IBM

VS FORTRAN Version 2 Interactive Debug Guide and Reference

Licensed Program

Order Number
SC26-4223-1

Program Numbers
5668-805
5668-806

Release 2

VS FORTRAN Version 2 Interactive Debug Guide and Reference

Licensed Program

Order Number
SC26-4223-1

Program Numbers
5668-805
5668-806

Release 2

Second Edition (June 1987)

This edition replaces and makes obsolete the previous edition, SC26-4223-0.

This edition applies to Release 2 of VS FORTRAN Version 2, Licensed Programs 5668-805 and 5668-806, and to any subsequent releases until otherwise indicated in new editions or technical newsletters. The changes for this edition are summarized under "Summary of Changes" following the preface. Specific changes are indicated by a vertical bar to the left of the change. These bars will be deleted at any subsequent publication of the page affected. Editorial changes that have no technical significance are not noted.

Changes are made periodically to this publication; before using this publication in connection with the operation of IBM systems, consult the latest *IBM System/370, 30xx, and 4300 Processors Bibliography*, GC20-0001, for the editions that are applicable and current.

References in this publication to IBM products, programs, or services do not imply that IBM intends to make these available in all countries in which IBM operates. Any reference to an IBM licensed program in this publication is not intended to state or imply that only IBM's program may be used. Any functionally equivalent program may be used instead.

Requests for IBM publications should be made to your IBM representative or to the IBM branch office serving your locality. If you request publications from the address given below, your order will be delayed because publications are not stocked there.

A form for readers' comments is provided at the back of this publication. If the form has been removed, comments may be addressed to IBM Corporation, P.O. Box 50020, Programming Publishing, San Jose, California, U.S.A. 95150. IBM may use or distribute whatever information you supply in any way it believes appropriate without incurring any obligation to you.

Preface

About This Book

Part 1. VS FORTRAN Version 2 Interactive Debug Guide

The guide contains the following information on how to use Interactive Debug:

- How to set up and use Interactive Debug in your environment
- A sample debugging session
- How to use some common commands
- Special considerations when using Interactive Debug

Part 2. VS FORTRAN Version 2 Interactive Debug Reference

The reference section contains:

- An explanation of syntax and statement identifier conventions
- A table listing the Interactive Debug commands according to the functions they perform
- Detailed descriptions of all the Interactive Debug commands

Appendixes

The appendixes include:

- Information on using the Interactive Debug HELP facility
- A summary of Interactive Debug commands, illustrating syntax and function
- A listing of Interactive Debug messages

Summary of Changes

Release 2, June 1987

Major Changes to the Product

- Support for 31-character symbolic names, including the underscore (_) character.
- Addition of a program sampling capability, to assist in identifying areas of a program that use the most CPU time.
- Addition of four new commands:
  1. ANNOTATE, to support program sampling.
  2. LISTSAMP, to support program sampling.
  3. LISTINGS, to display the listings data set specification panel when using ISPF Version 2.
  4. RECONNECT, to allow a CLOSED unit to be reset to its original (preconnected) state so that it can be implicitly opened again.
- Capability of providing substring notation for character variables.
- Capability of referencing array subscripts outside the range of the declared dimensions.
- Reduced dependency on TSO, including removal of the TSOLIB requirement under CMS batch.
- Capability to debug programs invoked under TSO LOADGO.
- Support for debugging CMS MODULE files under ISPF.

Contents

Figures

Part 1. VS FORTRAN Version 2 Interactive Debug Guide

VS FORTRAN Version 2 Interactive Debug provides a set of commands to help you locate and diagnose problems in your VS FORTRAN Version 1 or VS FORTRAN Version 2 programs. ("VS FORTRAN" is used in this book as a common term to refer to programs written with either product.)

This section of the book contains instructional material to guide you through the process of learning those commands.

For a summary of VS FORTRAN Version 2 Interactive Debug commands, including descriptions, keywords, and abbreviations, see Appendix B, "Interactive Debug Command Summary" on page 225.

If you already know how to use Interactive Debug, you may want to go directly to the reference section in "Part 2. VS FORTRAN Version 2 Interactive Debug Reference" on page 111.

Chapter 1. Introducing VS FORTRAN Version 2 Interactive Debug

VS FORTRAN Version 2 Interactive Debug is a flexible, efficient tool that assists you in monitoring the execution of VS FORTRAN Version 1 and VS FORTRAN Version 2 programs. (For convenience, we refer to both kinds of programs as *VS FORTRAN programs*, unless special information applies to only one product.)

With VS FORTRAN Interactive Debug, you can suspend program execution, analyze program performance, continue execution, skip sections of code, correct errors, display and set variables, set up expected input, and display output.

You can debug your programs in full screen mode, line mode, or batch mode. For full screen mode, the Interactive System Productivity Facility (ISPF) is required. With ISPF Version 2, you can view your source listing in a portion of the screen called a source listing window. You can control the pace of execution, and highlight the command currently executing.

VS FORTRAN Interactive Debug can be used easily and effectively by scientists, engineers, other professionals, and students—all those who use VS FORTRAN for engineering and scientific problem solving.

Application programmers using VS FORTRAN will also find VS FORTRAN Interactive Debug to be an essential aid to increased productivity. Through improved tracing, reduced dump analysis, and fewer recompilations, measurable savings in time and effort can be realized.

Interactive Debug Features

VS FORTRAN Version 2 Interactive Debug offers versatile capabilities for VS FORTRAN Version 1 or VS FORTRAN Version 2 program debugging.

VS FORTRAN Version 2 Interactive Debug provides the following features:

An easy invocation procedure: Usually, you simply specify the DEBUG option when you invoke the program to be debugged. This reduces the need for advance planning, and also allows you to debug previously compiled programs with minimal effort.

31-character names: You can use symbolic names up to 31 characters in length, including the underscore (_) character, for both local and global names.

Online HELP information: Information is provided for all VS FORTRAN Version 2 Interactive Debug commands and functions and is available under Time Sharing Option (TSO), Conversational Monitor System (CMS), and ISPF. By entering

HELP (or by using an equivalent PF key), you will be presented with the first of a set of screens containing HELP information.

Full screen display: VS FORTRAN Version 2 Interactive Debug offers full screen support. Using the facilities of ISPF and the Program Development Facility (PDF), you can split your screen to perform debugging on one section of the screen, while editing or browsing your program or files on the other section.

Source listing window and animation: Using ISPF Version 2, you can view your source listing in an area of the screen called a source listing window. You can highlight the command currently executing, and control the pace of execution. This capability is called program animation.

Ability to issue AT, OFF, and LIST with PF keys: When you use a source listing window, you can assign the AT, LIST, and OFF commands to a PF key, and then perform many tasks by "pointing" with the cursor, instead of typing identifiers on the command line. This allows you to use Interactive Debug without knowing complex syntax or remembering command names.

Selective debugging of program units: An optional execution-time control file lets you specify which program units will be debugged, and which statements in those program units will have debugging hooks. Those not selected for debugging will run at normal speed.

Manipulation of external files while debugging: Commands that are similar to VS FORTRAN Version 2 I/O statements (for example, ENDFILE, BACKSPACE, CLOSE, and REWIND) allow you to manipulate external sequential files. Also, while remaining in debug mode, you can browse or edit external sequential files used by the program being debugged.

Ability to issue system commands while debugging: Without terminating your debug session, you can issue commands at the system level.

On MVS/XA, ability to debug programs in either addressing mode: Programs that run in 31-bit addressing mode and reside either above or below the 16-megabyte line can be debugged.

Debugging of optimized code: Programs compiled at any optimization level can be debugged (with restrictions).

Debugging of vectorized code: Programs compiled at any vectorization level can be debugged (with restrictions).

Debugging of reentrant code: Programs compiled with the RENT option and run in reentrant mode can be debugged. (For full debugging function, the TEST option is also required.)

Using sequence numbers instead of ISNs for source statements: Programs compiled with the SDUMP (SEQ) option will generate the SDUMP statement table using the sequence numbers you supply in columns 73 through 80. This makes it possible for you to debug programs using those numbers instead of ISNs.

Program sampling: This capability enables you to obtain an activity analysis of an application program, identifying areas that take the most CPU time to execute.

This information can assist you in directing your performance improvement activities to where they can provide the most gain in efficiency.

Interactive Debug Programming Requirements

The operating system, compiler, and library requirements for Interactive Debug are listed in the following sections.

System Requirements

Execution with VS FORTRAN Version 2 Interactive Debug is supported under the following operating systems:

MVS/SP Version 1—All Releases (with or without TSO/E)
MVS/SP Version 2 (MVS/XA)—All Releases
MVS/XA DFP—Version 1 or Version 2 (with or without TSO/E)
VM/SP—Release 4 and later (with or without VM/SP HPO Release 4 or later)
VM/XA System Facility—Release 2 and later

VS FORTRAN Version 2 Interactive Debug operates in line or batch mode. It also operates in full screen mode with the Interactive System Productivity Facility (ISPF), and optionally, the Program Development Facility (PDF). Use of Interactive Debug in full screen mode, including support for the source listing window, animation, and color, requires the following:

ISPF Version 2 Release 1 or later

If your installation does not have PDF, you must follow special invocation procedures, as described in "Invoking IAD without PDF" on page 13, or "Invoking IAD for TSO without PDF" on page 18.

Note that VS FORTRAN Version 2 Release 2 will run in the VM/SP operating system only with the OS simulation facility. It cannot run with the DOS simulation capability of VM/CMS active.

Interactive debug supports the same operating systems as the compiler; however, TSO/E is required when running interactively on MVS or MVS/XA, or if TSO commands are issued in batch mode.

Compiler Requirements

VS FORTRAN Version 2 Interactive Debug is designed for use with programs compiled by VS FORTRAN Version 2, and by VS FORTRAN Version 1 Release 2 or later.

The TEST compiler option is required for program units compiled by VS FORTRAN Version 1 Release 2, and for those that are compiled by VS

FORTRAN Version 1 Release 3 if running on MVS/XA. It is also required for RENT program units in protected storage.

To be eligible for deugging, program units must be compiled with the SDUMP option in effect.

Library Requirements

All programs to be debugged with VS FORTRAN Version 2 Interactive Debug must be link-edited (or, under CMS, loaded) with the VS FORTRAN Version 2 Library, or with the Release 3.1, 4, or 4.1 VS FORTRAN Version 1 Library (5748-LM3).

VS FORTRAN modules link-edited with the Release 3.1 VS FORTRAN Library will be debuggable in the MVS/XA environment only when linked (or re-linked) to override the addressing and residency modes to 24/24.

Batch-mode debugging is only available if the program is using the VS FORTRAN Version 2 library.

In general, the VS FORTRAN library must be at least at the same release level as the compiler used for the program units. In addition, VS FORTRAN Version 2 Interactive Debug requires at least the VS FORTRAN Version 1 Release 3.1 library.

Restrictions

MVS/XA: On an MVS/XA system, using VS FORTRAN Version 2 Interactive Debug to debug VS FORTRAN programs running in 31-bit addressing mode requires that those programs be linked with Release 4 or 4.1 of the VS FORTRAN Version 1 Library, or any release of the VS FORTRAN Version 2 Library. Programs linked with the VS FORTRAN Version 1 Release 3.1 Library must specify an AMODE and RMODE of 24 in order to be debugged.

Shared Storage: VS FORTRAN reentrant programs that will reside in a shared area (DCSS or LPA) must have been compiled with the TEST option in order to be debuggable. Reentrant VS FORTRAN Version 2 programs can be debugged in user storage with or without the TEST option.

Overlays: VS FORTRAN Version 2 Interactive Debug cannot debug programs that use overlays.

Multi Tasking Facilities: If you are using the Multitasking Facility (MTF), only the main task can be debugged.

Performance Considerations

For programs compiled with VS FORTRAN Version 1 Release 3 or later, or with any release of VS FORTRAN Version 2, VS FORTRAN Version 2 Interactive Debug can be invoked merely by specifying the DEBUG option at execution time (so long as SDUMP was in effect for the compilation). There is no need to have compiled with the TEST option.

However, there are two exceptions:

- A reentrant program in a shared area (DCSS or LPA) requires the TEST option in order to be debuggable; and
- On an MVS/XA system, programs compiled by Release 3 of VS FORTRAN Version 1 without TEST will need to be recompiled with Release 3.1 or 4, or with VS FORTRAN Version 2, in order to be debuggable. (Programs compiled by VS FORTRAN Version 1 Release 3 with TEST, however, can be debugged under MVS/XA.)

VS FORTRAN Version 2 Interactive Debug permits debugging of optimized and vectorized code, but you need to be aware of the effects of optimization and vectorization to correctly interpret the debugging results. Whenever a program unit has been compiled with an optimization or vectorization level other than 0, Interactive Debug will issue a warning message indicating that the results of some debugging commands are unpredictable.

