMMARY-RATE = PERCENT / 100.
101           WRITE OUTREC FROM SUMMARY-LINE2 BEFORE ADVANCING 1.
102           MOVE YEARS TO SUMMARY-YEARS.
103           WRITE OUTREC FROM SUMMARY-LINE3 BEFORE ADVANCING 2.
104           MOVE MONTHLY-PAYMT TO SUMMARY-PAYMT.
105           WRITE OUTREC FROM SUMMARY-LINE4 BEFORE ADVANCING 2.
106
107   DETAIL-OUTPUT.
108           MOVE PRINCIPAL TO LOAN-BAL.
109           MOVE MONTHS TO MONTHS-LEFT.
110           MOVE 0 TO INT-TO-DATE.
111           WRITE OUTREC FROM HEADLINE BEFORE ADVANCING 2.
112           MOVE SPACES TO OUTREC.
113           PERFORM DO-DETAIL-LINE
114                 VARYING PAYMT-NUM FROM 1 BY 1
115                       UNTIL PAYMT-NUM > MONTHS.
116
117   DO-DETAIL-LINE.
118           COMPUTE PRIN-PAYMT ROUNDED =
119                 LOAN-BAL * MONTHLY-INT-RATE /
120   *             ------------------------------------------
121                 ((1 + MONTHLY-INT-RATE) ** MONTHS-LEFT - 1).
122           SUBTRACT 1 FROM MONTHS-LEFT.
123           COMPUTE INT-PAYMT = MONTHLY-PAYMT - PRIN-PAYMT.
124           SUBTRACT PRIN-PAYMT FROM LOAN-BAL.
125           ADD INT-PAYMT TO INT-TO-DATE.
126           MOVE PAYMT-NUM TO OUT-PAYMT-NUM.
127           MOVE INT-PAYMT TO OUT-MON-INT.
128           MOVE PRIN-PAYMT TO OUT-MON-PRIN.
129           MOVE LOAN-BAL TO OUT-BALANCE.
130           MOVE INT-TO-DATE TO OUT-INT-TO-DATE.
131           WRITE OUTREC BEFORE ADVANCING 2.
132
133   *       END OF PROGRAM
134
```

Figure C-8. Compilation Listing of the MORTPROG Program (continued)

```
NAME                      DATA TYPE        LOCATION      SIZE ATTRIBUTES

MORTPROG                  Program-id          0*          1W EX
OUTREC                    Alphanumeric        0B         72B S
OUT-PAYMT-NUM             Numeric-edit        0B          4B M
OUT-MON-INT               Numeric-edit       10B          8B M
OUT-MON-PRIN              Numeric-edit       21B         10B M
OUT-BALANCE               Numeric-edit       34B         10B M
OUT-INT-TO-DATE           Numeric-edit       52B         10B M
CRT-INPUTS                Alphanumeric       72B         16B S
PRINCIPAL                 Ascii no sign      72B          8B MI
PERCENT                   Ascii no sign      80B          4B MI
YEARS                     Ascii no sign      84B          2B MI
FUNCTION                  Ascii no sign      86B          1B MI
REPEAT-FLAG               Ascii no sign      87B          1B MI
TEMPS                     Alphanumeric       88B         52B S
MONTHLY-INT-RATE          Comp-2 Fl Pnt      88B          8B MI
MONTHS                    Ascii no sign      96B          4B MI
MONTHS-LEFT               Ascii no sign     100B          4B MI
MONTHLY-PAYMT             Ascii no sign     104B          6B MI
LOAN-BAL                  Ascii no sign     110B          8B MI
INT-TO-DATE               Ascii no sign     118B          8B MI
PAYMT-NUM                 Comp - Binary     126B          2B MI
INT-PAYMT                 Ascii no sign     128B          6B MI
PRIN-PAYMT                Ascii no sign     134B          6B MI
SUMMARY-LINE1             Alphanumeric      140B         76B S
SUMMARY-PRIN              Numeric-edit      156B         10B M
SUMMARY-LINE2             Alphanumeric      216B         76B S
SUMMARY-RATE              Numeric-edit      236B          6B M
SUMMARY-LINE3             Alphanumeric      292B         76B S
SUMMARY-YEARS             Numeric-edit      310B          2B M
SUMMARY-LINE4             Alphanumeric      368B         76B S
SUMMARY-PAYMT             Numeric-edit      386B          8B M
HEADLINE                  Alphanumeric      444B         80B I

      COMPILER STATISTICS
Total elapsed time in seconds:      17
Total cpu time in seconds:           4
Number of lines compiled:          134
Lines per elapsed minute:          472
Lines per cpu minute:             1704
```

Figure C-8. Compilation Listing of the MORTPROG Program (concluded)

```
------------------------------------------------------------
USER PROGRAM mortprog SWAT AUDIT ON 11/24/81 AT 11:10:26

AOS/VS SWAT Revision 02.00.00.00 ON 11/24/81 AT 11:10:29
PROGRAM -- :UDD:TOM:MORTPROG
> % This session illustrates the use of SWAT commands
> % with a program written in COBOL.
> %
> % When invoking the SWAT debugger, we used the /AUDIT switch
> % to open the audit file MORT.AU.
> %
> % Confirm that auditing is on.
> AUDIT
ON
> % Display the working environment.
> ENVIRONMENT
:MORTPROG
> % List the source code.
> LIST @ALL
2       IDENTIFICATION DIVISION.
3       PROGRAM-ID. MORTPROG.
4       AUTHOR. TEK REITER.
5       DATE-WRITTEN. 9 NOV 1981.
6       ENVIRONMENT DIVISION.
7       CONFIGURATION SECTION.
8       SOURCE-COMPUTER.  MV-8000.
9       OBJECT-COMPUTER.  MV-8000.
10      INPUT-OUTPUT SECTION.
11      FILE-CONTROL.
12              SELECT OUTFILE, ASSIGN TO PRINTER.
13      DATA DIVISION.
14      FILE SECTION.
15      FD      OUTFILE, BLOCK CONTAINS 512 CHARACTERS.
16      01      OUTREC.
17              02 OUT-PAYMT-NUM        PIC ZZZ9.
18              02 FILLER               PIC X(6).
19              02 OUT-MON-INT          PIC $(4)9.99.
20              02 FILLER               PIC X(3).
21              02 OUT-MON-PRIN         PIC $(6)9.99.
22              02 FILLER               PIC X(3).
23              02 OUT-BALANCE          PIC $(6)9.99.
24              02 FILLER               PIC X(8).
25              02 OUT-INT-TO-DATE      PIC $(6)9.99.
26              02 FILLER               PIC X(10).
27
28      WORKING-STORAGE SECTION.
29      01      CRT-INPUTS.
30              02 PRINCIPAL            PIC 9(6)V99.
31              02 PERCENT              PIC 99V99.
32              02 YEARS                PIC 99.
33              02 FUNCTION             PIC 9.
34              02 REPEAT-FLAG          PIC 9.
```

Figure C-9. Audit File of the MORTPROG SWAT Session (continues)

```
35
36      01      TEMPS.
37              02 MONTHLY-INT-RATE     USAGE COMP-1.
38              02 MONTHS               PIC 9(4).
39              02 MONTHS-LEFT          PIC 9(4).
40              02 MONTHLY-PAYMT        PIC 9(4)V99.
41              02 LOAN-BAL             PIC 9(6)V99.
42              02 INT-TO-DATE          PIC 9(6)V99.
43              02 PAYMT-NUM            PIC 9(4), USAGE COMP.
44              02 INT-PAYMT            PIC 9(4)V99.
45              02 PRIN-PAYMT           PIC 9(4)V99.
46
47      01      SUMMARY-LINE1.
48              02 FILLER       PIC X(16), VALUE "PRINCIPAL =      ".
49              02 SUMMARY-PRIN. PIC $(6)9.99.
50              02 FILLER       PIC X(50), VALUE SPACES.
51      01      SUMMARY-LINE2.
52              02 FILLER       PIC X(20), VALUE "INTEREST RATE =      ".
53              02 SUMMARY-RATE. PIC 9.9(4).
54              02 FILLER       PIC X(50), VALUE SPACES.
55      01      SUMMARY-LINE3.
56              02 FILLER       PIC X(18), VALUE "LOAN LIFE =        ".
57              02 SUMMARY-YEARS. PIC Z9.
58              02 FILLER       PIC X(6), VALUE " YEARS".
59              02 FILLER       PIC X(50), VALUE SPACES.
60      01      SUMMARY-LINE4.
61              02 FILLER       PIC X(18), VALUE "MONTHLY PAYMENT = ".
62              02 SUMMARY-PAYMT. PIC $(4)9.99.
63              02 FILLER       PIC X(50), VALUE SPACES.
64
65      01      HEADLINE       PIC X(80),
66                  VALUE " NUM       INTEREST     PRIN. PAY  PRIN.
67      -                 " BAL      INTEREST PAID TO DATE".
68
69      PROCEDURE DIVISION.
70      INIT.   OPEN OUTPUT OUTFILE.
71      OPERATOR.
72              DISPLAY "ENTER PRINCIPAL:  $" WITH NO ADVANCING.
73              ACCEPT PRINCIPAL.
74              DISPLAY "INTEREST RATE (%): " WITH NO ADVANCING.
75              ACCEPT PERCENT.
76              COMPUTE MONTHLY-INT-RATE = PERCENT / 100 / 12.
77              DISPLAY "YEARS TO PAY:       " WITH NO ADVANCING.
78              ACCEPT YEARS.
79              COMPUTE MONTHS = YEARS * 12.
80              DISPLAY "FUNCTION (0=SUMMARY, 1=FULL SCHEDULE): "
81                                      WITH NO ADVANCING.
82              ACCEPT FUNCTION.
83              COMPUTE MONTHLY-PAYMT ROUNDED =
84                      PRINCIPAL * MONTHLY-INT-RATE *
85                              (1 + MONTHLY-INT-RATE) ** MONTHS /
```

Figure C-9. Audit File of the MORTPROG SWAT Session (continued)