VS FORTRAN Version 2 Interactive Debug requires about 300K bytes of storage to begin execution (plus the storage required to load the program that will be debugged). Interactive Debug also acquires additional dynamic storage during execution. The amount varies according to the nature of the program being debugged, and the type and quantity of debugging commands issued.

Discovering Program Bugs

VS FORTRAN Version 2 Interactive Debug helps find program bugs by giving you control over program execution. You can use it to debug VS FORTRAN programs in an easy-to-use, conversational manner. You can:

- Start and stop the program
- Examine and change values of variables, arrays, and array elements
- Trace program transfers
- Track execution frequency of statements
- Control the action taken for execution errors
- Receive online help at the terminal
- Manipulate external files

- Locate errors, fix the problem, and continue debugging

VS FORTRAN Version 2 Interactive Debug allows you to place the output from some of the commands in a print data set for later examination. In addition, when executing under ISPF or in batch mode, a log of the debug session is placed in a data set. This data set can subsequently be used as input to Interactive Debug to re-create a previous debugging session.

Invoking Interactive Debug

The DEBUG execution-time option causes the VS FORTRAN Version 2 library initialization routines to load VS FORTRAN Version 2 Interactive Debug.

The format of the execution command depends on the particular environment in which the program is being run.

- In ISPF with PDF (under either CMS or TSO), a panel is provided for executing a VS FORTRAN program and passing it the DEBUG option.
- In CMS, there are several ways to invoke Interactive Debug. For example, entering the following commands will execute the program with the DEBUG option:

  ```
  LOAD progname
  START * DEBUG
  ```

- In TSO, the format for invoking VS FORTRAN Version 2 Interactive Debug is:

  ```
  CALL progname 'DEBUG'
  ```

  or

  ```
  LOADGO progname 'DEBUG'
  ```

In these examples, `progname` is the name of the program to be executed.

When a VS FORTRAN program is dynamically invoked from another program, the VS FORTRAN program can be debugged by passing DEBUG as the execution-time parameter, or by including a special object module. This is explained more fully in "Specifying Default Execution-Time Options" on page 96.

Chapter 2. Using Interactive Debug with ISPF

When VS FORTRAN Version 2 Interactive Debug runs with ISPF in either a CMS or a TSO environment, we say it is executing in *full screen mode* because the entire screen is available. In contrast, when Interactive Debug reads and writes one line at a time, we say it is executing in *line mode*.

When executing in an ISPF environment, VS FORTRAN Version 2 Interactive Debug will run in full screen mode using ISPF panels and services. You can enter Interactive Debug commands, and view both the commands and output on your terminal in the *scrollable log*.

Using ISPF Version 2, you can view your source listing in an area of the screen called the *source listing window*. You can also choose to look at the source file by splitting the physical screen into two logical screens. Using a *split screen*, you can browse or edit the source file (or, under MVS, even recompile) while debugging. The IBM Program Development Facility (PDF) is required for the browse and edit functions in split-screen mode.

Invoking Interactive Debug

During installation of VS FORTRAN Version 2 Interactive Debug, modifications should have been made to allow you to debug a VS FORTRAN program under ISPF using standard procedures. PDF is required for the foreground invocation panel supplied with Interactive Debug.

After you invoke ISPF/PDF, you will be presented with the Primary Option Menu. Select option 4, FOREGROUND PROCESSING. When the next panel appears, select the number of the option associated with VS FORTRAN Version 2 Interactive Debug. (This is likely to be option 11.)

You can also proceed directly to the Interactive Debug panel by typing `4.11` on the command line of the Primary Option Menu.

You will then be presented with the FOREGROUND VS FORTRAN Version 2 Interactive Debug panel. See Figure 1 on page 10 for CMS, and Figure 6 on page 17 for TSO.

Under CMS

If you are using CMS, you will see the panel shown in Figure 1. This panel is similar to other invocation panels accessible from the FOREGROUND PROCESSING panel under CMS.

```
-------     FOREGROUND VS FORTRAN VERSION 2.2.0 INTERACTIVE DEBUG     --------
COMMAND ===>

ISPF LIBRARY:
   PROJECT ===>
   LIBRARY ===>
   TYPE    ===>
   MEMBER  ===>               (Blank for member selection list)

CMS FILE:
   FILE ID    ===>                        (TEXT or MODULE file or LOADLIB  )
   MEMBER     ===>            (If member of a LOADLIB)
   IF NOT LINKED, SPECIFY:
   OWNER'S ID ===>           DEVICE ADDR. ===>      LINK ACCESS MODE ===>

READ PASSWORD ===>

EXECUTION TIME OPTIONS:
   DEBUG  ===>                (Enter DEBUG or NODEBUG)
   OTHER  ===>


SYSLIB TXTLIB:   (VSF2FORT, TSOLIB and CMSLIB already specified)
          ===>          ===>          ===>          ===>
```

Figure 1. The FOREGROUND VS FORTRAN Version 2 Interactive Debug Panel for CMS

Fill in either the ISPF LIBRARY fields or the CMS FILE fields (and, if necessary, the READ PASSWORD field) before pressing the ENTER key. (If you are not already familiar with ISPF libraries, you should probably use the CMS FILE fields.) If both fields are filled in, the CMS FILE fields will be read and the ISPF LIBRARY fields will be ignored.

You can process programs that have been loaded as TEXT files, MODULE files, or as members of a LOAD LIBRARY, as follows:

- For a TEXT file, fill-in the FILE ID line with the filename optionally followed by a filetype of TEXT. Leave the MEMBER line blank.
- For a MODULE file, fill-in the FILE ID line with the filename followed by a filetype of MODULE. Leave the MEMBER line blank.
- For a LOAD LIBRARY member, fill-in the FILE ID line with the filename and do not specify a filetype. Fill-in the MEMBER line with the library member name of your program.

The FILE ID or MEMBER line is also used to build file names for several Interactive Debug files described later in this chapter. These file names are:

fname INCLUDE *	(for AFFON)
fname LIST A	(for AFFPRINT)
fname LOG A	(for AFFOUT)
fname RESTART *	(for AFFIN)
fname LISTING *	(default listing name)

If you do not provide the file ID, the ISPF library member name will be used. Use the READ PASSWORD field to specify a required password for the disk access.

In all cases, you must enter DEBUG under EXECUTION OPTIONS to invoke Interactive Debug. If you specify NODEBUG, Interactive Debug will not be invoked unless you include a special object module to override the default. For further information about overriding the default options, see "Specifying Default Execution-Time Options" on page 96.

In addition to the DEBUG parameter, you can also specify other execution-time parameters in the OTHER field. You can wrap the list of other parameters onto the second line if necessary. Use commas with no embedded blanks to separate parameters. You must also use commas with no embedded blanks **within** individual parameters, such as the DEBUNIT option.

For an explanation of the possible parameters, enter HELP. From the main HELP panel, select option 2 for a description of the parameters and the IBM supplied default values. For more detailed information about the parameters, see *VS FORTRAN Version 2: Programming Guide*.

The SYSLIB TXTLIB fields allow you to enter up to four CMS TXTLIBs. These may be accessed in addition to VSF2FORT whenever CMS determines that it needs to search a TXTLIB (for example, when loading a TEXT file and resolving all the external references).

The VS FORTRAN Version 2 Library is capable of running in either of two "modes." If you choose to have the necessary library routines included within your executable program, you are operating in *link mode*. If, on the other hand, you choose to have the library routines loaded during execution of your program, you are operating in *load mode*. You make the choice of link mode or load mode by making the appropriate combination of libraries available when you create your executable program from your TEXT files.

VS FORTRAN Version 2 Interactive Debug's ISPF invocation EXEC is set up to execute TEXT files using load mode. If you want your program to operate in link mode, you should specify VSF2LINK as an additional TXTLIB on the bottom line of this invocation panel.

For more information on using link and load mode with the VS FORTRAN Version 2 Library, see *VS FORTRAN Version 2: Programming Guide*.

When you use the ISPF panel to start debugging a TEXT file, you can specify only a single TEXT file (program name). Should you be debugging a program that calls other programs, the calls will be automatically resolved for you — provided the called programs are either in a TXTLIB (specified on the panel as a SYSLIB TXTLIB) or referenced in the calling program by their file names. (The file names must be the same as the SUBROUTINE or FUNCTION names.) If your called programs are not in a SYSLIB TXTLIB or referenced by their file names, you will need to follow a different procedure. Here are three alternative methods:

Loading from a LOADLIB

Before invoking ISPF, place your main program and all its called programs in a LOADLIB. You can use the CMS LKED command to build a LOADLIB.

The input to LKED is a file containing any combination of text files and LKED control statements. (LKED control statements are the same as those accepted by the MVS linkage editor.) For example, you could create a file of control statements such as:

```
INCLUDE MYPROG
INCLUDE SUB1
INCLUDE MYMATH
ENTRY MAIN
NAME MYPROG(R)
```

Assume this file is named "MYFILE TEXT." Each control statement must be preceded by at least one blank.

Now, enter the FILEDEF and LKED commands for your LOADLIB; for example:

```
FILEDEF SYSLIB DISK VSF2FORT TXTLIB fm
LKED MYFILE (MAP
```

This produces a file named "MYFILE LOADLIB," which contains a link-edited program named MYPROG. It also produces a listing file named "MYFILE LKEDIT," which contains a map of your link-edited program.

Specify the LOADLIB name and member name in the FOREGROUND VS FORTRAN Version 2 Interactive Debug panel.

Loading a concatenated TEXT file

Alternatively, you can concatenate all the TEXT files into a single TEXT file. Use the CMS `COPYFILE` command with the APPEND option, or an editor such as XEDIT or the ISPF/PDF editor, to create a composite TEXT file. Then, specify this composite TEXT file in the FOREGROUND VS FORTRAN Version 2 Interactive Debug Panel.

Loading a module file

If you have generated a MODULE file containing your VS FORTRAN program, you can specify the module filename and filetype on the ISPF panel. However, the module must be generated with a higher origin than the default in order to leave room for the invocation program. (ORIGIN 22000 will probably be high enough.) If the module origin is too low, the message

```
MODULE ORIGIN TOO LOW
```

will appear at the upper right of the panel.

Invoking IAD without PDF

If you want to use Interactive Debug without PDF on VM/SP, you must complete the following steps:

1. Modify the ISPF invocation EXEC to include FILEDEF statements for VSF2PLIB and VSF2MLIB.

2. Modify the ISPF Master Application Menu to include an option for Interactive Debug. The panel must also be modified to call an application panel created by your installation (see step 3).

3. Create an application panel that prompts for the name of your text file and then invokes an EXEC created by your installation (see step 4).

4. Create an EXEC to execute the VS FORTRAN program and pass the DEBUG parameter, and any other parameters you need. (Note: EXEC2 or REXX is recommended for writing the EXEC to avoid tokenization problems.)

Each of these steps is described in more detail in the following sections.

Modify the ISPF Invocation EXEC: Figure 2 on page 14 shows how you might modify the ISPF invocation EXEC to include FILEDEF information for VSF2PLIB, VSF2MLIB, and IADCMDS table.

Before you invoke this EXEC, you must issue FILEDEFs for any additional panel, message, table, or skeleton libraries from which you plan to operate. You must also issue a FILEDEF for the MACLIB to be used for ISPPROF.

```
*
&TRACE  OFF
*
FILEDEF ISPPLIB CLEAR
FILEDEF ISPMLIB CLEAR
FILEDEF ISPPLIB DISK VSF2PLIB MACLIB * (PERM CONCAT
FILEDEF ISPPLIB DISK ISPPLIB MACLIB * (PERM CONCAT
*
FILEDEF ISPMLIB DISK VSF2MLIB MACLIB * (PERM CONCAT
FILEDEF ISPMLIB DISK ISPMLIB MACLIB * (PERM CONCAT
*
FILEDEF ISPTLIB DISK IADCMDS TABLE * (PERM CONCAT
FILEDEF ISPTLIB DISK ISPTLIB MACLIB * (PERM CONCAT
*
FILEDEF ISPPROF DISK DEFAULTS MACLIB A
*
ISPDCS  ISPDCSS ISPVM  PANEL(ISP@MSTR) &ARGSTRING
```

Figure 2. A Sample Modification of the ISPF Invocation EXEC

Modify the ISPF Master Application Menu: Figure 3 shows how you might modify the ISPF master application menu to provide an option for Interactive Debug (here, option 2) and invoke a prompt panel (here, called USERISP).

```
%-----------------------  ISPF MASTER APPLICATION MENU  ------------
%OPTION  ===>_ZMCD                                        %     +USERID   - &ZUSER
%   1 +SAMPLE1     - Sample application 1           +TIME     - &ZTIME
%   2 +VSF IAD     - VS FORTRAN Interactive Debug   +TERMINAL - &ZTERM
%   3 +.           - (Description for option 3)     +PF KEYS  - &ZKEYS
%   4 +.           - (Description for option 4)
%   5 +.           - (Description for option 5)
%   X +EXIT        - Terminate ISPF using list/log defaults
%
+Enter%END+command to terminate ISPF.
%
)INIT
  .HELP    = ISP00005     /* Help for this master menu            */
  &ZPRIM   = YES          /* This is a primary option menu        */
)PROC
  &ZSEL = TRANS( TRUNC (&ZCMD,'.')
                1,'PANEL(ISP@PRIM)' /* Sample primary option menu */
                2,'PANEL(USERISP)'   /* SAMPLE PANEL TO INVOKE IAD */
             /*****************************************************/
             /*                                                   */
             /* Add other applications here                       */
             /*                                                   */
             /*****************************************************/
               ' ',' '
                X,'EXIT'
                *,'?' )
)END
```

Figure 3. Sample Modifications to ISPF Master Application Menu

Create an Application Panel: You must create an application panel at your installation similar to the one shown in Figure 4. The sample prompts the user for the name of the text file, and then invokes an EXEC called USERDBG.

```
)ATTR DEFAULT(%+_)
      /*  % TYPE(TEXT) INTENS(HIGH)      defaults displayed for      */
      /*  + TYPE(TEXT) INTENS(LOW)       information only            */
      /*  _ TYPE(INPUT) INTENS(HIGH) CAPS(ON) JUST(LEFT)             */
  $ TYPE(INPUT) INTENS(LOW) PAD(_)    /* input field padded with '_' */
  ! TYPE(INPUT) INTENS(LOW) PAD(' ')  /* input field padded with ' ' */
)BODY
%--------------------  SAMPLE VSF IAD PANEL -----------------------------------
%COMMAND ===>_ZCMD                                                          %
+   ENTER THE NAME OF YOUR TEXT FILE BELOW...
+   FILE ID    %===>_NAME             + (TEXT or MODULE file or LOADLIB)
+   MEMBER     %===>!MEMBER  +  (if a member of LOADLIB)
+   ENTER THE EXECUTION TIME OPTIONS BELOW...
+   OPTIONS ===>!FDEBUG

)PROC
 VPUT (NAME,MEMBER,FDEBUG) PROFILE
 &ZSEL='CMD(USERDBG)'
)END
```

Figure 4. Sample Application Panel (USERISP) to Prompt for the Text File Name

Create an EXEC to Pass the DEBUG Parameter: We recommend that you use EXEC2 or REXX when creating the EXEC to avoid tokenization problems. The sample EXEC in Figure 5 on page 16 uses EXEC2 to invoke AFFLOADF, which in turn loads the VS FORTRAN program with the DEBUG parameter. For this example, we have called the EXEC "USERDBG."

```
      &TRACE OFF
      MAKEBUF
      &PRESUME  &SUBCOMMAND ISPEXEC
      &COMMAND GLOBAL TXTLIB VSF2FORT
      &USRLDLIB = &BLANK
      &TYPE = &BLANK
      ISPEXEC VGET (NAME,MEMBER,FDEBUG)
      &IF /&MEMBER = /&BLANK &GOTO -PARSE
      &USRLDLIB = &NAME
      &NAME = &MEMBER
      &COMMAND STATE &USRLDLIB LOADLIB *
      &IF &RETCODE ¬= 0 &GOTO -EXIT
      &COMMAND FILEDEF USRLB DISK &USRLDLIB LOADLIB *
      &GOTO -AFFFILE
-PARSE
      &IF /&NAME = /&BLANK &GOTO -EXIT
      &L = &LOCATION OF MODULE &NAME
      &IF &L = 0 &GOTO -TXTSTATE
      &LL = &L - 2
      &NAME = &LEFT OF &NAME &LL
      &TYPE  = MODULE
      &GOTO -AFFFILE
-TXTSTATE
      &COMMAND STATE &NAME TEXT *
      &IF &RETCODE ¬= 0 &GOTO -EXIT
-AFFFILE
      &COMMAND ERASE &NAME LOG A
      &COMMAND ERASE &NAME LIST A
      &COMMAND STATE &NAME RESTART *
      &IF &RETCODE =  0 &COMMAND FILEDEF AFFIN DISK &NAME RESTART *
      &IF &RETCODE ¬= 0 &COMMAND FILEDEF AFFIN DUMMY
      &COMMAND STATE &NAME INCLUDE *
      &IF &RETCODE =  0 &COMMAND FILEDEF AFFON DISK &NAME INCLUDE *
      &IF &RETCODE ¬= 0 &COMMAND FILEDEF AFFON DUMMY
      &COMMAND FILEDEF AFFOUT   DISK &NAME LOG  A (DISP MOD
      &COMMAND FILEDEF AFFPRINT DISK &NAME LIST A (DISP MOD
-RESTART
      &COMMAND LOAD DMSZIT (CLEAR
      &COMMAND LOADMOD MOVEFILE
      &STACK GLOBAL TXTLIB  VSF2LINK VSF2FORT CMSLIB TSOLIB
      &STACK GLOBAL LOADLIB &USRLDLIB VSF2LOAD
      &STACK &USRLDLIB
      &STACK &NAME &TYPE
      ISPEXEC SELECT PGM(AFFLOADF) PARM(&FDEBUG ) NEWPOOL NEWAPPL(IAD)
      &COMMAND SENTRIES
      &IF &RETCODE = 1 &READ VARS &RST
      &IF .&RST ¬= .RESTART &GOTO -EXIT
      &GOTO -RESTART
-EXIT
      &COMMAND DROPBUF
      &EXIT
```

Figure 5. Sample EXEC (USERDBG) to Invoke the Program without PDF

Under TSO

TSO users will see the ISPF/PDF panel shown in Figure 6. This panel is similar to other invocation panels accessible from the FOREGROUND panel under TSO. Fill in either the ISPF LIBRARY fields or the OTHER PARTITIONED DATA SET fields before pressing the ENTER key. If both fields are filled in, the OTHER PARTITIONED DATA SET fields will be read and the ISPF LIBRARY fields will be ignored.

If the OTHER PARTITIONED DATA SET line is filled in, you should specify both the data set and member names. If you do not specify the member name, you will see a panel with a list of member names. You then choose the member name you need. Specify the data set and member names as follows:

```
'data.set.name(member)'
```

Note: If you omit the quotation marks, ISPF will supply the data set prefix specified as the default in your TSO profile.

```
---------------  FOREGROUND VS FORTRAN VERSION 2.2.0 INTERACTIVE DEBUG -------
COMMAND ===>

ISPF LIBRARY:
   PROJECT ===>
   LIBRARY ===>
   TYPE    ===>
   MEMBER  ===>                  (Blank for member selection list)

OTHER PARTITIONED DATA SET:
   DATASET NAME  ===>

FILE ID FOR DEBUG FILES
           ===>                              PASSWORD ===>

EXECUTION TIME OPTIONS:
   DEBUG   ===>                (Enter DEBUG or NODEBUG)
   OTHER   ===>
```

Figure 6. The FOREGROUND VS FORTRAN Version 2 Interactive Debug Panel for TSO

Use the FILE ID FOR DEBUG FILES line to specify a file ID to be used to build data set names for the Interactive Debug data sets described later in this chapter. These data set names are:

userid.fileid.INCLUDE	(for AFFON)
userid.fileid.PRINT	(for AFFPRINT)
userid.fileid.LOG	(for AFFOUT)
userid.fileid.RESTART	(for AFFIN)
userid.fileid.LIST	(default listing name)

If you do not provide the file ID, the ISPF library member name will be used. Use the PASSWORD field to specify a password for any password-protected data sets.

In all cases, you must enter DEBUG under EXECUTION OPTIONS to invoke Interactive Debug.

If you do not specify DEBUG, Interactive Debug will not be invoked unless you include a special object module to override the default. (For further information about overriding the default options, see "Specifying Default Execution-Time Options" on page 96.)

In addition to the DEBUG parameter, you can also specify other execution-time parameters in the OTHER field. You can wrap the list of other parameters onto the second line if necessary. Use a comma to separate consecutive parameters. For an explanation of the possible parameters, enter HELP. From the main HELP panel, select option 2 for a description of the parameters and the IBM supplied default values. For more detailed explanations of the parameters, see *VS FORTRAN Version 2: Programming Guide.*

Invoking IAD for TSO without PDF

If you want to use Interactive Debug without PDF under TSO, you must complete the following steps:

1. Define the necessary data sets for running ISPF and IAD.
2. Modify the ISPF Master Application Menu to include an option for Interactive Debug. The panel must also be modified to call an application panel created by your installation (see step 3).
3. Create an application panel that prompts for the name of your load module and then invokes a CLIST created by your installation (see step 4).
4. Create a CLIST to execute the VS FORTRAN program and pass the execution-time options.

Each of these steps is described in more detail in the following sections.

Defining the Necessary Data Sets: Modify your allocations for the ISPF data sets to include allocations for the IAD panel library, message library and command table. Examples of lines which could be included in a CLIST are shown in Figure 7.

```
FREE  FI(ISPPLIB ISPMLIB ISPSLIB ISPLLIB ISPTLIB ISPPROF)
ALLOC FI(ISPPROF)  DA('userid.ISPF.PROFILE') SHR REUSE
ALLOC FI(SYSPROC)  DA('SYS1.VSF2CLIB' 'SYS1.ISPCLIB') SHR REUSE
ALLOC FI(ISPMLIB)  DA('SYS1.VSF2MLIB' 'SYS1.ISPMLIB') SHR REUSE
ALLOC FI(ISPPLIB)  DA('SYS1.VSF2PLIB' 'SYS1.ISPPLIB') SHR REUSE
ALLOC FI(ISPTLIB)  DA('SYS1.VSF2TLIB' 'SYS1.ISPTLIB') SHR REUSE
ALLOC FI(ISPLLIB)  DA('SYS1.VSF2LOAD' 'SYS1.ISPLOAD') SHR REUSE
ALLOC FI(FT05F001) DA(*)
ALLOC FI(FT06F001) DA(*)
```

Figure 7. Sample Modifications to the ISPF Invocation CLIST

Modify the ISPF Master Application Menu: Figure 8 shows how you might modify the ISPF master application menu to provide an option for Interactive Debug (here, option 2) and invoke a prompt panel (here, called USERISP).

```
%------------------------  ISPF MASTER APPLICATION MENU  ----------------------
%OPTION  ===>_ZCMD                              %     +USERID   - &ZUSER
%   1 +SAMPLE1      - Sample application 1             +TIME     - &ZTIME
%   2 +VSF IAD      - VS FORTRAN Interactive Debug     +TERMINAL - &ZTERM
%   3 +.            - (Description for option 3)       +PF KEYS  - &ZKEYS
%   4 +.            - (Description for option 4)
%   5 +.            - (Description for option 5)
%   X +EXIT         - Terminate ISPF using list/log defaults
%
+Enter%END+command to terminate ISPF.
%
)INIT
  .HELP    = ISP00005      /* Help for this master menu              */
  &ZPRIM   = YES           /* This is a primary option menu          */
)PROC
  &ZSEL = TRANS( TRUNC (&ZCMD,'.')
                1,'PANEL(ISP@PRIM)' /* Sample primary option menu  */
                2,'PANEL(USERISP)'   /* SAMPLE PANEL TO INVOKE IAD  */
             /*********************************************************/
             /*                                                       */
             /* Add other applications here.                          */
             /*                                                       */
             /*********************************************************/
               ' ',' '
                X,'EXIT'
                *,'?' )
)END
```

Figure 8. Sample Modifications to ISPF Master Application Menu

Create an Application Panel: You must create an application panel at your installation similar to the one shown in Figure 9 on page 20. The sample prompts the user for the name of the text file, and then invokes a CLIST called USERDBG.

```
)ATTR DEFAULT(%+_)
      /*  % TYPE(TEXT) INTENS(HIGH)      defaults displayed for      */
      /*  + TYPE(TEXT) INTENS(LOW)       information only            */
      /*  _ TYPE(INPUT) INTENS(HIGH) CAPS(ON) JUST(LEFT)             */
  ! TYPE(INPUT) INTENS(LOW) PAD(' ')  /* input field padded with ' ' */
)BODY
%--------------------  SAMPLE VSF IAD PANEL ----------------------------------
%COMMAND ===>_ZCMD                                                          %
+   ENTER THE NAME OF YOUR PROGRAM BELOW...
+  %===>!MEM
+   ENTER THE NAME OF YOUR LIBRARY BELOW...
+  %===>!LIB

+   ENTER THE LIST OF YOUR EXECUTION TIME OPTIONS BELOW...
+  %===>!FDEBUG

)PROC
 VER(&MEM,NB,NAME)
 VPUT (MEM,LIB,FDEBUG) PROFILE
 &ZSEL='CMD(%USERDBG)'
)END
```