```
86      *               ----------------------------------------
87                      ((1 + MONTHLY-INT-RATE) ** MONTHS - 1).
88              PERFORM SUMMARY-OUTPUT.
89              IF FUNCTION NOT = 0,
90                      PERFORM DETAIL-OUTPUT.
91              DISPLAY "TYPE 1 TO REPEAT, 0 TO STOP: " WITH NO ADVANCING.
92              ACCEPT REPEAT-FLAG.
93              IF REPEAT-FLAG NOT = 0, GO TO OPERATOR.
94              CLOSE OUTFILE.
95              STOP RUN.
96
97      SUMMARY-OUTPUT.
98              MOVE PRINCIPAL TO SUMMARY-PRIN.
99              WRITE OUTREC FROM SUMMARY-LINE1 BEFORE ADVANCING 1.
100             COMPUTE SUMMARY-RATE = PERCENT / 100.
101             WRITE OUTREC FROM SUMMARY-LINE2 BEFORE ADVANCING 1.
102             MOVE YEARS TO SUMMARY-YEARS.
103             WRITE OUTREC FROM SUMMARY-LINE3 BEFORE ADVANCING 2.
104             MOVE MONTHLY-PAYMT TO SUMMARY-PAYMT.
105             WRITE OUTREC FROM SUMMARY-LINE4 BEFORE ADVANCING 2.
106
107     DETAIL-OUTPUT.
108             MOVE PRINCIPAL TO LOAN-BAL.
109             MOVE MONTHS TO MONTHS-LEFT.
110             MOVE 0 TO INT-TO-DATE.
111             WRITE OUTREC FROM HEADLINE BEFORE ADVANCING 2.
112             MOVE SPACES TO OUTREC.
113             PERFORM DO-DETAIL-LINE
114                     VARYING PAYMT-NUM FROM 1 BY 1
115                             UNTIL PAYMT-NUM > MONTHS.
116
117     DO-DETAIL-LINE.
118             COMPUTE PRIN-PAYMT ROUNDED =
119                     LOAN-BAL * MONTHLY-INT-RATE /
120     *               ----------------------------------------
121                     ((1 + MONTHLY-INT-RATE) ** MONTHS-LEFT - 1).
122             SUBTRACT 1 FROM MONTHS-LEFT.
123             COMPUTE INT-PAYMT = MONTHLY-PAYMT - PRIN-PAYMT.
124             SUBTRACT PRIN-PAYMT FROM LOAN-BAL.
125             ADD INT-PAYMT TO INT-TO-DATE.
126             MOVE PAYMT-NUM TO OUT-PAYMT-NUM.
127             MOVE INT-PAYMT TO OUT-MON-INT.
128             MOVE PRIN-PAYMT TO OUT-MON-PRIN.
129             MOVE LOAN-BAL TO OUT-BALANCE.
130             MOVE INT-TO-DATE TO OUT-INT-TO-DATE.
131             WRITE OUTREC BEFORE ADVANCING 2.
> % Obtain information about program symbols:  constants and variables.
```

Figure C-9. Audit File of the MORTPROG SWAT Session (continued)

```
> DESCRIBE TEMPS
TEMPS (52 bytes at 34000044400R8 bytes) DISPLAY
    2 MONTHLY-INT-RATE (8 bytes at +34000044400R8 bytes) COMPUTATIONAL-1
    2 MONTHS (4 bytes at +34000044410R8 bytes) DISPLAY PICTURE 9(4)
    2 MONTHS-LEFT (4 bytes at +34000044414R8 bytes) DISPLAY PICTURE 9(4)
    2 MONTHLY-PAYMT (6 bytes at +34000044420R8 bytes) DISPLAY PICTURE 9(4)V9(2)
    2 LOAN-BAL (8 bytes at +34000044426R8 bytes) DISPLAY PICTURE 9(6)V9(2)
    2 INT-TO-DATE (8 bytes at +34000044436R8 bytes) DISPLAY PICTURE 9(6)V9(2)
    2 PAYMT-NUM (2 bytes at +34000044446R8 bytes) COMPUTATIONAL PICTURE 9(4)
    2 INT-PAYMT (6 bytes at +34000044450R8 bytes) DISPLAY PICTURE 9(4)V9(2)
    2 PRIN-PAYMT (6 bytes at +34000044456R8 bytes) DISPLAY PICTURE 9(4)V9(2)
> DESCRIBE HEADLINE
77 HEADLINE (80 bytes at 34000045144R8 bytes) DISPLAY PICTURE X(80)
> DESCRIBE CRT-INPUTS.YEARS
YEARS (2 bytes at +34000044374R8 bytes) DISPLAY PICTURE 9(2)
> DESCRIBE TEMPS.INT-TO-DATE
INT-TO-DATE (8 bytes at +34000044436R8 bytes) DISPLAY PICTURE 9(6)V9(2)
> DESCRIBE SUMMARY-LINE4
SUMMARY-LINE4 (76 bytes at 34000045030R8 bytes) DISPLAY
    2  (18 bytes at +34000045030R8 bytes) DISPLAY PICTURE X(18)
    2 SUMMARY-PAYMT (8 bytes at +34000045052R8 bytes) DISPLAY PICTURE EDITED
    2  (50 bytes at +34000045062R8 bytes) DISPLAY PICTURE X(50)
> DESCRIBE OPERATOR
OPERATOR (at 16001724556R8 DEBUGs) LABEL
> DESCRIBE DO-DETAIL-LINE
DO-DETAIL-LINE (at 16001727033R8 DEBUGs) LABEL
> % Set breakpoints at key statements.
> BREAKPOINT 69
Set at :MORTPROG:69
> BREAKPOINT 83 88 93 105 131
Set at :MORTPROG:83
Set at :MORTPROG:88
Set at :MORTPROG:93
Set at :MORTPROG:105
Set at :MORTPROG:131
> % List the current breakpoints.
> BREAKPOINT
Set at :MORTPROG:69
Set at :MORTPROG:83
Set at :MORTPROG:88
Set at :MORTPROG:93
Set at :MORTPROG:105
Set at :MORTPROG:131
> % Begin program execution.
> CONTINUE

Breakpoint trap at :MORTPROG:69
> % Display the current null command response.
> PROMPT
"LIST @BREAK"
```

Figure C-9. Audit File of the MORTPROG SWAT Session (continued)

```
> % Use the default null command response to list the lines around
> % the current breakpoint trap.
>
59              02 FILLER       PIC X(50), VALUE SPACES.
60      01      SUMMARY-LINE4.
61              02 FILLER       PIC X(18), VALUE "MONTHLY PAYMENT = ".
62              02 SUMMARY-PAYMT, PIC $(4)9.99.
63              02 FILLER       PIC X(50), VALUE SPACES.
64
65      01      HEADLINE        PIC X(80),
66                  VALUE " NUM       INTEREST     PRIN. PAY  PRIN.
67      -                 " BAL      INTEREST PAID TO DATE".
68
69C     PROCEDURE DIVISION.
70      INIT.   OPEN OUTPUT OUTFILE.
71      OPERATOR.
72              DISPLAY "ENTER PRINCIPAL:  $" WITH NO ADVANCING.
73              ACCEPT PRINCIPAL.
74              DISPLAY "INTEREST RATE (%): " WITH NO ADVANCING.
75              ACCEPT PERCENT.
76              COMPUTE MONTHLY-INT-RATE = PERCENT / 100 / 12.
77              DISPLAY "YEARS TO PAY:      " WITH NO ADVANCING.
78              ACCEPT YEARS.
79              COMPUTE MONTHS = YEARS * 12.
> % Skip the next statement (omit opening the printer) and continue execution
> % at statement 72.
> CONTINUE AT 72

Breakpoint trap at :MORTPROG:83
> % The program asked for principal:  we entered $4000
> % The program asked for interest rate:  we entered 17%
> % The program asked for years to pay:  we entered 3
> % We then indicated that we wanted a summary, not a full schedule.
> %
> % List the line where execution trapped.
> LIST 83
83C             COMPUTE MONTHLY-PAYMT ROUNDED =
> % Display the values of program variables.
> TYPE CRT-INPUTS.PRINCIPAL
4000.
> TYPE CRT-INPUTS.PERCENT
17.
> TYPE CRT-INPUTS.YEARS
3.
> TYPE CRT-INPUTS.YEARS * 12.
 3.600000000000000E+01
> TYPE TEMPS.MONTHS
36.
> TYPE CRT-INPUTS.FUNCTION
0.
```

Figure C-9. Audit File of the MORTPROG SWAT Session (continued)

```
> % Change the principal amount to $3000.
> SET CRT-INPUTS.PRINCIPAL = 3000.
Old value: 4000.
New Value: 3000.0
> % Display computed monthly interest rate.
> TYPE TEMPS.MONTHLY-INT-RATE
 1.416666E-02
> % Change the interest rate and recompute.
> SET CRT-INPUTS.PERCENT = 15
Old value: 17.
New Value: 15.
> CONTINUE AT 76

Breakpoint trap at :MORTPROG:83
> TYPE TEMPS.MONTHLY-INT-RATE
 1.250000E-02
> % Continue execution to perform the monthly payment calculation.
> CONTINUE

Breakpoint trap at :MORTPROG:88
> TYPE TEMPS.MONTHLY-PAYMT
.08
> TYPE TEMPS.MONTHLY-INT-RATE
 1.250000E-02
> TYPE CRT-INPUTS.PRINCIPAL
3000.
> TYPE CRT-INPUTS.PERCENT
15.
> TYPE CRT-INPUTS.YEARS
4.
> TYPE CRT-INPUTS.PRINCIPAL * TEMPS.MONTHLY-INT-RATE
 3.750000E+01
> % Display the working directory.
> DIRECTORY
:UDD:TOM
5 % Display the current search list.
> SEARCHLIST
:UDD:TOM :UTIL :SWAT :LANG_RT :COBOL
> % End the debugging session.
> BYE

SWAT TERMINATED
```

Figure C-9. Audit File of the MORTPROG SWAT Session (concluded)

A Sample PASCAL Program and Audit File

This PASCAL program, called EXCHANGEP, is similar to the sample PL/I program used in Chapter 3 of this manual. It calculates currency exchange values based on daily exchange rates. The program consists of a single module, which interacts with the operator and a file called RATE_FILE.

The compilation and link commands are:

```
PASCAL/DEBUG/L=EXCHANGEP.LS EXCHANGEP
```

```
PASLINK/DEBUG EXCHANGEP
```

(The PASLINK macro includes the SWATI file automatically; you do not specify it as an argument.)

Figure C-10 presents the compilation listing of the EXCHANGEP program. A sample SWAT session audit file appears in Figure C-11.