Figure 9. Sample Application Panel (USERISP) to Prompt for the Text File Name

Create a CLIST to Pass the Execution-Time Options: The sample CLIST in Figure 10 invokes AFFLOAD which in turn loads the VS FORTRAN program with the DEBUG parameter. For this example, we have called the CLIST "USERDBG."

```
PROC 0
CONTROL NOLIST MAIN NOFLUSH NOMSG
/*    MEM    - MEMBER  NAME                                          */
/*    LIB    - LIBRARY NAME                                          */
/*    FDEBUG - EXECUTION TIME OPTIONS                                */
ISPEXEC VGET (MEM,LIB,FDEBUG,ZPREFIX)
SET &FAFFID = &MEM                          /* DEFAULT TO MEMBER NAME  */
SET &ZFAFFIN = &STR('&ZPREFIX..&FAFFID..RESTART')
SET &ZFAFFOU = &STR('&ZPREFIX..&FAFFID..LOG')
SET &ZFAFFON = &STR('&ZPREFIX..&FAFFID..INCLUDE')
SET &ZFAFFPR = &STR('&ZPREFIX..&FAFFID..PRINT')
FREE FI(AFFPRINT AFFIN AFFOUT AFFON AFFLOAD)
ALLOC FI(AFFIN) DA(&ZFAFFIN) SHR
IF &LASTCC ¬= 0 THEN ALLOC FI(AFFIN) DUMMY RECFM(F)
ALLOC FI(AFFON) DA(&ZFAFFON) SHR
IF &LASTCC ¬= 0 THEN ALLOC FI(AFFON) DUMMY RECFM(F)
DELETE &ZFAFFOU
ALLOC FI(AFFOUT) DA(&ZFAFFOU) MOD CATALOG SPACE (1) CYLINDERS
DELETE &ZFAFFPR
ALLOC FI(AFFPRINT) DA(&ZFAFFPR) MOD CATALOG SPACE(1) CYLINDERS
ALLOC FI(AFFLOAD) DA(&LIB) SHR
ISPEXEC SELECT PGM(AFFLINKF) PARM(&MEM/&FDEBUG) NEWAPPL(AFF) NEWPOOL
FREE FI(AFFPRINT AFFIN AFFOUT AFFON AFFLOAD)
```

Figure 10. Sample CLIST (USERDBG) to Invoke the Program without PDF

Using the Execution Panel

After you have filled in the FOREGROUND VS FORTRAN Version 2 Interactive Debug panel and pressed the ENTER key, you will be presented with the Interactive Debug execution panel. This panel is used for almost all communication with Interactive Debug. See Figure 11.

Note: You may initially be presented with the listings data set specification panel if every debuggable program routine does not have a listing defined. See "Modifying the Listings Data Set Specification Panel" on page 27.

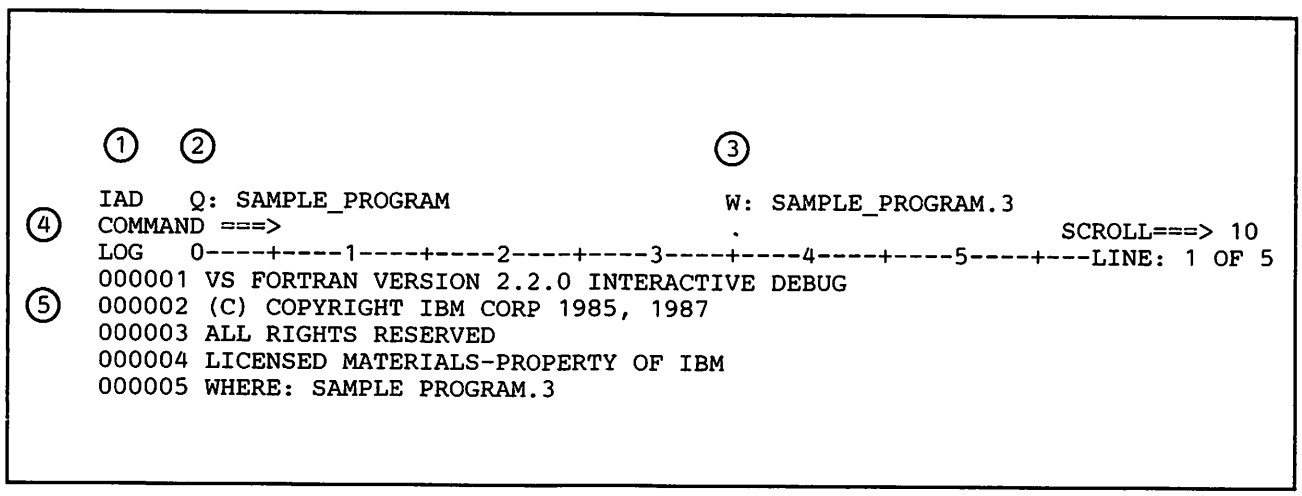

Figure 11. VS FORTRAN Version 2 Interactive Debug Execution Panel

The labels and fields in the execution panel include the following: (The numbers to the left of each item refer to the circled numbers in Figure 11.)

① IAD — Indicates Interactive Debug status. May be coupled with a /R (read), /E (error), /W (write), or /F (finished).

② Q: — The name of the currently qualified program unit. Normally, the name is that of the program unit executing. In this example, the unit is SAMPLE_PROGRAM

③ W: — The statement in the VS FORTRAN program at which execution has been suspended. Also displayed is the name of the program unit where the statement is located. In our sample panel, this is statement 3 in SAMPLE_PROGRAM (SAMPLE_PROGRAM.3)

④ COMMAND ===> — The input area for the next debug command.

⑤ The numbered list is the log of debug commands and responses.

Entering Commands

Commands are entered on the command line near the top of the panel.

To enter an Interactive Debug command, you may either type the complete command, or move the cursor into the scrollable log, modify the command in the log (removing the asterisk is sufficient), and press the Enter key. This will cause the command to be copied to the command line. You may modify the command further before pressing the Enter key.

Certain keywords (such as UP or SPLIT) have special meaning to ISPF and will not be passed to Interactive Debug.

It may be necessary to enter a command that will not fit on the command line. In this case, you may enter some portion of the command, then the continuation character (-) to indicate that the command is not yet complete. Then enter the remainder of the command.

The continuation character (-) must be the last character entered on the command line. You may enter up to 252 characters, including blanks, in one command. If a continuation segment requires leading blanks, type a quotation mark (") first, then the required leading blanks. Interactive Debug will remove the quotation mark and recognize the leading blanks.

While Interactive Debug is awaiting the continuation of a command, COMMAND will be replaced on the panel by MORE.... If, after entering a portion of a command, you decide that you don't want to complete the command, you may enter END (or press a PF key that has been assigned the character string END) and the entire command will be ignored.

The following commands cannot be used with the continuation character:

COLOR
LISTINGS
MOVECURS
POSITION
PREVDISP
PROFILE
SEARCH
WINDOW

Using Program Function Keys to Enter Commands

Under ISPF, you can use program function keys (PF keys) to enter commands on the command line. This saves you time because you can merely press a PF key instead of typing in a long command.

To define PF keys when running Interactive Debug under ISPF, enter the command KEYS on the execution panel command line. This is an ISPF command, not an Interactive Debug command, but it should be entered from the execution panel to define Interactive Debug PF keys.

ISPF immediately presents you with a panel that displays the current PF key definitions and allows you to change them. Move the cursor to the definition you

want to change, and type over it with the new definition. For example, you can assign any Interactive Debug command or a stack of commands to any PF key. There are certain keys you may want to avoid changing, because they have standard ISPF meanings. When you have made all the changes you want, press the END key.

You can use the ISPF command PFSHOW to display your PF key settings across the bottom of the screen.

Entering Input to a VS FORTRAN Version 2 Program

Note: The discussion below assumes that the TERMIO setting is IAD, which causes Interactive Debug routines to be used for terminal I/O. (TERMIO IAD is the default, so you may not need to issue the command.) If TERMIO IAD is not in effect, you do not have the ability to enter Interactive Debug commands while a VS FORTRAN READ is pending, and the discussion is not applicable. The TERMIO command is described in "TERMIO Command" on page 207. For additional information, see "Entering Terminal Input" on page 91.

VS FORTRAN Version 2 Interactive Debug will issue a message before attempting to read input for your program from the terminal. You can enter Interactive Debug commands (except GO, STEP, or ENDDEBUG) prior to responding to the program's request for input.

When execution is suspended for terminal input, the IAD heading will change to IAD /R and the following message will appear:

```
FT05F001 INPUT: PRECEDE INPUT WITH % OR ENTER IAD COMMAND
```

When entering input to a VS FORTRAN program, you must precede it with a percent sign (%); the leading percent sign will not be passed to the program. If you need to precede your input with one or more leading blanks, you may do so following the percent sign and your program will receive the leading blanks. Similarly, you can include trailing blanks in your input by typing a percent sign at the end of the blanks. To simulate entering a null line to indicate end-of-data set, enter:

```
%%
```

Viewing the Scrollable Log

Interactive Debug commands and output sent to the terminal will appear on the execution panel below the command line. This output is called the *scrollable log*. Space for this output is limited, especially if the screen is split into two portions. If you have not changed the default settings, you can use the PF7 and PF8 keys to move the scrollable log up and down. You may also use the standard ISPF commands, such as UP and DOWN, for positioning the scrollable log vertically. To position the scrollable log horizontally, use the standard ISPF commands LEFT and RIGHT.

The last 1000 lines are retained for viewing in the scrollable log.

Error Messages

If Interactive Debug detects an error, a general message, ERROR DETECTED BY IAD, will overlay the upper right corner of the screen, and an alarm will sound. In addition, a specific error message will usually appear in the scrollable log. Interactive Debug messages are described in Appendix C, "Interactive Debug Messages" on page 231

If you need help at the terminal, see Appendix A, "Using the Interactive Debug HELP Facility" on page 219.

Breaks Initiated by Interactive Debug

ISPF does not display results until all output is completed. Using the PROFILE panel, you can specify how often Interactive Debug should suspend its production of output so you can examine the information produced. The "Output halt value" field on the PROFILE panel allows you to specify the number of lines after which Interactive Debug should initiate a break. The halt value is described in "Setting or Changing the Defaults for Your Debugging Sessions" on page 25.

The default value of 50 indicates that Interactive Debug suspends the output after every 50 lines produced. For example, if a program produces more than 50 lines of output to the terminal without an intervening breakpoint, Interactive Debug forces a break in execution and displays the message HALTED FOR OUTPUT. If Interactive Debug, in response to a command, produces more than the designated number of lines of output without breaking, execution is suspended at the next statement boundary and the message "NEXT" FORCED FOR OUTPUT is displayed. Issuing the GO command will resume execution until the next break.

Using Interactive Debug Features under ISPF Version 2

The following features are available with Interactive Debug under ISPF Version 2, and are described in this section:

- Setting or changing the defaults for your debugging sessions
- Displaying your source listing in a window, including an optional overlay of program sampling statistics in the form of a bar chart
- Using cursor-oriented commands
- Searching the source listing or the log for variable strings
- Animating the execution of your program
- Changing the color attributes of your panel

Running Interactive Debug with ISPF Version 2, you can view your source listing in one area of the screen without splitting the screen. This area is called the *source listing window.*

You can also highlight the command currently executing, and control the pace of execution when using the STEP command. This capability is called *program animation*, because it creates an animated picture of your program's execution. The screen is automatically rewritten at every statement hook. In addition, you can change the color attributes of the execution panel.

The following commands are valid if you are using Interactive Debug with ISPF Version 2:

COLOR	PREVDISP
LISTINGS	PROFILE
MOVECURS	SEARCH
POSITION	WINDOW

These commands are referred to in this manual as the *full screen display commands*.

Setting or Changing the Defaults for Your Debugging Sessions

When you begin a debugging session under ISPF Version 2, you may want to display the default settings for various parameters that affect the way IAD runs. To see these parameters, enter the command `PROFILE`. A display panel will then appear on your screen. You can modify this display any time during a debugging session. Initially, the current setting for any of the parameters displayed will be the same as your profile setting. However, if you want to modify a parameter for the current session, you can type over that field on the display panel with a new value. If you modify the profile settings, the new values are saved and used whenever you begin a new debugging session.

After you have modified some of your current settings, the profile panel might look like this:

```
            VS FORTRAN INTERACTIVE DEBUG      (CURRENT & PROFILE Settings)

COMMAND ===>

                          CURRENT SETTING          PROFILE SETTING
                          ---------------          ---------------
Step delay (.01 sec)            10                       50
Frequency count display         NO                       YES
Window columns                  60                       40
Window rows                     10                       10
Window ON                       NO                       YES
Log line numbers                YES                      YES
Output halt value               50                       50

Enter END or RETURN to go back to IAD panel.
```

Figure 12. Modifying the PROFILE Command Panel

Initially, the profile settings have the values displayed in Figure 12. However, you can modify these values (in addition to modifying the current settings).