```
Source file: EXCHANGEP
Compiled on 13-Sep-82 at 10:42:27 by AOS/VS PASCAL Rev 01.00.00.00
Options: PASCAL/DEBUG/L=EXCHANGEP.LS

    1   PROGRAM
    2           exchange (INPUT, OUTPUT, rate_file);
    3
    4   {       This program calculates currency exchange values based on
    5   *       daily international exchange rates.  The operator selects
    6   *       the type of exchange, then enters the incoming amount.  The
    7   *       program performs the calculation and displays the result.    }
    8
    9   {$C+    Compiler directive allows ASCII '<nn>' notation in the program }
   10
   11   CONST
   12           TAB = '<11>';
   13           CLEAR_SCREEN = '<14>';
   14
   15   TYPE
   16           country = (belgian,
   17                      w_german,
   18                      dutch,
   19                      italian,
   20                      swiss,
   21                      canadian,
   22                      saudi,
   23                      japanese,
   24                      no_choice);
   25
   26           monetary_symbols = PACKED ARRAY [1..4] OF CHAR;
   27           ratedata = RECORD
   28                       rate_date : PACKED ARRAY [1..6] OF CHAR;
   29                       rates : RECORD
   30                            us_foreign : ARRAY[country] OF REAL;
   31                            foreign_us : ARRAY[country] OF REAL;
   32                              END; { rates }
   33                       END; { ratedata }
   34
   35           rfile = FILE OF ratedata;
   36
   37   VAR
   38
   39           selection : country;
   40           currency : ARRAY [country] OF monetary_symbols;
   41           us_to_foreign : BOOLEAN;
   42           rate_file : rfile;
   43           daily_rates : ratedata;
   44           i : INTEGER;
   45           response : CHAR;
   46
   47   {-------------------------------------------------------------------}
```

Figure C-10. Compilation Listing of the EXCHANGEP Program (continues)

```
48  PROCEDURE
49          builder;
50
51  {       This procedure builds the currency rate file for the exchange
52  *       program.    }
53
54  VAR
55          selection : country;
56          rate_rec  : ratedata;
57
58  BEGIN
59          REWRITE (rate_file);
60
61          WRITELN (CLEAR_SCREEN);
62          WRITELN (TAB, 'Building new currency rate file');
63          WRITELN;
64          WRITE (TAB, 'Enter the date in yymmdd format:  ');
65          READLN (rate_rec.rate_date);
66
67          selection := belgian;
68          REPEAT
69                  WRITE (TAB, 'Enter rate for US$ to ',
70                                  currency[selection], ' :  ');
71                  READLN (rate_rec.rates.us_foreign[selection]);
72
73                  WRITE (TAB, 'Enter rate for ' , currency[selection],
74                                  ' to US$ :  ');
75                  READLN (rate_rec.rates.foreign_us[selection]);
76                  selection := SUCC(selection);
77
78          UNTIL (selection = no_choice);
79
80          WRITE (rate_file, rate_rec);
81
82          RESET (rate_file);
83          READ (rate_file, daily_rates);
84  END;
85  {-------------------------------------------------------------------}
86  PROCEDURE
87          menu;
88
89  BEGIN
90          WITH daily_rates DO
91          BEGIN
92          WRITE ('Rates for ');
93          WRITE (rate_date[3], rate_date[4],'/');
94          WRITE (rate_date[5], rate_date[6],'/');
95          WRITE (rate_date[1], rate_date[2]);
```

Figure C-10. Compilation Listing of the EXCHANGEP Program (continued)

```
 96          WRITELN;
 97          WRITELN ('Identify the type of currency:');
 98          WRITELN;
 99          WRITELN (TAB, '1   Belgian francs');
100          WRITELN (TAB, '2   W. German marks');
101          WRITELN (TAB, '3   Dutch guilders');
102          WRITELN (TAB, '4   Italian lire');
103          WRITELN (TAB, '5   Swiss francs');
104          WRITELN (TAB, '6   Canadian dollars');
105          WRITELN (TAB, '7   Saudi riyals');
106          WRITELN (TAB, '8   Japanese yen');
107          WRITELN (TAB, '(Type 0 to end the program.)');
108          WRITELN;
109          WRITE ('Enter the currency code:  ');
110          END;
111
112  END;
113  {-------------------------------------------------------------------------}
114  PROCEDURE
115          get_selection ( VAR selection : country );
116
117  {       Gets the users currency code input and sets parameter selection
118  *       to the corresponding country.                                     }
119
120  VAR
121          currency_code : INTEGER;
122
123  BEGIN
124
125          READLN (currency_code);
126
127          IF (currency_code >= 0) AND (currency_code <= 8) THEN
128
129                  BEGIN
130
131                  CASE currency_code OF
132
133                          0 : selection := no_choice;
134                          1 : selection := belgian;
135                          2 : selection := w_german;
136                          3 : selection := dutch;
137                          4 : selection := italian;
138                          5 : selection := swiss;
139                          6 : selection := canadian;
140                          7 : selection := saudi;
141                          8 : selection := japanese;
142                  END
143                  END
144
145          ELSE    selection := no_choice;
146
147   END; { get_selection }
```

Figure C-10. Compilation Listing of the EXCHANGEP Program (continued)

```
148
149  {-------------------------------------------------------------------}
150
151  PROCEDURE
152          conversion ( selection : country );
153
154  VAR
155          inval, outval, rate : REAL;
156          response            : CHAR;
157
158  BEGIN
159          WRITELN;
160          WRITELN (TAB, 'Do you want to convert US$ into Foreign currency?');
161          WRITE (TAB, 'Enter "Y" or "N":  ');
162          READLN (response);
163
164          IF ((response = 'Y') OR (response = 'y')) THEN
165                  BEGIN
166                          us_to_foreign := TRUE;
167                          rate := daily_rates.rates.us_foreign[selection];
168                  END
169
170          ELSE
171                  BEGIN
172                          us_to_foreign := FALSE;
173                          rate := daily_rates.rates.foreign_us[selection];
174                  END;
175
176          WRITELN;
177
178          IF us_to_foreign THEN WRITE (TAB, 'Enter US$:  ')
179          ELSE WRITE (TAB, 'Enter ', currency[selection], ':  ');
180
181          READLN (inval);
182          outval := inval * rate;
183
184          WRITELN;
185          IF us_to_foreign THEN WRITELN ('$', inval:10:2,
186            ' US equivalent to:  ', outval:10:2, ' ', currency[selection])
187          ELSE WRITELN (inval:10:2, ' ', currency[selection],
188            ' equivalent to:  $', outval:10:2, ' US');
189
190   END;
191  {-------------------------------------------------------------------}
192
193
194  BEGIN { main procedure }
195
196          currency[belgian] :=  'BF';
197          currency[w_german] :=  'DM';
198          currency[dutch] :=  'gld';
199          currency[italian] :=  'Lit';
```

Figure C-10. Compilation Listing of the EXCHANGEP Program (continued)

```
200         currency[swiss] :=  'SF';
201         currency[canadian] :=  'Can$';
202         currency[saudi] :=  'SRI';
203         currency[japanese] :=  'Y';
204
205         WRITELN (CLEAR_SCREEN, TAB, TAB, 'Exchange Program');
206         WRITELN;
207
208         WRITELN (TAB, 'Do you want to create a new data file?');
209         WRITE (TAB, 'Enter "Y" or "N":  ');
210         READLN (response);
211
212         IF ((response = 'Y') OR (response = 'y')) THEN builder
213         ELSE
214         BEGIN
215
216                 RESET (rate_file);
217                 READ (rate_file, daily_rates);
218
219                 WRITELN;
220                 WRITE (TAB, 'The file contains exchange rates for:  ');
221                 WRITELN (daily_rates.rate_date);
222
223                 WRITE (TAB,
224                        'Do you want to change it? Enter "Y" or "N":  ');
225                 READLN (response);
226
227                 IF ((response = 'Y') OR (response = 'y')) THEN builder;
228         END;
229
230         WRITELN(CLEAR_SCREEN);
231
232         menu;
233         get_selection (selection);
234
235         WHILE (selection <> no_choice) DO
236
237         BEGIN
238                 conversion (selection);
239
240                 WRITELN;
241                 WRITE (TAB, 'Press New Line to continue');
242                 READLN;
243                 WRITELN (CLEAR_SCREEN);
244
245                 menu;
246                 get_selection (selection);
247         END;
248
249 WRITELN (CLEAR_SCREEN, TAB, 'Exchange program ended.');
250
251 END {EXCHANGE program}.
```

Figure C-10. Compilation Listing of the EXCHANGEP Program (continued)

```
Source file: EXCHANGEP
Compiled on 13-Sep-82 at 10:42:35 by AOS/VS PASCAL Rev 01.00.00.00

                        DECLARATIONS


PROGRAM EXCHANGE ON LINE 1


VARIABLE ALLOCATION
NAME            SCOPE          LINE  SIZE   LOC   TYPE

CURRENCY        GLOBAL           40   36C   50W   ARRAY[COUNTRY] OF
                                                    MONETARY_SYMBOLS
DAILY_RATES     GLOBAL           43   39W   4W    RATEDATA
I               GLOBAL           44   2W    2W    INTEGER
RATE_FILE       GLOBAL           42   4W    44W   RFILE
RESPONSE        GLOBAL           45   1C    0W    CHAR
SELECTION       GLOBAL           39   1W    68W   COUNTRY
US_TO_FOREIGN   GLOBAL           41   1W    48W   BOOLEAN


REFERENCED CONSTANTS
NAME            VALUE                                  LINE

BELGIAN         ENUMERATION CONSTANT                     16
CANADIAN        ENUMERATION CONSTANT                     21
CLEAR_SCREEN
DUTCH           ENUMERATION CONSTANT                     18
ITALIAN         ENUMERATION CONSTANT                     19
JAPANESE        ENUMERATION CONSTANT                     23
NO_CHOICE       ENUMERATION CONSTANT                     24
SAUDI           ENUMERATION CONSTANT                     22
SWISS           ENUMERATION CONSTANT                     20
TAB                                                      12
W_GERMAN        ENUMERATION CONSTANT                     17


REFERENCED TYPES
NAME            FIELD OFFSET   LINE  SIZE      TYPE DESCRIPTION

COUNTRY                         16   1W        (BELGIAN,W_GERMAN,DUTCH,ITALIAN,
                                                 SWISS,CANADIAN,SAUDI,JAPANESE,
                                                 NO_CHOICE)

MONETARY_SYMBOLS
                                26   4C        PACKED ARRAY[1..4] OF CHAR
RATEDATA                        27   39W       RECORD
                    0C                 6C          RATE_DATE :PACKED ARRAY[1..6]
                                                     OF CHAR
                    3W                 36W         RATES :RECORD
                    0W                 18W           US_FOREIGN :ARRAY[COUNTRY]
                                                       OF REAL
                    18W                18W           FOREIGN_US :ARRAY[COUNTRY]
                                                       OF REAL
RFILE                           35   4W        FILE OF RATEDATA
```

Figure C-10. Compilation Listing of the EXCHANGEP Program (continued)

```
PROCEDURE BUILDER ON LINE 48


VARIABLE ALLOCATION
NAME            SCOPE          LINE  SIZE   LOC   TYPE

RATE_REC        LOCAL            56   39W   14W  RATEDATA
SELECTION       LOCAL            55   1W    12W  COUNTRY




PROCEDURE MENU ON LINE 86
NO DECLARED NAMES




PROCEDURE GET_SELECTION ON LINE 114


VARIABLE ALLOCATION
NAME            SCOPE          LINE  SIZE   LOC   TYPE

CURRENCY_CODE LOCAL             121   2W    12W  INTEGER
SELECTION       PARAMETER        115   1W         COUNTRY



PROCEDURE CONVERSION ON LINE 151


VARIABLE ALLOCATION
NAME            SCOPE          LINE  SIZE   LOC   TYPE

INVAL           LOCAL            155   2W    12W  REAL
OUTVAL          LOCAL            155   2W    14W  REAL
RATE            LOCAL            155   2W    16W  REAL
RESPONSE        LOCAL            156   1C    18W  CHAR
SELECTION       PARAMETER        152   1W         COUNTRY
```

Figure C-10. Compilation Listing of the EXCHANGEP Program (concluded)

```
USER PROGRAM exchangep SWAT AUDIT ON 09/13/82 AT 13:14:34

AOS/VS SWAT Revision 02.20.00.00 ON 09/13/82 AT 13:14:39
PROGRAM -- :UDD:TOMF:SAMPLES:EXCHANGEP
>% This session illustrates the use of SWAT commands
>% with a program written in PASCAL.
>%
>% When invoking the SWAT debugger, we used the /AUDIT= switch
>% to open an audit file.
>%
>% Confirm that auditing is on.
>AUDIT
ON
>% Display the working environment, which should be the main procedure.
>ENVIRONMENT
:EXCHANGE
>% List all the source code.
>LIST @ALL
1       PROGRAM
2               exchange (INPUT, OUTPUT, rate_file);
3
4       {       This program calculates currency exchange values based on
5               daily international exchange rates.  The operator selects
6               the type of exchange, then enters the incoming amount.  The
7               program performs the calculation and displays the result.     }
8
9       {$C+    Compiler directive allows ASCII '<nn>' notation in the program }
10
11      CONST
12              TAB = '<11>';
13              CLEAR_SCREEN = '<14>';
14
15      TYPE
16              country = (belgian,
17                         w_german,
18                         dutch,
19                         italian,
20                         swiss,
21                         canadian,
22                         saudi,
23                         japanese,
24                         no_choice);
25
26              monetary_symbols = PACKED ARRAY [1..4] OF CHAR;
27              ratedata = RECORD
28                          rate_date : PACKED ARRAY [1..6] OF CHAR;
29                          rates : RECORD
30                                  us_foreign : ARRAY[country] OF REAL;
31                                  foreign_us : ARRAY[country] OF REAL;
32                                     END; { rates }
33                          END; { ratedata }
34
35              rfile = FILE OF ratedata;
36
```

Figure C-11. Audit File of the EXCHANGEP SWAT Session (continues)

```
37      VAR
38
39              selection : country;
40              currency : ARRAY [country] OF monetary_symbols;
41              us_to_foreign : BOOLEAN;
42              rate_file : rfile;
43              daily_rates : ratedata;
44              i : INTEGER;
45              response : CHAR;
46
47      {-------------------------------------------------------------------}
48      PROCEDURE
49              builder;
50
51      {       This procedure builds the currency rate file for the exchange
52              program.    }
53
54      VAR
55              selection : country;
56              rate_rec  : ratedata;
57
58      BEGIN
59              REWRITE (rate_file);
60
61              WRITELN (CLEAR_SCREEN);
62              WRITELN (TAB, 'Building new currency rate file');
63              WRITELN;
64              WRITE (TAB, 'Enter the date in yymmdd format:  ');
65              READLN (rate_rec.rate_date);
66
67              selection := belgian;
68              REPEAT
69                      WRITE (TAB, 'Enter rate for US$ to ',
70                                      currency[selection], ' :  ');
71                      READLN (rate_rec.rates.us_foreign[selection]);
72
73                      WRITE (TAB, 'Enter rate for ' , currency[selection],
74                                      ' to US$ :  ');
75                      READLN (rate_rec.rates.foreign_us[selection]);
76                      selection := SUCC(selection);
77
78              UNTIL (selection = no_choice);
79
80              WRITE (rate_file, rate_rec);
81
82              RESET (rate_file);
83              READ (rate_file, daily_rates);
84      END;
85      {-------------------------------------------------------------------}
```

Figure C-11. Audit File of the EXCHANGEP SWAT Session (continued)

```
86      PROCEDURE
87              menu;
88
89      BEGIN
90              WITH daily_rates DO
91              BEGIN
92              WRITE ('Rates for ');
93              WRITE (rate_date[3], rate_date[4],'/');
94              WRITE (rate_date[5], rate_date[6],'/');
95              WRITE (rate_date[1], rate_date[2]);
96              WRITELN;
97              WRITELN ('Identify the type of currency:');
98              WRITELN;
99              WRITELN (TAB, '1   Belgian francs');
100             WRITELN (TAB, '2   W. German marks');
101             WRITELN (TAB, '3   Dutch guilders');
102             WRITELN (TAB, '4   Italian lire');
103             WRITELN (TAB, '5   Swiss francs');
104             WRITELN (TAB, '6   Canadian dollars');
105             WRITELN (TAB, '7   Saudi riyals');
106             WRITELN (TAB, '8   Japanese yen');
107             WRITELN (TAB, '(Type 0 to end the program.)');
108             WRITELN;
109             WRITE ('Enter the currency code:  ');
110             END;
111
112     END;
113     {-------------------------------------------------------------------}
114     PROCEDURE
115             get_selection ( VAR selection : country );
116
117     {       Gets the users currency code input and sets parameter selection
118             to the corresponding country.                                 }
119
120     VAR
121             currency_code : INTEGER;
122
123     BEGIN
124
125             READLN (currency_code);
126
127             IF (currency_code >= 0) AND (currency_code <= 8) THEN
128
129                     BEGIN
130
131                     CASE currency_code OF
132
133                             0 : selection := no_choice;
134                             1 : selection := belgian;
135                             2 : selection := w_german;
136                             3 : selection := dutch;
137                             4 : selection := italian;
138                             5 : selection := swiss;
139                             6 : selection := canadian;
140                             7 : selection := saudi;
141                             8 : selection := japanese;
```

Figure C-11. Audit File of the EXCHANGEP SWAT Session (continued)

```
142                     END
143                     END
144
145             ELSE    selection := no_choice;
146
147      END; { get_selection }

148
149     {-------------------------------------------------------------------------}
150
151     PROCEDURE
152             conversion ( selection : country );
153
154     VAR
155             inval, outval, rate : REAL;
156             response            : CHAR;
157
158     BEGIN
159             WRITELN;
160             WRITELN (TAB, 'Do you want to convert US$ into Foreign currency?');
161             WRITE (TAB, 'Enter "Y" or "N":  ');
162             READLN (response);
163
164             IF ((response = 'Y') OR (response = 'y')) THEN
165                     BEGIN
166                             us_to_foreign := TRUE;
167                             rate := daily_rates.rates.us_foreign[selection];
168                     END
169
170             ELSE
171                     BEGIN
172                             us_to_foreign := FALSE;
173                             rate := daily_rates.rates.foreign_us[selection];
174                     END;
175
176             WRITELN;
177
178             IF us_to_foreign THEN WRITE (TAB, 'Enter US$:  ')
179             ELSE WRITE (TAB, 'Enter ', currency[selection], ':  ');
180
181             READLN (inval);
182             outval := inval * rate;
183
184             WRITELN;
185             IF us_to_foreign THEN WRITELN ('$', inval:10:2,
186               ' US equivalent to:  ', outval:10:2, ' ', currency[selection])
187             ELSE WRITELN (inval:10:2, ' ', currency[selection],
188               ' equivalent to:  $', outval:10:2, ' US');
189
190      END;
191     {-------------------------------------------------------------------------}
```

Figure C-11. Audit File of the EXCHANGEP SWAT Session (continued)

```
192
193
194     BEGIN { main procedure }
195
196             currency[belgian] := 'BF';
197             currency[w_german] := 'DM';
198             currency[dutch] := 'gld';
199             currency[italian] := 'Lit';
200             currency[swiss] := 'SF';
201             currency[canadian] := 'Can$';
202             currency[saudi] := 'SRI';
203             currency[japanese] := 'Y';
204
205             WRITELN (CLEAR_SCREEN, TAB, TAB, 'Exchange Program');
206             WRITELN;
207
208             WRITELN (TAB, 'Do you want to create a new data file?');
209             WRITE (TAB, 'Enter "Y" or "N":  ');
210             READLN (response);
211
212             IF ((response = 'Y') OR (response = 'y')) THEN builder
213             ELSE
214             BEGIN
215
216                     RESET (rate_file);
217                     READ (rate_file, daily_rates);
218
219                     WRITELN;
220                     WRITE (TAB, 'The file contains exchange rates for:  ');
221                     WRITELN (daily_rates.rate_date);
222
223                     WRITE (TAB,
224                             'Do you want to change it? Enter "Y" or "N":  ');
225                     READLN (response);
226
227                     IF ((response = 'Y') OR (response = 'y')) THEN builder;
228             END;
229
230             WRITELN(CLEAR_SCREEN);
231
232             menu;
233             get_selection (selection);
234
235             WHILE (selection <> no_choice) DO
236
237             BEGIN
238                     conversion (selection);
239
240                     WRITELN;
241                     WRITE (TAB, 'Press New Line to continue');
242                     READLN;
243                     WRITELN (CLEAR_SCREEN);
```

Figure C-11. Audit File of the EXCHANGEP SWAT Session (continued)

```
244
245                     menu:
246                     get_selection (selection);
247             END;
248
249     WRITELN (CLEAR_SCREEN, TAB, 'Exchange program ended.');
250
251     END {EXCHANGE program}.



>% Display the working directory.
>DIRECTORY
:UDD:TOMF:SAMPLES
>% Display the current search list.
>SEARCHLIST
:UDD:TOMF :UTIL :SWAT :LANG_RT :PASCAL
>% Obtain information about global symbols.
>DESCRIBE CLEAR_SCREEN
CLEAR_SCREEN (1 byte) CONST CHAR
>DESCRIBE country, swiss, monetary_symbols
COUNTRY (1 word) TYPE (BELGIAN, W_GERMAN, DUTCH, ITALIAN, SWISS, CANADIAN,
    SAUDI, JAPANESE, NO_CHOICE)
SWISS (1 word) CONST COUNTRY
MONETARY_SYMBOLS (4 bytes) TYPE PACKED ARRAY [1..4] OF CHAR
>DESCRIBE rfile  selection  currency
RFILE (4 words) TYPE FILE OF RATEDATA
SELECTION (1 word at 16000001046R8 words) VAR COUNTRY
CURRENCY (36 bytes at 34000002050R8 bytes) VAR ARRAY [COUNTRY] OF
    MONETARY_SYMBOLS
>DESCRIBE i, response
I (2 words at 16000000744R8 words) VAR INTEGER
RESPONSE (1 byte at 34000001704R8 bytes) VAR CHAR
>DESCRIBE get_selection
GET_SELECTION (at 16001744532R8 words) SUBROUTINE line 115 to 147
>% Set the environment to that of the procedure called builder.
>ENVIRONMENT builder
:EXCHANGE:BUILDER
>% Display information about this procedure's symbols.
>DESCRIBE selection, rate_rec
SELECTION (1 word at +14R8 words) VAR COUNTRY
RATE_REC (39 words at +16R8 words) VAR RATEDATA
>% Set the environment to that of the procedure called get_selection.
>ENVIRONMENT ^get_selection
:EXCHANGE:GET_SELECTION
>% Display information about this procedure's symbols.
>DESCRIBE currency_code
CURRENCY_CODE (2 words at +14R8 words) VAR INTEGER
>% Set the environment to that of the procedure called conversion.
>ENVIRONMENT ^conversion
:EXCHANGE:CONVERSION
>% Display information about this procedure's symbols.
>DESCRIBE inval response
INVAL (2 words at +14R8 words) VAR REAL
RESPONSE (1 byte at +22R8 words) VAR CHAR
>% Return to the environment of the main procedure
>ENVIRONMENT @MAIN
:EXCHANGE
```

Figure C-11. Audit File of the EXCHANGEP SWAT Session (continued)