The parameters are:

Step delay Initially set to 50. This value controls the pace of animation, measured in hundredths of a second. For more information, see Figure 15 on page 32.

Frequency count display
Initially set to YES. YES indicates that the statement frequency counts or sampling percentages are shown within the source listing window.

Window columns
Initially set to 40. This value specifies the width of the source window listing, measured in characters. To make the window wider, increase this value.

Window rows
Initially set to 10. This value specifies how deep the source listing window will be, measured in lines. To see more than 10 lines at a time in the window, increase this value.

Window ON
Initially set to YES. This value indicates that a source listing window automatically appears on your screen, provided that its dimensions have been defined and the listing data set is known for the current program unit. Setting this value to YES is equivalent to entering the `WINDOW` or `WINDOW ON` command.

Log line numbers
Initially set to YES. This value indicates that the log line numbers are displayed in your scrollable log. Enter NO to inhibit the display of these numbers.

Output halt value
Initially set to 50. This value indicates how often Interactive Debug should suspend its production of output so you can examine the information produced. The value is specified as the number of lines after which Interactive Debug should initiate a break.

Displaying Your Source Listing in a Window

You can use a window to display the source listing for a selected program unit. The source listing window overlays the log display, without splitting the screen. One way to define a window is by changing the "window rows" and "window columns" values on the profile panel, as described in the previous section. Another way is to use the WINDOW command.

Defining the Window with the WINDOW Command

To define the dimensions of your source listing window, first type `WINDOW` on the command line, then position your cursor at the place where you would like to have the lower left corner of the source display. By moving the cursor downward or to the left, you can expand the dimensions of the window. (The upper right corner of the window is fixed at the upper right of the screen.) When you have the dimensions you want, press `ENTER`.

Note: If you have not yet modified the listings data set specification panel (described below), you may not immediately see the source listing window, but the dimensions will be saved and the window is defined.

If you have a PF key assigned to the WINDOW command, move your cursor to the position where you want the lower left corner and press the PF key.

You can change the dimensions of the window at any time during a debugging session to take advantage of your terminal or your special debugging needs. You can save the window definition on the PROFILE panel, so it will be used the next time you debug a program with Interactive Debug.

To temporarily turn the window off, use the command `WINDOW OFF`. The window returns when you enter `WINDOW` or `WINDOW ON` (or when you press the equivalent PF key), provided that the listings data set is known to Interactive Debug for the current program unit.

Displaying Program Sampling Bar Charts

As an option, you can have the source listing window overlaid with bar charts that display program sampling statistics for each statement in the listing. For a description of this feature and of program sampling, see "Program Sampling" on page 79.

Modifying the Listings Data Set Specification Panel

If there is at least one debuggable program routine that has no listing defined, the listings data set specification panel is automatically displayed at IAD initialization. This can occur if these routines are not all specified in AFFON, described on page 37, or subsequently not found in an attempt made by Interactive Debug to search a file or data set whose name is generated by IAD from the given application program name. You can enter missing data set definitions on the listings data set specification panel. Press the END key when you are done.

Note: If a restart file is present, this automatic display of the listings data set specification panel is deferred until the end of the restart file is reached. If a QUIT command is executed from the restart file, the automatic intervention is bypassed.

You can also request the listings data set specification panel from the debugging session by entering the LISTINGS command or by typing over the first character of the Q: field in the header with a question mark (?). A sample CMS panel is shown in Figure 13 on page 28

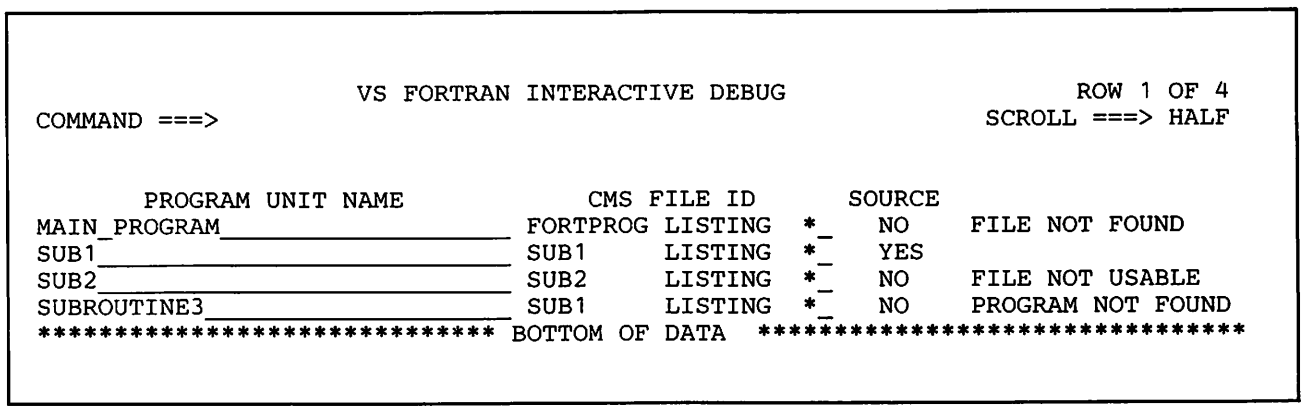

Figure 13. The Listings Data Set Specification Panel in CMS

The TSO panel is similar. The CMS FILE ID column is replaced by the DATA SET NAME column. Under TSO, the top line of the panel looks like this:

```
PROGRAM UNIT          DATA SET NAME                SOURCE
```

If the source listing you want does not appear on the panel, you must specify the file ID or data set name for the program unit source listing. To fill in the listings data set specification panel, enter the name of the program unit source listing in the "CMS FILE ID" or "DATA SET NAME" column. Interactive Debug then automatically fills in the names of all program units found in that listing, and changes the "SOURCE" column to YES. If the listing is in a PDS member, include the member name in parentheses at the end of the data set name.

To display a listing in the source listing window, the "SOURCE" column for that listing must specify YES. You may want to change some of these values to NO if you do not want particular program units to be displayed.

In Figure 13, the CMS message FILE NOT FOUND indicates that the specified file was not found on any of your accessible disks. Under TSO, the equivalent message is DATA SET NOT FOUND. To correct the problem, make sure the file ID for the listing containing the program unit is correctly spelled and is accessible. If it is necessary to use system commands (CMS or TSO) to make the listing accessible, you can enter them on the command line prefixed by "CMS" or "TSO," as appropriate.

The message PROGRAM NOT FOUND indicates that the listing file was found, but did not contain the named program (in this example, SUBROUTINE3). To correct the problem, make sure the file ID for the listing containing the program unit is correctly spelled and is accessible.

If you receive the message FILE NOT USABLE, it indicates that the data set was found but cannot be used as a listing in the source window. Under TSO, the message will be DATA SET NOT USABLE. In either case, check for an invalid record length (it should not be greater than 137 bytes). If operating under CMS, insure that the file is not in PACKED format. If under TSO, insure that the file is not security protected by another user.

After filling in the listings data set specification panel, you return to the execution panel by entering END, usually PF key 3.

A Sample Source Listing Window

A debugging session using a source window might look like that in Figure 14.

```
IAD/F Q: SUB1                                    W: ?
COMMAND ===>                                                  SCROLL ===> HALF

 *TOP*                    TOP OF SUB1                                        ....
       C ************************************************************** ....
       C *     SUBROUTINE SUB1                                            ....
       C ************************************************************** ....
     1         SUBROUTINE SUB1                                            ....
     2         INTEGER I                                                  ....
     3         DO 10 I = 1, 2                                          * 0017
     4           IF (I.EQ.1) THEN                                    *** 0039
     5             CALL SUB3                                 ********** 0296
     6           ENDIF                                                    0006
     7           IF (I.EQ.2) THEN                                  **** 0056
     8             CALL SUB2                                ********** 0218
     9           ENDIF                                                 * 0017
    10 10      CONTINUE                                               ***** 0073
    11         END                                                      ** 0034
 *END*                  BOTTOM OF SUB1                                     ....
------------------------------------------------------------------------------
LOG   0----+----1----+----2----+----3----+----4----+----5----+---LINE:  1 OF 12
000001 VS FORTRAN VERSION 2.2.0 INTERACTIVE DEBUG
000002 (C) COPYRIGHT IBM CORP 1985, 1987
000003 ALL RIGHTS RESERVED
000004 LICENSED MATERIALS-PROPERTY OF IBM
000005 WHERE: TDYNCM.3
000006 * list i
000007 TDYNCM.I          =           0
000008 * q sub3
000009 * enddebug sample(5) called
000010 PROGRAM HAS TERMINATED; RC ( 0)
000011 * q sub1
000012 * annotate on all
```

Figure 14. Example of Debugging Using a Source Window

Note: The border below the source listing window is not displayed on 7-color terminals. However, you can choose a contrasting color for the window to make it distinct and more visible.

When program sampling bar charts are shown, the column at the far right of the source window shows sampling percentages relative to the displayed program unit. One decimal place is implied, for example, 0218 means 21.8%.

The source listing window displays the source listing for the current program unit (if possible). Whenever execution is suspended, you can change the Q: field on the panel heading to tell Interactive Debug to temporarily display the source listing for another program unit. When you resume execution, the display automatically returns to the source listing for the current program unit (if possible). To change the Q: field in the panel heading, merely type over the qualification field with a new valid unit name.

If the following conditions are met, Interactive Debug automatically displays the source listing for the currently executing program unit:

- The data set must be accessible to Interactive Debug, and defined on the Listings Data Set Specification Panel.
- The program unit must be debuggable, and must have been compiled with VS FORTRAN Version 2, or with VS FORTRAN Version 1 Release 3.1 or later. If compiled with VS FORTRAN Version 1, the options TEST and NOSDUMP must not have been specified together.
- The "Source" column of the Listings Data Set Specification panel must specify YES for this program unit.

Scrolling the Listing

You can scroll the source listing vertically or horizontally, just as you scroll the session log. Use the same ISPF commands (UP and DOWN, LEFT and RIGHT) or function keys. The position of the cursor determines which information is scrolled. If the cursor is within the source window, the source listing is scrolled. If the cursor is anywhere else on the execution panel, the log is scrolled.

Moving the Cursor

To quickly move the cursor back and forth between the source window and the command line, use the MOVECURS command. If the cursor is not on the command line when you enter the MOVECURS command, it is placed at the command line. If the cursor is on the command line when you enter the MOVECURS command, the cursor is moved to its previous position in the window, or to the upper left corner of the window if the previous position is not known.

You can assign the MOVECURS function to a PF key by issuing the ISPF KEYS command. You might want to assign it to the PF key normally used for the PDF CURSOR function.

Using Cursor-Oriented Commands

When you use a source window, you can perform many tasks by "pointing" with the cursor, instead of typing operands on the command line. The commands that can be issued this way are called *cursor-oriented* commands. The AT, LIST, and OFF commands are cursor-oriented commands.

AT You can issue an AT command (with no command list) using your cursor. If the AT command is already assigned to a PF key, place the cursor at an ISN or sequence number field in the source window, and press the PF key for AT.

Instead of assigning a PF key to the AT command, you can type the AT command over the ISN or sequence number, or you can type AT on the command line and move the cursor to the target statement number before pressing ENTER.

LIST You can issue a simple LIST command using your cursor. If the LIST command is already assigned to a PF key, place the cursor at a variable name in the source window, and press the PF key for LIST. The variable may include either subscript or substring notation. If both are present, only the subscript will be included in the command.

Instead of assigning a PF key to the LIST command, you can type LIST on the command line and move the cursor to a variable name before pressing ENTER.

OFF You can issue an OFF command using your cursor. If the OFF command (with no parameters) is already assigned to a PF key, place the cursor at an ISN or sequence number field in the source window, and press the PF key for OFF.

Instead of assigning a PF key to the OFF command, you can type the OFF command over the ISN or sequence number, or you can type OFF on the command line and move the cursor to the target statement number before pressing ENTER.

Cursor-oriented commands are recorded in your session log as if the equivalent command had been typed on the command line. For example, if you type AT over ISN 12 in the source window while program unit SUB1 is displayed, the command `AT 12` is recorded in the session log.

If the command line contains a command when you press a PF key for a cursor-oriented command, the command on the command line is not executed. If you type over multiple ISN or sequence number fields with AT or OFF commands, only the first command is executed.

Searching the Source Listing or the Log for Character Strings

You can use the SEARCH command to search the source listing or the log for a string up to 64 characters long, depending on your terminal type. For the syntax of the search command, see "SEARCH Command" on page 197.

If you issue the SEARCH command while the cursor is in the source listing window, the source listing is searched; otherwise, the log information is searched. The search argument is translated to uppercase, and the entire listing is treated as uppercase. (That is, the target will be found regardless of case.) The most recent search argument issued on a SEARCH command is always saved, so if you later issue the SEARCH command without an argument, the saved argument is assumed.