```
>% Set breakpoints at key statements
>BREAKPOINT 76 ACTION := 'LIST 76;TYPE rate_rec.rates.us_foreign[selection];
  TYPE rate_rec.rates.foreign_us[selection]'
Set at :EXCHANGE:BUILDER:76 action="LIST 76;
  TYPE rate_rec.rates.us_foreign[selection];
  TYPE rate_rec.rates.foreign_us[selection]"
>BREAKPOINT 147 ACTION := 'TYPE currency_code;TYPE selection'
Set at :EXCHANGE:GET_SELECTION:147 action="TYPE currency_code;
  TYPE selection"
>BREAKPOINT 167 ACTION := 'LIST 167; WALKBACK'
Set at :EXCHANGE:CONVERSION:167 action="LIST 167; WALKBACK"
>BREAKPOINT 173 ACTION := 'LIST 173; WALKBACK'
Set at :EXCHANGE:CONVERSION:173 action="LIST 173; WALKBACK"
>BREAKPOINT 182 ACTION := 'TYPE inval, rate'
Set at :EXCHANGE:CONVERSION:182 action="TYPE inval, rate"
>BREAKPOINT 212
Set at :EXCHANGE:212
>% Display the breakpoints currently in effect.
>BREAKPOINT
Set at :EXCHANGE:BUILDER:76 action="LIST 76;
  TYPE rate_rec.rates.us_foreign[selection];
  TYPE rate_rec.rates.foreign_us[selection]"
Set at :EXCHANGE:GET_SELECTION:147 action="TYPE currency_code;
  TYPE selection"
Set at :EXCHANGE:CONVERSION:167 action="LIST 167; WALKBACK"
Set at :EXCHANGE:CONVERSION:173 action="LIST 173; WALKBACK"
Set at :EXCHANGE:CONVERSION:182 action="TYPE inval, rate"
Set at :EXCHANGE:212
>% Begin execution of the program.
>CONTINUE

Breakpoint trap at :EXCHANGE:212
>% The program asked if we wanted to create a new data file; we said N.
>% We can, however, change our response.
>TYPE response
"N"
>SET response := 'Y'
Old value: "N"
New Value: "Y"
>CLEAR 212
Cleared at :EXCHANGE:212
>CONTINUE

Breakpoint trap at :EXCHANGE:BUILDER:76 action="LIST 76;
  TYPE rate_rec.rates.us_foreign[selection];
  TYPE rate_rec.rates.foreign_us[selection]"
76C                    selection := SUCC(selection);
 4.752000E+01
 2.100000E-02
>% We entered the date and the first exchange rate pair (Belgian francs)
>CONTINUE
```

Figure C-11. Audit File of the EXCHANGEP SWAT Session (continued)

```
Breakpoint trap at :EXCHANGE:BUILDER:76 action="LIST 76;
  TYPE rate_rec.rates.us_foreign[selection];
  TYPE rate_rec.rates.foreign_us[selection]"
76C                    selection := SUCC(selection);
 2.492500E+00
 4.015000E-01


>% We entered the next pair of exchange rates (West German marks)
>% We'll clear this breakpoint, then fill in the rest of the table.
>TYPE selection
W_GERMAN
>CLEAR 76
Cleared at :EXCHANGE:BUILDER:76 action="LIST 76;
  TYPE rate_rec.rates.us_foreign[selection];
  TYPE rate_rec.rates.foreign_us[selection]"
>CONTINUE

Breakpoint trap at :EXCHANGE:GET_SELECTION:147
  action="TYPE currency_code;TYPE selection"
2
W_GERMAN
>% The program displayed the menu.  We entered code 2 (West German marks)
>% We can change the variable and redirect execution.  This time we'll
>% convert Swiss francs.
>SET selection := swiss
Old value: W_GERMAN
New Value: SWISS
>CONTINUE

Breakpoint trap at :EXCHANGE:CONVERSION:167 action="LIST 167; WALKBACK"
167C                     rate := daily_rates.rates.us_foreign[selection];
Current location is :EXCHANGE:CONVERSION:167
Called from :EXCHANGE:238
>% The program asked if we are exchanging US dollars to foreign currency.
>% We responded Y.
>TYPE daily_rates.rates.us_foreign[selection]
 2.116500E+00
>CONTINUE

Breakpoint trap at :EXCHANGE:CONVERSION:182 action="TYPE inval, rate"
 2.500000E+02
 2.116500E+00
>TYPE inval * rate
 5.291250E+02
>% The result of the exchange should be 529.12 Swiss francs.
>CONTINUE

Breakpoint trap at :EXCHANGE:GET_SELECTION:147
  action="TYPE currency_code;TYPE selection"
6
CANADIAN
```

Figure C-11. Audit File of the EXCHANGEP SWAT Session (continued)

```
>% The program displayed the expected result.  It then presented the menu
>% and we entered 6 (for Canadian dollars).
>CONTINUE

Breakpoint trap at :EXCHANGE:CONVERSION:173 action="LIST 173: WALKBACK"
173C                            rate := daily_rates.rates.foreign_us[selection];
Current location is :EXCHANGE:CONVERSION:173
Called from :EXCHANGE:238
>% The program asked the exchange direction.  We entered N.
>% We will set a breakpoint at 190 so that we do not exit that procedure.
>BREAKPOINT 190
Set at :EXCHANGE:CONVERSION:190
>CONTINUE

Breakpoint trap at :EXCHANGE:CONVERSION:182 action="TYPE inval, rate"
 2.500000E+02
 8.014000E-01
>CONTINUE

Breakpoint trap at :EXCHANGE:CONVERSION:190
>SET inval := 250.
Old value:  2.500000E+02
New Value:  2.500000E+02
>SET us_to_foreign := TRUE
Old value: FALSE
New Value: TRUE
>SET rate := daily_rates.rates.us_foreign[selection]
Old value:  8.014000E-01
New Value:  1.247800E+00
>CONTINUE AT 184

Breakpoint trap at :EXCHANGE:CONVERSION:190
>CONTINUE

Breakpoint trap at :EXCHANGE:GET_SELECTION:147
  action="TYPE currency_code:TYPE selection"
0
NO_CHOICE
>% We entered 0 to end the program.  If we continue, both the program
>% and SWAT will terminate.  We can also end this session by typing
>BYE

SWAT TERMINATED
```

Figure C-11. Audit File of the EXCHANGEP SWAT Session (concluded)

A Sample C Program and Audit File

This C program, called EXCHANGEC, is similar to the sample PL/I program used in Chapter 3 of this manual. It calculates currency exchange values based on daily exchange rates. The program consists of a single module, which interacts with the operator and a file called RATE_FILE.

The compilation and link commands are as follows:

CC/DEBUG/LS EXCHANGEC

CCL/DEBUG EXCHANGEC

(The CCL macro automatically includes the SWATI file; do not specify it as an argument.)

Figure C-12 presents the compilation listing of the EXCHANGEC program. A sample SWAT session audit file appears in Figure C-13.

```
Source file: exchangec
Compiled on 15-Sep-82 at 10:41:39 by AOS/VS C Rev 01.00.08.126
Options:  cc/l=exchangec.ls/debug

   1 0  /*      This program calculates currency exchange values based on
   2*0          daily international exchange rates.  The operator selects
   3*0          the type of exchange, then enters the incoming amount.  The
   4*0          program performs the calculation and displays the result.    */
   5 0
   6 0  #nolist
 633 0  #list
 634 0
 635 0  #define TRUE 1
 636 0  #define FALSE 0
 637 0  typedef enum    {belgian,
 638 1                   w_german,
 639 1                   dutch,
 640 1                   italian,
 641 1                   swiss,
 642 1                   canadian,
 643 1                   saudi,
 644 1                   japanese,
 645 1                   no_choice} country ;
 646 0
 647 0  typedef char monetary_symbols[5];
 648 0  typedef struct {
 649 1                    char rate_date[10];
 650 1                    struct {
 651 2                         float us_foreign[no_choice];
 652 2                         float foreign_us[no_choice];
 653 2                    } rates;
 654 1          } ratedata;
 655 0
 656 0  FILE *rate_file;
 657 0
 658 0  country selection;
 659 0  static monetary_symbols currency[no_choice] =
 660 0                {"BF","DM","gld","Lit","SF","Can$","SRI","Y",""};
```

Figure C-12. Compilation Listing of the EXCHANGEC Program (continues)

```
661 0  int us_to_foreign;
662 0  ratedata daily_rates;
663 0  int i;
664 0  char response;
665 0  /*---------------------------------------------------------------*/


666 0   builder()
667 0
668 0  /*      This procedure builds the currency rate file for the
669*0         exchange program.                                        */
670 0  {
671 1  country selection;
672 1  ratedata rate_rec;
673 1
674 1          rate_file = fopen ("rate_file", "u");
675 1
676 1          printf ("\f");
677 1          printf ("\tBuilding new currency rate file\n\n");
678 1
679 1          printf ("\tEnter the date in yymmdd format: ");
680 1          scanf ("%s", rate_rec.rate_date);
681 1
682 1          for (selection = belgian; selection != no_choice; selection++)
683 1          {
684 2          printf ("\tEnter rate for US$ to %s : ", currency[selection]);
685 2          scanf ("%f", &rate_rec.rates.us_foreign[selection]);
686 2
687 2          printf ("\tEnter rate for %s to US$ : ", currency[selection]);
688 2          scanf ("%f", &rate_rec.rates.foreign_us[selection]);
689 2          }
690 1
691 1          fwrite (&rate_rec, sizeof(rate_rec), 1, rate_file);
692 1
693 1          fseek (rate_file, 0L, 0);
694 1          fread (&daily_rates, sizeof(daily_rates), 1, rate_file);
695 1  }
696 0  /*---------------------------------------------------------------*/
697 0  menu ()
698 0
699 0  /* Displays the selection menu.                                  */
700 0
701 0  {
702 1  printf ("Rates for ");
703 1  printf ("%c%c/", daily_rates.rate_date[2], daily_rates.rate_date[3]);
704 1  printf ("%c%c/", daily_rates.rate_date[4], daily_rates.rate_date[5]);
705 1  printf ("%c%c\n", daily_rates.rate_date[0], daily_rates.rate_date[1]);
706 1
707 1          printf ("Identify the type of currency:\n\n");
708 1          printf ("\t1   Belgian francs\n");
709 1          printf ("\t2   W. German marks\n");
710 1          printf ("\t3   Dutch guilders\n");
711 1          printf ("\t4   Italian lire\n");
712 1          printf ("\t5   Swiss francs\n");
713 1          printf ("\t6   Canadian dollars\n");
714 1          printf ("\t7   Saudi riyals\n");
```

Figure C-12. Compilation Listing of the EXCHANGEC Program (continued)

```
768 2          }
769 1
770 1          printf("\n");
771 1
772 1          if (us_to_foreign == TRUE)  printf ("\n \tEnter US$:  ");
773 1          else printf ("\n \tEnter %s :", currency[selection]);
774 1
775 1          scanf ("%f", &inval);
776 1          outval = inval * rate;