You can position at a log line number, or an ISN or sequence number, by using the POSITION command. (For the syntax of the POSITION command, see "POSITION Command" on page 183.) If you issue the POSITION command while the cursor is in the source listing window, an ISN or sequence number is searched for; otherwise, a log line is searched for.

Animating the Execution of Your Program

When you use the STEP command with the source listing window, Interactive Debug "animates" the execution of your program so you can watch the execution progress. The currently executing line is highlighted in the source listing.

The source, as well as the log and a display generated by the AUTOLIST command (if such a display exists), is redisplayed after each step of the program has been executed. When the STEP command terminates, or when execution is halted by some other means (such as breakpoints), animation ends. For more detail, see "STEP Command" on page 203.

Controlling the Pace of Execution

You can control the timing of animation by modifying the STEP DELAY field in the profile panel. To change the value, enter the command PROFILE.

When the profile panel appears, type over the existing value in the "Step delay" field. The delay time is specified in hundredths of a second. The default is set to 50 hundredths of a second. Figure 15 shows that the current setting for the delay time has been changed to 10 hundredths of a second, although the setting for the profile is still 50 (the default).

```
              VS FORTRAN INTERACTIVE DEBUG          (CURRENT & PROFILE Settings)

COMMAND ===>
                           CURRENT SETTING           PROFILE SETTING
                           ---------------           --------------
Step delay (.01 sec)             10                        50
Frequency count display          YES                       NO
Window columns                   60                        40
Window rows                      10                        10
Window ON                        NO                        YES
Log line numbers                 YES                       YES
Output halt value                50                        50

Enter END or RETURN to go back to the IAD panel.
```

Figure 15. Modifying the STEP DELAY Field

When the source display is inhibited, the program steps with the minimum processing time between steps (in other words, no delay).

Another way of "animating" your program is to issue repeated GO commands in conjunction with a HALT statement. To do this, assign the GO command to a PF key. Next issue a HALT statement to specify the conditions under which the program will be halted and the source listing redisplayed. If you now repeatedly press the PF key for the GO command, you can watch the flow of control in the program by observing the highlighted line in the source window.

Changing the Color Attributes of Your Panel

You will probably find debugging more convenient if you highlight certain parts of the Interactive Debug Execution Panel, such as the current statement in the source window, and the statement identifiers at which breakpoints have been set. You can change the color, highlighting, or intensity of various fields on the panel, using the COLOR command. For more information, see "COLOR Command" on page 134.

Splitting the Screen Using ISPF and PDF

With ISPF and PDF, you can look at a compiler listing file (or the source file) by splitting the physical screen into two logical screens. With a split screen, a source or listing file may be browsed or edited (or even recompiled under TSO) while debugging. This assumes the presence of IBM Program Development Facility (PDF), which provides the browse and edit functions.

To split the screen, use the SPLIT command (or a PF key assigned the SPLIT function, normally PF2), and then use BROWSE or EDIT to look at the appropriate file or data set.

Note: The second screen in split-screen mode cannot be used to run a second session of Interactive Debug (or any other debugging product). You cannot run any program in the second screen that would intercept attentions, unless you let that program terminate before trying to continue with Interactive Debug.

Recompiling a Program while Using a Split Screen (TSO Only)

You can use the split screen to perform a number of tasks. For example, under TSO, you might want to split the screen, recompile a program, and then restart the debugging session using the new compilation. You would need to complete the following steps:

1. Split the screen into two portions.
2. Go into edit mode on the lower half of the screen (usually panel 2).
3. Make changes to the source program in the lower half of your screen.
4. Request the VS FORTRAN compilation panel (usually 4.3), and specify the member name to be compiled.

5. When the program has compiled, request the link-edit panel (usually 4.7), and specify the member name to be link-edited.

6. When the program has been link-edited, end the split-screen mode and issue the RESTART command on the command line of the execution panel.

You can now debug the newly-compiled program. If you want, you can display the new source listing by splitting the screen, or by issuing the WINDOW command under ISPF Version 2.

After Ending the Debugging Session

When you enter the QUIT command to end Interactive Debug activity, ISPF automatically enters BROWSE so you can examine the complete output log (AFFOUT). If you have used the PRINT keyword on any Interactive Debug commands that allow it, ISPF first enters BROWSE for the AFFPRINT file.

After browsing these files, enter the END command, or use the PF key assigned to END (usually PF 3). You will then be presented with the standard ISPF FOREGROUND PRINT OPTIONS panel, allowing you to print each file and then to keep or delete the file. A sample panel is shown in Figure 16.

```
------------------------------ FOREGROUND PRINT OPTIONS -------------------------
OPTION ===>

   PK - Print file and keep              K - Keep file (without printing)
   PD - Print file and delete            D - Delete (erase) file (without printing)

   If END command is entered, file is kept without printing.

FILE ID: MAIN LOG A

SPOOL OPTIONS:
   NUMBER OF COPIES ===> 1               SPOOL CLASS ===> A
   BIN NUMBER       ===>                 'FOR' USER  ===>
   3800 KEYWORDS    ===>

FOR SPOOLING TO ANOTHER USER OR MACHINE:
   USER / MACHINE ID ===>
   NODE / LINK ID    ===>
   TAG TEXT          ===>
```

Figure 16. The Foreground Print Options Panel under ISPF in CMS

You can fill in the file ID field and any spooling options, and then enter one of the four options listed at the top of the panel. To leave this panel without printing your log file, enter the END command or use the PF key assigned to END.

Bypassing the BROWSE Step

It is possible to bypass the BROWSE step by modifying the AFFFX11 EXEC under CMS, or the AFFFC11 CLIST in TSO.

Under CMS, follow these steps:

1. Edit the AFFFX11 EXEC.
2. Add the lines `zfbrows = ''` and `zfprint = ''` to the EXEC as follows:

```
zfbrows = ''
zfprint = ''
/* Browse print file if it exists, set message if not */
afftype = 'PRINT' /* afftype is used in message AFFS006 */
```

3. Alternatively, you can use `'/*'` and `'*/'` to comment out the lines

```
If zfbrows ¬= '' then 'ISPEXEC BROWSE FILE('lid')'
If zfprint ¬= '' then
  'ISPEXEC SELECT CMD(ISRFXPRT ISRFPPRT' lid')'
```

These lines occurs twice--once for displaying the print file and once for displaying the log file. For example, after you comment out the lines for displaying the log file, the EXEC should look like this:

```
/* Browse log file if it exists, set message if not */
formsg = ''
afftype = 'LOG'
lid = zfname 'LOG A'
'STATE' lid
If rc <= 0  then Do
  Address 'ISPEXEC'
/*  If zfbrows ¬= '' then 'ISPEXEC BROWSE FILE('lid')'
    If zfprint ¬= '' then
      'ISPEXEC SELECT CMD(ISRFXPRT ISRFPPRT' lid')'*/
    Address CMS
End
Else formsg = 'AFFS006'
```

If you want to bypass browsing the print menu, search for the first occurrence of these lines and comment them out also.

Under TSO, follow these steps:

1. Edit the AFFFC11 CLIST.
2. Search for the following line:

```
SET ZFBROWS = BROWSE        /* ASK FOR BROWSE        */
```

3. Change the line to read as follows:

```
SET ZFBROWS = &Z           /* BYPASS BROWSE          */
```

You can also bypass the print menu by setting ZFPRINT to a null string, as follows:

```
SET ZFPRINT = &Z            /* BYPASS PRINT          */
```

Using Optional Debugging Files or Data Sets

The following sections discuss the output log file, and three optional files (or data sets). It describes how to define each file when invoking Interactive Debug with ISPF.

Specifying an Output Log File or Data Set (AFFOUT)

When running under ISPF, Interactive Debug creates a log of its activity for you to examine; this log can be viewed after completion of the debugging session. The log information is contained in the AFFOUT data set or the AFFOUT file.

The AFFOUT file does not need to exist prior to execution of the VS FORTRAN program. Under CMS, the invocation procedures define a file named *fname* LOG (where *fname* is the name specified on the FILE ID line of the invocation panel). Under TSO, the invocation procedures define a data set named *userid.fname*.LOG (where *fname* is the name specified on the MEMBER line or on the FILE ID FOR DEBUG FILES line on the invocation panel).

Note: To prevent an existing file from being overwritten, you must rename it before executing a program.

The log file has the following characteristics:

- All Interactive Debug I/O is logged (except for a small number of commands associated with functions available under ISPF Version 2: COLOR, LISTINGS, MOVECURS, POSITION, PREVDISP, PROFILE, SEARCH, WINDOW). Unless TERMIO LIBRARY has been specified, all program-initiated terminal I/O is also logged.
- The file is created with a RECFM of FB and an LRECL of 80 (and a BLKSIZE of 800 for TSO).
- Each line is preceded with an equal sign (=).
- Each line of input is preceded by an asterisk (*) following the equal sign.
- Input and output occurring within an attention exit are not logged; however, the entering of an attention exit is logged.

Specifying a Print File or Data Set (AFFPRINT)

Certain VS FORTRAN Version 2 Interactive Debug commands allow you to specify a PRINT keyword, causing command output to be sent to a print data set rather than to your terminal. This may prove useful, for example, when you are listing the contents of a large array and want to keep this output separate from the normal log. This print data set is referred to as the AFFPRINT data set or the AFFPRINT file.

AFFPRINT does not need to exist prior to executing a VS FORTRAN program. Under CMS, the invocation procedures will define a file named *fname* LIST A (where *fname* is the name specified on the FILE ID line of the invocation panel). Under TSO, the invocation procedures will define a data set named

userid.fname.PRINT (where *fname* is the name specified on the MEMBER line or on the FILE ID FOR DEBUG FILES line on the invocation panel).

Note: To prevent an existing AFFPRINT file from being overwritten, you must rename it before executing a program with the same name.

Specifying Program Units to be Debugged (AFFON)

When Interactive Debug is invoked, you can set breakpoints in any debuggable program units. Because there is some overhead associated with this ability, it is more efficient to exclude from debugging those program units that are known to be error free. You can also exclude part of a program unit, which may be helpful if there are heavily-executed sections that are error free.

The AFFON file or AFFON data set is a sequential file containing a list of program unit names, up to 31 characters in length, to be debugged. The AFFON file is read by Interactive Debug; it must exist before execution of the VS FORTRAN program. (Using the RESTART command, you can change the allocation or the content of the existing file prior to restarting.) Using the information in the AFFON file entries, Interactive Debug will attempt to automatically identify the data sets containing the program listings for use in the source window and with the ANNOTATE command.

The invocation procedures must include a correct FILEDEF statement (CMS) or ALLOCATE statement (TSO) to define the AFFON file or data set. The invocation procedure supplied by IBM for use with ISPF/PDF under CMS assumes that the AFFON file is called *fname* INCLUDE (where *fname* is the name specified on the FILE ID line of the invocation panel). Under TSO, it is called *userid.fname*.INCLUDE (where *fname* is the name specified on the MEMBER line or on the FILE ID FOR DEBUG FILES line on the invocation panel).

The file contains a list of program unit names to be selected for debugging, and may contain comment entries identified by an asterisk (*) in the first record position. Program unit names are assumed to be the first character string in the record (terminated by a blank), and may appear after initial leading blanks.

Each program unit name can be followed by the data set name containing the source listing for a program unit as well as a list of statement numbers (ISNs or sequence numbers), or statement number ranges. (ISNs are the default, unless you specified SDUMP(SEQ) when you compiled the program unit.) If any statement numbers are specified, only statements falling within the specified ranges, or included in the list of statement numbers, are selected for debugging. If no statement numbers are specified, all executable statements are selected. Note that statement labels are **not** allowed.

The syntax of each entry in the AFFON file is as follows:

```
unitname ['dataset[(member)]'][n[:n]]...
```

or

```
unitname ['dataset[(member)]'][ENTRY]
```

or

```
unitname ['dataset[(member)]'][NONE]
```

Blanks or commas separate entries in the list of statement number ranges. The program unit name and the list of statement number ranges must all fit within one record. The ENTRY parameter specifies that only ENTRY and EXIT hooks are to be placed in the program unit. This is helpful if you want to increase the accuracy of timing information. The NONE parameter allows data set names to be specified, but by passes inclusion of debugging hooks.

For example, to place debugging hooks between statements 6 and 16 and at statement 18 of SUB1, you could make the following entry, under CMS, in your AFFON file:

```
Sub1     'SUB1 Listing *'              6:16  18
```

In the above example, SUB1 Listing * is defined as the default listing. ENTRY and EXIT hooks are always placed in the program unit whenever any statement number ranges are specified.

An equivalent example for TSO would be:

```
Sub1     'userid.sub1.list'            6:16  18
```

Note that there may be executable statements in the specified ranges that cannot have debugging hooks placed on them because of optimization or vectorization. To find out which statements have had hooks placed on them, you can use the LISTFREQ command.

Program units that do not meet the requirements for debugging will not have program hooks inserted, even though they may be in the AFFON list.