777 1
778 1          printf ("\n");
779 1          if (us_to_foreign == TRUE)
780 1                 printf ("\n\t$ %5.2f  US equivalent to: %5.2f  %s",
781 1                                inval, outval, currency[selection]);
782 1
783 1          else    printf ("\n\t %5.2f %s equivalent to: $ %5.2f US",
784 1                                inval, currency[selection], outval);
785 1   }
786 0   /*---------------------------------------------------------------*/
787 0   main ()
788 0
789 0   /* The main procedure drives the interactive session through the
790*0          rate file processing and conversion selection phases.   */
791 0
792 0   {
793 1          printf ("\f\t\tExchange Program\n\n");
794 1
795 1          printf ("\tDo you want to create a new data file?\n");
796 1          printf ("\tEnter Y or N:  ");
797 1          scanf ("%s", &response);
798 1
799 1          if ((response == 'Y') || (response == 'y')) builder ();
800 1          else
801 1          {
802 2
803 2                 rate_file = fopen ("rate_file", "r");
804 2                 fread (&daily_rates, sizeof(daily_rates), 1, rate_file);
805 2
806 2                 printf ("\n\tThe file contains exchange rates for:  ");
807 2                 printf ("%s\n", daily_rates.rate_date);
808 2
809 2                 printf ("\tDo you want to change it? Enter Y or N:  ");
810 2                 scanf ("%s", &response);
811 2
812 2                 if ((response == 'Y') || (response == 'y'))
813 2                 {
814 3                        fclose (rate_file);
815 3                        builder ();
816 3                 }
817 2          }
818 1
819 1          printf ("\f");
820 1
821 1          menu ();
822 1          get_selection ();
823 1
824 1          while (selection != no_choice)
825 1
```

Figure C-12. Compilation Listing of the EXCHANGEC Program (continued)

```
715 1          printf ("\t8   Japanese yen\n");
716 1          printf ("\t(Type 0 to end the program.)\n\n");
717 1
718 1          printf ("Enter the currency code: ");
719 1    }
720 0    /*-------------------------------------------------------------*/

721 0    get_selection ()
722 0
723 0    /* Gets the users currency code input and sets parameter selection
724*0          to the corresponding country.                          */
725 0    {
726 1
727 1    int currency_code;
728 1
729 1          scanf ("%d", &currency_code);
730 1
731 1          switch  (currency_code)
732 1          {
733 2                case 1 : selection = belgian; break;
734 2                case 2 : selection = w_german; break;
735 2                case 3 : selection = dutch; break;
736 2                case 4 : selection = italian; break;
737 2                case 5 : selection = swiss; break;
738 2                case 6 : selection = canadian; break;
739 2                case 7 : selection = saudi; break;
740 2                case 8 : selection = japanese; break;
741 2
742 2                default : selection = no_choice; break;
743 2          }
744 1    }
745 0    /*-------------------------------------------------------------*/
746 0    conversion ()
747 0
748 0    /* Performs the actual conversion based on the selection and
749*0          outputs the result.                                    */
750 0
751 0    {
752 1    float inval, outval, rate;
753 1    char response;
754 1
755 1          printf ("\nDo you want to convert US$ into Foreign currency?");
756 1          printf ("\tEnter Y or N: ");
757 1          scanf ("%s", &response);
758 1
759 1          if ((response == 'Y') || (response == 'y'))
760 1          {
761 2                us_to_foreign = TRUE;
762 2                rate = daily_rates.rates.us_foreign[selection];
763 2          }
764 1          else
765 1          {
766 2                us_to_foreign = FALSE;
767 2                rate = daily_rates.rates.foreign_us[selection];
```

Figure C-12. Compilation Listing of the EXCHANGEC Program (continued)

```
swiss           const.   ----- -----       641   Constant 4
canadian        const.   ----- -----       642   Constant 5
saudi           const.   ----- -----       643   Constant 6
japanese        const.   ----- -----       644   Constant 7
no_choice       const.   ----- -----       645   Constant 8

FILE            typedef    32W -----        67   struct _iobuf
country         typedef     1W -----       645   enum
monetary_symbols
                typedef     5C -----       647   char [5]
ratedata        typedef    37W -----       654   struct

currency        static     45C     0+ 0C  659   monetary_symbols [9]
$iob           #extern   2048W  ext.        75   FILE [64]
fclose          extern      0W  ext.        83   int ()
fopen           extern      0W  ext.        87   FILE *()
freopen        #extern      0W  ext.       101   FILE *()
fgetc          #extern      0W  ext.       109   int ()
fputc          #extern      0W  ext.       114   int ()
$getc          #extern      0W  ext.       120   int ()
$putc          #extern      0W  ext.       124   int ()
ungetc         #extern      0W  ext.       128   void ()
fputs          #extern      0W  ext.       136   int ()
fgets          #extern      0W  ext.       143   char *()
puts           #extern      0W  ext.       152   char *()
gets           #extern      0W  ext.       158   char *()
fread           extern      0W  ext.       165   int ()
fwrite          extern      0W  ext.       178   int ()
getw           #extern      0W  ext.       192   int ()
putw           #extern      0W  ext.       196   int ()
fseek           extern      0W  ext.       200   int ()
ftell          #extern      0W  ext.       214   long int ()
fflush         #extern      0W  ext.       220   int ()
printf          extern      0W  ext.       284   int ()
fprintf        #extern      0W  ext.       291   int ()
sprintf        #extern      0W  ext.       299   int ()
scanf           extern      0W  ext.       308   int ()
fscanf         #extern      0W  ext.       317   int ()
sscanf         #extern      0W  ext.       327   int ()
atoi           #extern      0W  ext.       342   int ()
atof           #extern      0W  ext.       347   double ()
atol           #extern      0W  ext.       352   long int ()
atou           #extern      0W  ext.       358   unsigned ()
ftoa           #extern      0W  ext.       364   char *()
itoa           #extern      0W  ext.       377   char *()
utoa           #extern      0W  ext.       383   char *()
calloc         #extern      0W  ext.       395   char *()
malloc         #extern      0W  ext.       401   char *()
alloc          #extern      0W  ext.       407   char *()
intss          #extern      0W  ext.       429   int ()
intso          #extern      0W  ext.       432   int ()
strcmp         #extern      0W  ext.       442   int ()
strncmp        #extern      0W  ext.       449   int ()
```

Figure C-12. Compilation Listing of the EXCHANGEC Program (continued)

```
 826 1          {
 827 2                  conversion ();
 828 2
 829 2                  printf ("\n\tEnter Y to continue : ");
 830 2                  scanf ("%s", &response);
 831 2                  printf ("\f");
 832 2
 833 2                  menu ();
 834 2                  get_selection ();
 835 2          }
 836 1
 837 1  printf ("\f\tExchange program ended.");
 838 1  }

-------
1234567       Listing format:

Columns 1-4:  Line Number.

Column  5:    One of the character(s):
                * Line started out inside comment.

Column  6:    Number of nested {'s:
                0 ... 9 --  0-9  nested {'s.
Source file: exchangec
Compiled on 15-Sep-82 at 10:41:52 by AOS/VS C Rev 01.00.08.126

                              Allocation Map

External Block
--------------

Identifier     Storage   Size  Loc.Off.  Line   Type

_iobuf         #struct    32W -----        53   struct
_ptr           #member     2W     0W       54   unsigned char *
_bstart        #member     2W     2W       55   unsigned char *
_xmt           #member     2W     4W       56   int
_icount        #member     1W     6W       57   short int
_ocount        #member     1W     7W       58   short int
_bflag         #member     1W     8W       59   short int
_bufsize       #member     1W     9W       60   short int
_fd            #member     1W    10W       61   short int
_tsize         #member     1W    11W       62   short int
_look_ahead    #member     1C    12+ 0C    63   unsigned char
_temp_buf      #member     1C    12+ 1C    64   unsigned char
_pad           #member     1W    13W       65   short int
_reserved      #member    18W    14W       66   int [9]

(no name)       struct    37W -----       648   struct
rate_date       member    10C     0+ 0C   649   char [10]
rates           member    32W     5W      653   struct

(no name)       struct    32W -----       650   struct
us_foreign      member    16W     0W      651   float [8]
foreign_us      member    16W    16W      652   float [8]

(no name)      #enum       1W -----       637   enum
belgian         const.  ----- -----       637   Constant 0
w_german        const.  ----- -----       638   Constant 1
dutch           const.  ----- -----       639   Constant 2
italian         const.  ----- -----       640   Constant 3
```

Figure C-12. Compilation Listing of the EXCHANGEC Program (continued)

```
Function conversion on line 751
-------------------------------

Identifier      Storage   Size  Loc.Off.  Line   Type

inval           auto        2W    12W      752   float
outval          auto        2W    14W      752   float
rate            auto        2W    16W      752   float
response        auto        1C    10+ 0C   753   char


Function main on line 792
-------------------------

       No declared entries.

# -- This identifier was not referenced.

Character following size field:
    C       Size is number of bytes ( 8 bits)
    W       Size is number of words (16 bits)

(Location field is word address + bit/byte offset)
```

Figure C-12. Compilation Listing of the EXCHANGEC Program (concluded)

```
strcpy          #extern      0W  ext.       456   char *()
strncpy         #extern      0W  ext.       461   char *()
strcat          #extern      0W  ext.       470   char *()
strncat         #extern      0W  ext.       475   char *()
strlen          #extern      0W  ext.       484   int ()
strsave         #extern      0W  ext.       488   char *()
strnsave        #extern      0W  ext.       493   char *()
strchr          #extern      0W  ext.       501   char *()
strrchr         #extern      0W  ext.       507   char *()
wdleng          #extern      0W  ext.       517   int ()
$classify       #extern   -----  ext.       570   short int []
rate_file        extern      2W  ext.       656   FILE *
selection        extern      1W  ext.       658   country
us_to_foreign   extern      2W  ext.       661   int
daily_rates      extern     37W  ext.       662   ratedata
i               #extern      2W  ext.       663   int
response         extern      1C  ext.       664   char
free            #builtin     0W     0W      415   void ()
sqrt            #builtin     0W     0W      515   double ()


Function builder on line 670
----------------------------

Identifier      Storage   Size  Loc.Off.  Line   Type

selection       auto         1W    10W      671   country
rate_rec        auto        37W    12W      672   ratedata


Function menu on line 701
-------------------------

        No declared entries.