One way to create an AFFON file is to edit the map produced when the module to be executed is built. Normally, this is the linkage editor map or the CMS load map. The map can be edited by placing an asterisk before any program unit name that is not to be debugged, nominating the other program units for debugging. You can then add in a list of statement numbers or statement number range specifications.

Note: Because the CMS load map truncates long names to 7 characters, you will have to replace truncated names with complete names as required by Interactive Debug.

AFFON can have any record format; entries can be in lower case. The logical record length can be any value up to 255 bytes. If the file exists but has incorrect attributes, you will receive an error message stating that the AFFON file cannot be read. AFFON can contain references to sequence numbers for VS FORTRAN programs compiled with SDUMP(SEQ), but AFFON itself cannot be sequenced.

If the AFFON data set is found and contains at least one entry that is not a comment, only the valid program unit names found in the data set or the program unit compiled with the test option will be considered for debugging. Otherwise, all VS FORTRAN program units will be considered for debugging.

For compiler and library restrictions that may make a program unit nondebuggable, see "Interactive Debug Programming Requirements" on page 5.

Specifying a Restart File or Data Set (AFFIN)

When running under ISPF, Interactive Debug commands can be read from the AFFIN input file. AFFIN is a file of debugging commands, initially created either with an editor or obtained by editing the output from a previous debugging session. This file is optional under ISPF.

The AFFIN file must exist prior to executing the VS FORTRAN program. (If you use the RESTART command, you can change the allocation or the content of the existing file prior to restarting.)

Under CMS, the ISPF invocation procedures supplied by IBM assume that the AFFIN file is called *fname* RESTART (where *fname* is the name specified on the FILE ID line or MEMBER line of the invocation panel). Under TSO, it is assumed to be *userid.fname*.RESTART (where *fname* is the name specified on the MEMBER line or on the FILE ID FOR DEBUG FILES line on the invocation panel).

The AFFIN file must have a RECFM of F or FB and an LRECL of 80. The file will be read until end-of-file is encountered. After the commands in the file have been executed, additional input can be entered from the terminal.

Note: Sequence numbers are **not** allowed in columns 73 through 80.

You may want to use the log file (AFFOUT) as input to a subsequent debugging session if, for example, you had to discontinue a debugging session but had not yet solved the problem. To do this, follow these steps:

1. Use the QUIT command to stop debugging.
2. Keep the AFFOUT log file, and edit it to remove the QUIT command.
3. Prior to using the AFFOUT file as input to Interactive Debug, you must rename it, as described above.
4. The log file can now be used as input to retrace the steps taken in the previous session.

After the log file has been executed, you should be at the same position as when you stopped the previous session.

Interactive Debug recognizes a saved log file by the presence of leading equal signs (=). Lines in the input file beginning with an equal sign and an asterisk (=*), and lines not beginning with an equal sign, are assumed to be debugging commands. Lines beginning with an equal sign **not** followed by an asterisk are assumed to be output and are ignored.

Chapter 3. Using Interactive Debug in Line Mode

Interactive Debug may be executed in line mode in either a CMS or a TSO environment. It is referred to as line mode because input and output are presented sequentially, one line at a time. This is in contrast to executing in full screen mode, when the entire screen is controlled and used by Interactive Debug. Use of Interactive Debug in line mode is intended primarily for users with access to only a typewriter-like terminal or when ISPF is not installed.

Invoking Interactive Debug

The DEBUG execution-time option causes the VS FORTRAN library initialization routines to load Interactive Debug. The program will be suspended at the first statement with a debugging hook, so you can issue Interactive Debug commands before execution proceeds.

The invocation procedure varies depending on whether you are using CMS or TSO. Both procedures are described below.

Under CMS

When you are executing under CMS without ISPF, you can build your executable program and invoke Interactive Debug in one of three ways:

1. Use the LOAD command to load your VS FORTRAN program. You execute the program using the START command with the DEBUG option. No permanent copy of the executable program is made.

2. Use the LOAD and GENMOD commands to generate a module. You can invoke the program later by issuing a CMS command with the same name as the MODULE file, and the DEBUG operand.

3. Use the LKED command to link-edit your VS FORTRAN program into a load library. You execute the program later, using the OSRUN command with the PARM=DEBUG parameter.

The steps to invoke Interactive Debug under each of these methods are described in the following sections. The steps include instructions to invoke Interactive Debug under both link mode and load mode with VS FORTRAN Version 1 or VS FORTRAN Version 2. The steps also include instructions for executing programs created prior to Version 1 Release 4 of VS FORTRAN.

Link Mode and Load Mode

If you are running under the VS FORTRAN Version 1 Release 4, 4.1, or VS FORTRAN Version 2 Library, you should be aware of the distinction between two modes of operation.

If you choose to have the necessary library routines included within your executable program, you are operating in *link mode*. If, on the other hand, you choose to have the library routines loaded during execution of your program, you are operating in *load mode*. You make the choice of link mode or load mode by making the appropriate combination of libraries available when you create your executable program from your TEXT files.

For more information on link versus load mode, see *VS FORTRAN Version 2: Programming Guide.*

Using the LOAD and START Commands

The LOAD command creates a temporary copy of your executable program in virtual storage. The object code from which the executable program is built is either in TEXT files or in text libraries, or both.

1. You must make the appropriate VS FORTRAN Version 2 Library text libraries as well as your own text libraries, if any, available using a GLOBAL command.

   a. If you want your program to execute in link mode, use this command:

      ```
      GLOBAL TXTLIB VSF2LINK VSF2FORT CMSLIB TSOLIB userlib...
      ```

   b. If you want your program to execute in load mode, use this command:

      ```
      GLOBAL TXTLIB VSF2FORT CMSLIB TSOLIB userlib...
      ```

2. To create the temporary copy of your executable program in virtual storage, issue the LOAD command, as follows:

   ```
   LOAD myprog...
   ```

3. Next issue the following command:

   ```
   GLOBAL LOADLIB VSF2LOAD
   ```

4. Finally, to execute the temporary copy of your program that has been built in virtual storage and invoke Interactive Debug, issue the following command:

   ```
   START * DEBUG
   ```

Using the LOAD and GENMOD Commands

The LOAD and GENMOD commands create an executable program that is stored as a nonrelocatable file on your CMS disk. The object code from which the executable program is built is either in a TEXT file or in a member of a text library.

Note: If you have created a nonrelocatable file with a file type of MODULE for your program in the past, you can execute the program by completing steps 4 through 6 under the "Executing the Program" section that follows.

Creating the MODULE File

1. You must make the appropriate VS FORTRAN Version 2 Library text libraries as well as your own text libraries, if any, available using a GLOBAL command.

   a. If you want your program to execute in link mode, use this command:

      ```
      GLOBAL TXTLIB VSF2LINK VSF2FORT userlib...
      ```

   b. If you want your program to execute in load mode, use this command:

      ```
      GLOBAL TXTLIB VSF2FORT userlib...
      ```

2. Next, create the temporary copy of your executable program in virtual storage by issuing the LOAD command, as follows:

   ```
   LOAD myprog...
   ```

3. To create the nonrelocatable file on your CMS disk, issue the following GENMOD command:

   ```
   GENMOD modname
   ```

   This command builds a file with the file name assigned as `modname`, and a file type of MODULE. This program can be executed at any time.

Executing the Program

4. You may need to issue one or more GLOBAL commands before executing your program. Issue the following command if the simulation of extended precision floating-point instructions is required on the machine you are using:

   ```
   GLOBAL TXTLIB CMSLIB TSOLIB
   ```

5. Next issue the following command:

   ```
   GLOBAL LOADLIB VSF2LOAD
   ```

6. Finally, to execute your program that is stored as a nonrelocatable file and to invoke Interactive Debug, issue the following command:

   ```
   modname DEBUG
   ```

   where `modname` is the file name of your MODULE file, as originally specified in the GENMOD command.

Using the LKED Command

Use the LKED command to link-edit your program and store it as a relocatable load module in a member of a CMS LOADLIB.

Note: If you have created a load module from your program in the past, you can execute the program by completing steps 3 through 5 under the "Executing the Program" section that follows.

Creating a Load Module

1. Prior to issuing the LKED command, issue the following FILEDEF command:

   ```
   FILEDEF SYSLIB DISK VSF2FORT TXTLIB fm
   ```

   where `fm` is either the file mode of the CMS disk that contains the library VSF2FORT, or an asterisk (*).

2. Then issue the following LKED command:

   ```
   LKED myprog (LIBE libname NAME membname
   ```

   where:

   myprog is the file name of the TEXT file that contains your object code.

   libname is the file name of the LOADLIB file into which the resulting load module is to be placed as a member.

   membname is the name of the member in the LOADLIB file designated by `libname` above, into which the resulting load module is to be placed.

Executing the Program

3. Next issue the following GLOBAL command:

   ```
   GLOBAL LOADLIB VSF2LOAD libname
   ```

   where `libname` is the file name of the LOADLIB file into which your load module was placed as a member by the LKED command.

4. Issue the following command if the simulation of extended precision floating-point instructions is required on the machine you are using:

   ```
   GLOBAL TXTLIB CMSLIB TSOLIB
   ```

5. Issue FILEDEF statements for the AFFON and AFFPRINT files. For example,

   ```
   FILEDEF AFFON DISK progname AFFON A
   FILEDEF AFFPRINT DISK progname AFFPRINT A
   ```

6. Finally, issue the following command to execute your program and invoke Interactive Debug:

   ```
   OSRUN membname PARM=DEBUG
   ```

   where `membname` is the name of the member that contains the load module created with the LKED command.

A Sample Invocation EXEC: To invoke VS FORTRAN programs, you may find it convenient to create a CMS EXEC. If you already have one, you will probably want to modify it to invoke Interactive Debug.

Figure 17 is an example of a simple EXEC that invokes the VS FORTRAN program with the DEBUG option. The EXEC allocates an AFFPRINT file, and does not allocate an AFFON file. (The AFFPRINT and AFFON files are described later in this chapter.) If DEBUG is not specified (or defaulted by the EXEC), Interactive Debug will not be invoked unless you have included a special object module to override the default. For further information about how to override the default, see "Specifying Default Execution-Time Options" on page 96.

This sample EXEC assumes you are running with a previously-linked VS FORTRAN module, or are starting with TEXT files you want to link with the VS FORTRAN Version 2 Library to run in **load mode**. On the other hand, if you are starting with TEXT files and want to link with the VS FORTRAN Version 2 Library to run in **link mode**, you need to alter the sample EXEC: Add VSF2LINK as the first GLOBAL TXTLIB (see the examples above under "Link Mode and Load Mode" on page 42).

```
&TRACE
GLOBAL TXTLIB VSF2FORT CMSLIB TSOLIB
* THE ABOVE STATEMENT ASSUMES YOU WILL RUN IN LOAD
* MODE.  IF YOU WISH TO RUN IN LINK MODE, REPLACE THE
* ABOVE STATEMENT WITH THE FOLLOWING GLOBAL STATEMENT:
* GLOBAL TXTLIB VSF2LINK VSF2FORT CMSLIB TSOLIB
*
GLOBAL LOADLIB VSF2LOAD
FILEDEF AFFON    DUMMY
FILEDEF AFFPRINT DISK &1 AFFPRINT A
&PARM = &2
&IF .&PARM = . &PARM = DEBUG
LOAD &1 (CLEAR
START * &PARM &3 &4 &5 &6 &7 &8
&EXIT &RETCODE
```

Figure 17. Sample CMS EXEC to Invoke a VS FORTRAN Program

If the EXEC in this example is invoked without specifying a second parameter, DEBUG will be the default option. If the EXEC were named FORTIAD, specifying

```
FORTIAD MYPROG
```

would cause MYPROG to be invoked with the DEBUG option.

Under TSO

When you are executing under TSO without ISPF, VS FORTRAN Version 2 Interactive Debug can be invoked with the CALL command:

```
CALL progname 'DEBUG'
```

or with the LOADGO command:

```
LOADGO progname 'DEBUG'
```

To invoke VS FORTRAN programs, you may find it convenient to create a TSO CLIST. If you already have one, you will probably want to modify it to invoke Interactive Debug.

Figure 18 is an example of a simple CLIST that allocates an AFFPRINT file (described later in this chapter), and a dummy AFFON file. It invokes the VS FORTRAN program with or without the DEBUG option. Unless you specify NODEBUG when invoking the CLIST, Interactive Debug will be invoked.