Function get_selection on line 725
----------------------------------

Identifier      Storage   Size  Loc.Off.  Line   Type

currency_code   auto         2W    10W      727   int
```

Figure C-12. Compilation Listing of the EXCHANGEC Program (continued)

```
>SEARCHLIST
:UDD:TOMF :UTIL :SWAT :LANG_RT :C
>% Obtain information about global symbols.
>DESCRIBE country, swiss, monetary_symbols
country (1 word) typedef { belgian, w_german, dutch, italian, swiss, canadian,
    saudi, japanese, no_choice }
swiss (1 word) enum
monetary_symbols (5 bytes) typedef char [5]
>DESCRIBE ratedata  selection  currency
ratedata (37 words) typedef
    rate_date (10 bytes at +0R8 bytes) char [10]
    rates (32 words at +5R8 words) struct
      us_foreign (16 words at +0R8 words) float [8]
      foreign_us (16 words at +20R8 words) float [8]
selection (1 word at +12R8 words) country
currency (45 bytes at 34000023664R8 bytes) static typedef char [9][5]
>DESCRIBE i, response
i (2 words at 16000005014R8 words) extern int
response (1 byte at 34000023604R8 bytes) extern char
>% Set the environment to that of the procedure called main.
>ENVIRONMENT :main
:main
>% List this procedure.
>DESCRIBE :main
main (at 16001733173R8 words) extern line 792 to 838 int ()
>LIST 792 838
792     {
793             printf ("\f\t\tExchange Program\n\n");
794
795             printf ("\tDo you want to create a new data file?\n");
796             printf ("\tEnter Y or N:  ");
797             scanf ("%s", &response);
798
799             if ((response == 'Y') || (response == 'y')) builder ();
800             else
801             {
802
803                     rate_file = fopen ("rate_file", "r");
804                     fread (&daily_rates, sizeof(daily_rates), 1, rate_file);
805
806                     printf ("\n\tThe file contains exchange rates for:  ");
807                     printf ("%s\n", daily_rates.rate_date);
808
809                     printf ("\tDo you want to change it? Enter Y or N:  ");
810                     scanf ("%s", &response);
811
812                     if ((response == 'Y') || (response == 'y'))
813                     {
814                             fclose (rate_file);
815                             builder ();
816                     }
817             }
818
819             printf ("\f");
820
821             menu ();
822             get_selection ();
823
824             while (selection != no_choice)
825
```

Figure C-13. Audit File of the EXCHANGEC SWAT Session (continued)

```
USER PROGRAM exchangec SWAT AUDIT ON 09/15/82 AT 12:27:38

AOS/VS SWAT Revision 02.20.00.00 ON 09/15/82 AT 12:27:42
PROGRAM -- :UDD:TOMF:SAMPLES:EXCHANGEC
>% This session illustrates the use of SWAT commands
>% with a program written in the C language.
>%
>% When invoking the SWAT debugger, we used the /AUDIT= switch
>% to open an audit file.
>%
>% Confirm that auditing is on.
>AUDIT
ON
>% Display the environments in this module.
>ENVIRONMENT @ALL
:builder
:menu
:get_selection
:conversion
:main
CURRENT ENVIRONMENT IS :builder
>% Display this procedure.
>DESCRIBE :builder
builder (at 16001732066R8 words) extern line 670 to 695 int ()
>LIST 670 695
670     {
671     country selection;
672     ratedata rate_rec;
673
674             rate_file = fopen ("rate_file", "u");
675
676             printf ("\f");
677             printf ("\tBuilding new currency rate file\n\n");
678
679             printf ("\tEnter the date in yymmdd format: ");
680             scanf ("%s", rate_rec.rate_date);
681
682             for (selection = belgian; selection != no_choice; selection++)
683             {
684             printf ("\tEnter rate for US$ to %s : ", currency[selection]);
685             scanf ("%f", &rate_rec.rates.us_foreign[selection]);
686
687             printf ("\tEnter rate for %s to US$ : ", currency[selection]);
688             scanf ("%f", &rate_rec.rates.foreign_us[selection]);
689             }
690
691             fwrite (&rate_rec, sizeof(rate_rec), 1, rate_file);
692
693             fseek (rate_file, 0L, 0);
694             fread (&daily_rates, sizeof(daily_rates), 1, rate_file);
695     }
>% Display the working directory.
>DIRECTORY
:UDD:TOMF:SAMPLES
>% Display the current search list.
```

Figure C-13. Audit File of the EXCHANGEC SWAT Session (continues)

```
>% List this procedure.
>DESCRIBE :conversion
conversion (at 16001732651R8 words) extern line 751 to 785 int ()
>LIST 751 785
751     {
752     float inval, outval, rate;
753     char response;
754
755             printf ("\nDo you want to convert US$ into Foreign currency?");
756             printf ("\tEnter Y or N: ");
757             scanf ("%s", &response);
758
759             if ((response == 'Y') || (response == 'y'))
760             {
761                     us_to_foreign = TRUE;
762                     rate = daily_rates.rates.us_foreign[selection];
763             }
764             else
765             {
766                     us_to_foreign = FALSE;
767                     rate = daily_rates.rates.foreign_us[selection];
768             }
769
770             printf("\n");
771
772             if (us_to_foreign == TRUE)  printf ("\n \tEnter US$:  ");
773             else printf ("\n \tEnter %s :", currency[selection]);
774
775             scanf ("%f", &inval);
776             outval = inval * rate;
777
778             printf ("\n");
779             if (us_to_foreign == TRUE)
780                     printf ("\n\t$ %5.2f  US equivalent to: %5.2f  %s",
781                                     inval, outval, currency[selection]);
782
783             else    printf ("\n\t %5.2f %s equivalent to: $ %5.2f US",
784                                     inval, currency[selection], outval);
785     }
>% Display information about this procedure's symbols.
>DESCRIBE inval  response
inval (2 words at +14R8 words) auto float
response (1 byte at +12R8 words) auto char
>% Return to the environment of the main procedure
>ENVIRONMENT @MAIN
:builder
>% Set breakpoints at key statements
>BREAKPOINT 684 ACTION="TYPE rate_rec.rates.us_foreign[selection];
  TYPE rate_rec.rates.foreign_us[selection]"
Set at :builder:684 action="TYPE rate_rec.rates.us_foreign[selection];
  TYPE rate_rec.rates.foreign_us[selection]"
>BREAKPOINT 744 ACTION = "TYPE currency_code:TYPE selection"
Set at :get_selection:744 action="TYPE currency_code:TYPE selection"
>BREAKPOINT 762 ACTION = "LIST 762"
Set at :conversion:762 action="LIST 762"
>BREAKPOINT 767 ACTION = "LIST 767"
Set at :conversion:767 action="LIST 767"
>BREAKPOINT 776 ACTION = "TYPE inval, rate"
Set at :conversion:776 action="TYPE inval, rate"
>BREAKPOINT 799
Set at :main:799
```

Figure C-13. Audit File of the EXCHANGEC SWAT Session (continued)

```
826             {
827                     conversion ();
828
829                     printf ("\n\tEnter Y to continue : ");
830                     scanf ("%s", &response);
831                     printf ("\f");
832
833                     menu ();
834                     get_selection ();
835             }
836
837     printf ("\f\tExchange program ended.");
838     }
>% Display information about this procedure's symbols.
>DESCRIBE selection, ratedata
selection (1 word at 16000011703R8 words) country
ratedata (37 words) typedef
    rate_date (10 bytes at +0R8 bytes) char [10]
    rates (32 words at +5R8 words) struct
      us_foreign (16 words at +0R8 words) float [8]
      foreign_us (16 words at +20R8 words) float [8]
>% Set the environment to that of the procedure called get_selection.
>ENVIRONMENT ^get_selection
:get_selection
>% List this procedure.
>DESCRIBE :get_selection
get_selection (at 16001732543R8 words) extern line 725 to 744 int ()
>LIST 725 744
725     {
726
727     int currency_code;
728
729             scanf ("%d", &currency_code);
730
731             switch  (currency_code)
732             {
733                     case 1 : selection = belgian; break;
734                     case 2 : selection = w_german; break;
735                     case 3 : selection = dutch; break;
736                     case 4 : selection = italian; break;
737                     case 5 : selection = swiss; break;
738                     case 6 : selection = canadian; break;
739                     case 7 : selection = saudi; break;
740                     case 8 : selection = japanese; break;
741
742                     default : selection = no_choice; break;
743             }
744     }
>% Display information about this procedure's symbols.
>DESCRIBE currency_code
currency_code (2 words at +12R8 words) auto int
>% Set the environment to that of the procedure called conversion.
>ENVIRONMENT ^conversion
:conversion
```

Figure C-13. Audit File of the EXCHANGEC SWAT Session (continued)

```
>% The program asked if we are exchanging US dollars to foreign currency.
>% We responded Y.
>TYPE daily_rates.rates.us_foreign[selection]
 2.116500E+00
>CONTINUE

Breakpoint trap at :conversion:776 action="TYPE inval, rate"
 2.500000E+02
 2.116500E+00
>TYPE inval * rate
 5.291250E+02
>% The result of the exchange should be 529.12 Swiss francs.
>CONTINUE

Breakpoint trap at :get_selection:744 action="TYPE currency_code:
  TYPE selection"
6
canadian
>% The program displayed the expected result.  It then presented the menu
>% and we entered 6 (for Canadian dollars).
>CONTINUE

Breakpoint trap at :conversion:767 action="LIST 767"
767C                    rate = daily_rates.rates.foreign_us[selection];
>% The program asked the exchange direction.  We entered N.
>% We will set a breakpoint at 785 so that we do not exit that procedure.
>BREAKPOINT 785
Set at :conversion:785
>CONTINUE

Breakpoint trap at :conversion:776 action="TYPE inval, rate"
 2.500000E+02
 8.014000E-01
>CONTINUE

Breakpoint trap at :conversion:785
>SET inval = 250.
Old value:  2.500000E+02
New Value:  2.500000E+02
>SET us_to_foreign = 1
Old value: 0
New Value: 1
>SET rate = daily_rates.rates.us_foreign[selection]
Old value:  8.014000E-01
New Value:  1.247800E+00
>CONTINUE AT 779

Breakpoint trap at :conversion:785
>CONTINUE

Breakpoint trap at :get_selection:744 action="TYPE currency_code:
  TYPE selection"
0
no_choice


>% We entered 0 to end the program.  If we continue, both the program
>% and SWAT will terminate.  We can also end this session by typing
>BYE

SWAT TERMINATED
```

Figure C-13. Audit File of the EXCHANGEC SWAT Session (concluded)

End of Appendix

Index

Within this index, the letter "f" (or "ff") following a page entry means "and the following page" (or "pages").

& (ampersand) 2-6, 4-2
* (asterisk) 1-3, 4-3
^ (caret) 1-7, 4-4
, (comma) 1-9, 4-2
% (percent sign) 2-7, 3-13, 4-4
; (semicolon) 2-6, 4-2

A

abbreviating a command or keyword 2-7, 4-3
action string, breakpoint 3-16, 4-8
@ALL keyword
  with CLEAR 3-17, 4-12
  with ENVIRONMENT 4-24
  with LIST 3-19, 4-29
ampersand (to continue a command) 2-6, 4-2
AOS/VS SWAT A-2
AT keyword 4-16, 3-23
.AU extension 2-3f, 4-6
AUDIT command 4-6f, 2-3, 3-13, 4-1, 4-4
audit file, 2-3, 3-12ff, 3-23, 4-6f
/AUDIT switch 2-3f, 3-12, 4-4
auditing 2-3f, 3-13, 4-6f, 4-19

B

@BIT keyword 3-24, 4-40f
break file 2-4
@BREAK keyword
  with ENVIRONMENT 3-20f, 4-4, 4-24
  with LIST 4-29
/BREAK switch 2-4
breakpoint
  about 3-14ff, 1-1, 1-8
  clearing 3-17, 4-12
  displaying 3-15, 4-8
  identified in listing 3-18, 3-20f, 4-29
  setting 3-14, 4-8
BREAKPOINT command 4-8ff, 3-14ff, 4-1
breakpoint environment 1-8, 3-19ff
BYE command 4-11, 1-2, 3-31, 4-1

C

C programs 2-2, A-4, C-58ff
calling procedures, listing 4-44
calling SWAT 2-3ff, 1-2, 3-12, 3-32
/CALLS switch 2-4
caret 1-7, 4-4
chaining programs 2-7, 1-2
@CHARACTER keyword 3-24f, 4-40f
check list for SWAT A-1
clause number 1-3, 4-8
CLEAR command 4-12f, 3-17, 4-1
clearing
  action string 4-8
  breakpoint 3-17, 4-12
  proceed count 4-8
CLI command 4-14f, 2-1, 2-4, 3-29, 4-1
CLI operations 1-2, 3-29, 4-14f
COBOL programs 2-2, A-4, C-32ff
command
  abbreviating 2-7, 4-3
  continuing 2-6
  delimiter 4-2
  editing 4-5, A-6
  file 1-2, 3-30, 4-26
  format conventions iv
  list 4-1, 2-6
  screenedit A-6
  separating 2-6, 4-2
comments, audit file 2-7, 3-13, 4-4
compiling your program 2-2
console interrupt 2-7
/CONSOLE switch 2-4f, 2-7, 4-14
CONTINUE command 4-16ff, 3-19, 3-22, 4-1, 4-11
continuing a command line 2-6, 4-2
continuing execution 1-2
control command 2-7, 4-5
conventions, command format iv
copy of program file, debugging 1-2, 2-5
COUNT keyword 3-22, 4-16
/CPU switch 2-4

```
>% Display the breakpoints currently in effect.
>BREAKPOINT
Set at :builder:684 action="TYPE rate_rec.rates.us_foreign[selection];
  TYPE rate_rec.rates.foreign_us[selection]"
Set at :get_selection:744 action="TYPE currency_code;TYPE selection"
Set at :conversion:762 action="LIST 762"
Set at :conversion:767 action="LIST 767"
Set at :conversion:776 action="TYPE inval, rate"
Set at :main:799
>% Begin execution of the program.
>CONTINUE

Breakpoint trap at :main:799
>% The program asked if we wanted to create a new data file; we said N.
>% We can, however, change our response.
>TYPE response
"N"
>SET response = "Y"
Old value: "N"
New Value: "Y"
>CLEAR 799
Cleared at :main:799
>CONTINUE

Breakpoint trap at :builder:684
  action="TYPE rate_rec.rates.us_foreign[selection];
  TYPE rate_rec.rates.foreign_us[selection]"
 1.222220E-76
-9.414063E-01
>% We entered the date and the first exchange rate pair (Belgian francs)
>CONTINUE

Breakpoint trap at :builder:684
  action="TYPE rate_rec.rates.us_foreign[selection];
  TYPE rate_rec.rates.foreign_us[selection]"
 7.183055E-50
 0.000000E+00
>% We entered the next pair of exchange rates (West German marks)
>% We'll clear this breakpoint, then fill in the rest of the table.
>TYPE selection
w_german
>CLEAR 684
Cleared at :builder:684
  action="TYPE rate_rec.rates.us_foreign[selection];
  TYPE rate_rec.rates.foreign_us[selection]"
>CONTINUE

Breakpoint trap at :get_selection:744 action="TYPE currency_code;
  TYPE selection"
2
w_german
>% The program displayed the menu.  We entered code 2 (West German marks)
>% We can change the variable and redirect execution.  This time we'll
>% enter S for Swiss francs.
>SET selection = swiss
Old value: w_german
New Value: swiss
>CONTINUE

Breakpoint trap at :conversion:762 action="LIST 762"
762C                    rate = daily_rates.rates.us_foreign[selection];
```

Figure C-13. Audit File of the EXCHANGEC SWAT Session (continued)

D

/ACL switch 2-4
@DATA file 2-4
Data General, contacting v
/DATA switch 2-4f
data type conversion 1-10, 3-25
DEBUG command 4-19f, 2-5, 3-13, 4-1
/DEBUG switch
  compiling or linking 2-2f, 2-8, A-1
  invoking SWAT 2-4f, 3-13, 4-19
debugger
  lines file 2-3
  symbols file 2-3
debugger, user 1-2, 3-13f, 4-19, A-6
DESCRIBE command 4-21f, 3-27, 4-1
DIRECTORY command 4-23, 3-29, 4-1
display modes 3-24f, 4-40f
displaying
  breakpoint(s) 3-15, 4-8
  calling procedures 4-44f
  expression result 3-25f
  information about a symbol 3-27
.DL extension 2-3
.DS extension 2-3

E

editing a command line 4-5, A-6
EMASM macro 2-1
ending a debugging session 1-2, 3-31, 4-11
environment
  about 1-3ff
  breakpoint 1-8, 3-19ff
  specifier 1-6ff, 3-14, 4-4
  working 1-5ff, 3-14f, 4-24
ENVIRONMENT command 4-24f, 1-5f, 3-14, 4-1
ERMES 2-1
error
  common user A-4f
  handling 2-8
  message 3-28
  message file 2-1
  types 2-8
error code interpretation 3-28, 4-23
examining variables 3-24, 4-40f
EXECUTE command 4-26, 3-30, 4-1
Expression argument 1-8ff
extension
  .AU 2-4f, 4-6
  .DL 2-3
  .DS 2-3
external procedure block 1-6, 4-4

F

fatal error 2-8
file
  audit 2-5, 3-12f, 3-14, 3-23, 4-6f
  debugger lines 2-3
  debugger symbols 2-3
  generic 2-5
  SWAT command 3-30, 4-26
  SWAT Interface 2-2f
@FLOAT keyword 3-24f, 4-40f
FORTRAN 77 programs 2-2f, A-4, C-17ff
fully qualifying a reference 1-7f, 3-14ff

G

generic file 2-5
global switches 2-3f

H

HELP command
  from CLI A-5
  from SWAT 4-27f, 3-27f, 4-1
helpful hints A-5f

I

identifying a program statement 1-3
information about program symbols 3-27, 4-21
@INPUT file 2-4
/INPUT switch 2-4f
installing SWAT 2-1
@INTEGER keyword 3-24f, 4-40f
Interface file, SWAT 2-2f
internal procedure block 1-5f
interpreting error codes 3-28, 4-23
interrupt, console 2-7
invoking SWAT 2-3ff

K

keyword
  @ALL 3-17, 3-19, 4-12, 4-24, 4-29
  AT 4-16, 3-23
  @BIT 3-24, 4-40f
  @BREAK 3-20f, 4-4, 4-24, 4-29
  @CHARACTER 3-24f, 4-40f
  COUNT 3-22, 4-16
  @FLOAT 3-24f, 4-40f
  @INTEGER 3-24f, 4-40f
  @MAIN 3-14, 4-4, 4-24
  @POINTER 3-24f, 4-40f
  @R 3-24f, 4-40f

086-000045-00 updates
093-000258-01

L

label locator 1-3, 1-7f, 4-3
label, numeric 1-3
line number locator 1-3, 1-7f, 4-3
Link 2-2f
linking your program 2-2f
LIST command 4-29ff, 3-17ff, 4-1
@LIST file 2-4
/LIST switch 2-4f
listing source code 3-17ff, 4-29ff
locator 1-3f, 3-14ff, 4-3
log file (See audit file)
lowercase entries 4-2

M

.MAIN 1-6, A-4
@MAIN keyword 1-6, 3-14, 4-4, 4-24
main procedure 1-5, 3-14
manuals, related iii, iv
/MEMORY switch 2-4
MESSAGE command 4-32, 3-28, 4-1
module, program 1-3

N

/NAME switch 2-4
/NOCONSOLE switch 2-4, 2-7f, 3-29, 4-14, 4-19
/NOCOPY switch 2-4, 2-7
null command response 1-2, 3-21, 3-30f, 4-3, 4-34

O

operators in expressions 1-9
original program file, debugging 2-5
@OUTPUT file 2-4
/OUTPUT switch 2-4f
overlays 1-2

P

PARSWAT.SR 2-1
PASCAL programs 2-2f, A-4, C-42ff
percent sign (for comment) 2-7, 3-13, 4-4
PL/I programs 2-2f, A-3, C-1ff
@POINTER keyword 3-24f, 4-40f
precedence of data types 1-10
/PREEMPTIBLE switch 2-4
PREFIX command 4-33, 3-12, 4-1, 4-2
preparing a program for debugging 2-2f
/PRIORITY switch 2-4
privileges, user 2-1
procedure block
  about 1-3, 1-5f, 4-4
  calling sequence 4-44
  unnamed 3-16, 4-4
proceed count 3-16, 3-22, 4-8, 4-16
profile, user 2-1, 4-14, A-2
program
  compilation 2-2
  execution 4-16
  flow 1-2, 3-23, 4-16
  linking 2-2f
  module 1-3
  revision 1-2
  samples 3-1ff, C-1ff
PROMPT command 4-34, 3-21, 3-30f, 4-1, 4-3
prompt, SWAT 2-5, 3-12, 4-2, 4-33

R

@R keyword 3-24f, 4-40f
radix symbol 3-26
release notice 2-1, A-2
required software 2-1
/RESIDENT switch 2-4
revising a source file 1-2
root environment 1-6
running a program 4-16
running SWAT 2-3ff, 3-12, 3-32

S

sample programs 3-1ff, C-1ff
scope 1-5ff, 4-24
screenedit commands, using A-6
search list 1-2, 3-29, 4-35
SEARCHLIST command 4-35f, 3-29, 4-1
semicolon 2-6, 4-2
separating commands 2-6, 4-2
SET command 4-37ff, 3-26, 4-1
setting
  breakpoint 3-14, 4-8
  directory 3-29, 4-23
  environment 4-24f, 1-5, 3-14
  search list 3-29, 4-35
  value of variable 3-26, 4-37
soft error 2-8
software requirements 2-1
/SONS switch 2-4, 4-14
source code
  listing 3-17ff
  revising 1-2
starting a debugging session 2-3ff, 1-2, 3-12, 3-32
statement identifier (See locator.)
SWAT
  commands 4-1ff
  concepts 1-2
  features 1-1f
  prompt 1-2, 2-6, 3-12, 4-2, 4-33
  user privileges 2-1
SWAT Interface file 2-2f
SWATERMES.OB 2-1
SWATI 2-2f, A-2
SWAT16 2-2f,4-4, A-2

witches, list of global 2-4
ymbol,
  description of 3-27, 4-21f
  displaying value of 3-24ff, 4-40f
  setting value of 3-26, 4-37ff
Symbolic Debugger (See user debugger.)

T

terminating a SWAT session 1-2, 3-31, 4-11
topic, SWAT 1-1, 3-27, 4-27
trap, breakpoint 1-1, 3-14, 3-19
TYPE command 4-40ff, 3-24ff, 4-1

U

unnamed procedure block 3-16, 4-4
uppercase entries 4-2
user debugger 1-2, 3-13f, 4-19, A-6
/USERNAME switch 2-4
using upper- and lowercase 4-2

V

variable
  examining 3-24ff, 4-40ff
  setting the value of 3-26, 4-37

W

WALKBACK command 4-44f, 4-1
working directory 1-2, 3-29, 4-23
working environment 1-5ff, 3-14f, 4-24
/WSMAX switch 2-4
/WSMIN switch 2-4

086-000045-00 updates
093-000258-01