In this example, if the CLIST is invoked without specifying an OPTION parameter, the VS FORTRAN program will be invoked with the DEBUG execution-time parameter. If the CLIST were named FORTIAD, specifying

```
FORTIAD  MYPROG DSN(MYLIB.LOAD)
```

would cause program MYPROG, contained in library *userid*.MYLIB, to be invoked with the DEBUG option specified.

```
PROC 1 MEMBER DSN(FORTRAN.LOAD) OPTION(DEBUG)
CONTROL NOMSG NOFLUSH NOLIST NOSYMLIST NOCONLIST
IF &OPTION = DEBUG THEN DO
   FREE FI(AFFPRINT AFFON)
   ALLOC FI(AFFON) DUMMY
   ALLOC FI(AFFPRINT) DA(&MEMBER..PRINT) SHR
   IF &LASTCC ¬= 0 THEN +
    ALLOC FI(AFFPRINT) DA(&MEMBER..PRINT) NEW CATALOG SPACE(5 5) TRACKS
END
SET RCODE = &LASTCC
CALL '&SYSUID..&DSN.(&MEMBER) ' '&OPTION'
FREE FI(AFFPRINT AFFON)
WRITE RETURN CODE: &RCODE
EXIT
END
```

Figure 18. Sample TSO CLIST to Invoke a VS FORTRAN Program

Entering Commands

In both CMS and TSO, after invoking a VS FORTRAN program with the DEBUG option, you will receive the informational message `VS FORTRAN VERSION 2.2.0 Interactive Debug`, and several lines of copyright information, followed by a WHERE message identifying the statement about to be executed. This is followed by the Interactive Debug prompt FORTIAD, indicating that the VS FORTRAN program has reached the first debugging hook. At this point, execution is temporarily suspended to allow you to enter debugging commands.

Commands are normally entered following the Interactive Debug prompt. Commands may also be issued when execution is suspended because of an input request originating in the VS FORTRAN program while TERMIO IAD is in effect.

This is described further in "Entering Input to a VS FORTRAN Program" on page 47.

When executing Interactive Debug in line mode, a command is limited to 131 characters, including blanks. You may choose not to enter an entire command on one line of the terminal. In this case, you may enter some portion of the command, follow it with the continuation character (-), and then start a new line to enter the remainder of the command. The continuation character is a hyphen (-) and must be the last character entered on the command line before pressing the RETURN key.

When Interactive Debug is awaiting the continuation of a command, you will not receive another prompt until the command has been completely entered and processed.

Using Program Function Keys to Enter Commands

If you are using a 3270-type terminal on VM/SP, you can use program function keys (PF keys) to enter commands in line mode. (It is not possible to define PF keys under TSO in line mode.)

To define PF keys when running Interactive Debug under CMS in line mode, enter the CP command SET PF on the command line. You can use this command to change any current PF key value. For example, you might want to set certain PF keys to Interactive Debug commands that you issue frequently. To set PF key 4 to the NEXT command, enter this on the command line:

```
SYSCMD SET PF4 NEXT
```

Entering Input to a VS FORTRAN Program

Note: This discussion assumes that TERMIO IAD is in effect. If not, then you do not have the ability to enter Interactive Debug commands while a read is pending, and the discussion below is not applicable. For additional information, see "Entering Terminal Input" on page 91.

VS FORTRAN Version 2 Interactive Debug will issue a message before attempting to read input from the terminal. You can enter Interactive Debug commands prior to responding to the program's request for input.

When execution is suspended for terminal input, the FORTIAD prompt will appear following the message:

```
FT05F001 INPUT: PRECEDE INPUT WITH % OR ENTER IAD COMMAND
```

When entering input to a VS FORTRAN program, you must precede it with a percent sign (%). The leading percent sign will not be passed to the VS FORTRAN program. If you need to precede your input with one or more leading blanks, you may do so following the percent sign and your program will receive the leading blanks. To simulate entering a null line, enter %%.

Using a Listing File while Debugging

To view a compiler listing file (or the source file) while debugging, use the SYSCMD command. You can use the SYSCMD command to list the file, to edit the file, or to execute other system commands. For more information, see "SYSCMD Command" on page 205.

Using Optional Debugging Files or Data Sets

The following sections discuss two optional files (or data sets), and how to define them when invoking Interactive Debug in line mode.

Specifying Program Units to Be Debugged (AFFON)

When Interactive Debug is invoked, you can set breakpoints in any debuggable program units. Because there is some overhead associated with this ability, it will be more efficient to exclude from debugging those program units that are known to be error free. You can also exclude part of a program unit, which may be helpful if there are heavily-executed sections that are error free.

The AFFON file or AFFON data set is a sequential file containing a list of program unit names, up to 31 characters in length, to be debugged. AFFON is read by Interactive Debug; it must be defined prior to execution of the VS FORTRAN program.

In a CMS environment, the EXEC used to execute VS FORTRAN programs should contain a FILEDEF statement for the AFFON file; for example:

```
FILEDEF AFFON DISK progname INCLUDE A
```

In a TSO environment, the CLIST used to execute VS FORTRAN programs, should contain an ALLOCATE command for the AFFON data set; for example:

```
ALLOCATE FI(AFFON) DA(progname.INCLUDE)
```

where *progname* is the name of the program you want to debug.

The AFFON file contains a list of program unit names to be selected for debugging, and may contain comment entries identified by an asterisk (*) in the first record position. Program unit names are assumed to be the first character string in the record (terminated by a blank), and may appear after initial leading blanks.

Each program unit name can be followed by the data set name containing the source listing for a program unit, as well as a list of statement numbers (ISNs or sequence numbers) and statement number ranges. (ISNs are the default, unless you specified SDUMP(SEQ) when you compiled the program unit.) If any statement numbers are specified, only statements falling within the specified ranges, or included in the list of statement numbers, are selected for debugging. If no statement numbers are specified, all executable statements are selected. Note that statement labels are **not** allowed.

The syntax of each entry in the AFFON file is as follows:

```
unitname ['dataset[(member)]'][n[:n]]...
```

or

```
unitname ['dataset[(member)]'][ENTRY]
```

or

```
unitname ['dataset[(member)]'][NONE]
```

Blanks or commas separate entries in the list of statement number ranges. The program unit name and the list of statement number ranges must all fit within one record. The ENTRY parameter specifies that only ENTRY and EXIT hooks are to be placed in the program unit. This is especially helpful if you want to increase the accuracy of timing information. The NONE parameter allows data set names to be specified, but bypasses inclusion of debugging hooks.

For example, to place debugging hooks between statements 6 and 16 and at statement 18 of SUB1, you could make the following entry, under CMS, in your AFFON file:

```
Sub1     'SUB1 Listing *'          6:16  18
```

In the above example, SUB1 LISTING * is defined as the default listing. ENTRY and EXIT hooks are always placed in the program unit whenever any statement number ranges are specified.

An equivalent example for TSO would be:

```
Sub1     'userid.sub1.list'        6:16  18
```

Note that there may be executable statements in the specified ranges that cannot have debugging hooks placed on them because of optimization or vectorization. To find out which statements have had hooks placed on them, you can use the LISTFREQ command.

Program units that do not meet the requirements for debugging will not have program hooks inserted, even though they may be in the AFFON list.

One way to create an AFFON file is to edit the map produced when the module to be executed is built. Normally, this is the linkage editor map or the CMS load map. The map can be edited by placing an asterisk before any program unit name that is not to be debugged, nominating the other program units for debugging. You can then add in a list of statement numbers or statement number range specifications.

Note: Because the CMS load map truncates long names to 7 characters, you will have to replace truncated names with complete names as required by Interactive Debug.

AFFON can have any record format, and entries can be in lower case. The logical record length can be any value up to 255 bytes. If the file exists but has incorrect attributes, you will receive an error message stating that the AFFON file cannot be found. AFFON can contain references to sequence numbers for VS FORTRAN programs compiled with SDUMP(SEQ), but AFFON itself cannot be sequenced.

If the AFFON data set is found and contains at least one entry that is not a comment, only the valid program unit names or ranges found in the data set or the program unit compiled with the test option will be considered for debugging. Otherwise, all VS FORTRAN program units will be considered for debugging.

For compiler and library restrictions that may make a program unit nondebuggable, see "Interactive Debug Programming Requirements" on page 5.

Specifying a Print File or Data Set (AFFPRINT)

Certain VS FORTRAN Version 2 Interactive Debug commands allow you to specify a PRINT keyword causing command output to be sent to a print data set rather than to your terminal. This may prove useful, for example, when you are listing the contents of a large array and want to keep this output separate from the normal log. This print data set is referred to as the AFFPRINT data set or the AFFPRINT file.

AFFPRINT must be defined prior to executing a VS FORTRAN program.

In a CMS environment, the EXEC used to execute VS FORTRAN programs should contain a FILEDEF statement for the AFFPRINT file.

```
FILEDEF AFFPRINT DISK progname LIST A
```

In a TSO environment, the CLIST used to execute VS FORTRAN programs should contain an ALLOCATE command for the AFFPRINT data set.

```
ALLOCATE FI(AFFPRINT) DA(progname.PRINT)
```

Note: To prevent an existing AFFPRINT file from being overwritten, you must rename it before executing a program with the same name.

Chapter 4. Using Interactive Debug in Batch Mode

VS FORTRAN Version 2 Interactive Debug can be run in **batch mode**, which creates a noninteractive debugging session. Batch mode can be run in either a CMS or a TSO environment.

In batch mode, Interactive Debug takes its input from a file or data set with the DD name AFFIN, and writes its normal output to one with the DD name AFFOUT. During a batch session, you cannot interact with the batch job from your terminal. Commands that require user interaction (such as prompt panels) cannot be used.

You might want to run a debugging session in batch mode if:

- You want to restrict the resources used. Batch mode generally uses fewer resources than interactive mode.
- You have a program that might tie up your terminal for long periods of time. If you use batch mode, you can continue to use your terminal for other work while the batch job runs.
- You are using Interactive Debug to collect performance or execution data about your program. For example, batch mode might be helpful if you want to use LISTFREQ to get statement frequency information, but you do not want to do any debugging.

Be aware that the following Interactive Debug commands are NOT AVAILABLE in batch mode:

AUTOLIST	PREVDISP
COLOR	PROFILE
HELP	REFRESH
LISTINGS	RESTART
MOVECURS	SEARCH
POSITION	WINDOW

Invoking Interactive Debug

The invocation procedure you use to start a batch session will depend on the batch procedures set up for you by your installation. Be sure you understand how batch mode is invoked on your system before running Interactive Debug in batch mode.

Under CMS

There are many batch facilities that can be run on CMS (CMSBATCH, BATCHMON, VMBATCH, and so on). Interactive Debug considers batch mode to occur whenever there is no physical terminal attached to the console, which is true of all the batch facilities listed above. The following gives an overview of an EXEC or job file for a batch job to be run on CMS:

1. The first part of the job is usually a set of control statements for the batch machine, such as a job statement, accounting information, console routing control, and resource limit specifications.

2. The job probably needs CMS commands to link and access the disks containing the application program, the Interactive Debug product, input files, and any other EXECs or programs needed.

3. Next, the job needs commands to run your application with the DEBUG option. These are similar to the commands in the sample EXEC to invoke Interactive Debug in line mode (FORTIAD EXEC), but must include definitions for the AFFIN and AFFOUT files. The sample FORTIAD EXEC is described in Chapter 3, "Using Interactive Debug in Line Mode" on page 41.

   If you have a FORTIAD EXEC, modify it as above if necessary, and then enter the following in your job file:

   ```
   EXEC FORTIAD myprog
   ```

   If you want to include the information contained in FORTIAD directly in your job file, it might look like that shown in Figure 19.

```
GLOBAL TXTLIB VSF2FORT CMSLIB
GLOBAL LOADLIB VSF2LOAD
FILEDEF AFFIN DISK myprog AFFIN *
FILEDEF AFFOUT DISK myprog AFFOUT A
FILEDEF AFFON    DUMMY
FILEDEF AFFPRINT DISK myprog AFFPRINT A
LOAD myprog (CLEAR
START * DEBUG
```

Figure 19. Sample Commands for a Batch Debugging Session under CMS

   These sample statements invoke the program with the DEBUG option. They allocate an AFFPRINT file, but not an AFFON file. They assume you are starting with TEXT files you want to link with the VS FORTRAN Version 2 Library to run in load mode.

4. Finally, you need CMS commands to send back any output files that were written to disk.

   For example, to send the AFFOUT file to your reader, you might use these commands:

```
CP SPOOL PUN TO userid NOCONT
PUNCH myprog AFFOUT
```

To send the AFFPRINT file to the printer, you might use these commands:

```
CP SPOOL PRT FOR userid
PRINT myprog AFFPRINT
```

When the EXEC or job file is complete, submit it to the batch machine using the procedures set up for your installation.

Under MVS

In MVS, Interactive Debug considers batch mode to occur whenever no physical terminal is attached. There are two ways to run Interactive Debug under MVS: with TSO and without TSO.

To set up the batch job using TSO, you might use JCL that looks like that shown in Figure 20 on page 54.

```
//SAMPLE   JOB (accounting-information),'programmer-name',
// MSGLEVEL=1,MSGCLASS=Z,USER=userid,
// TIME=(0,5),NOTIFY=userid,CLASS=A,
// PASSWORD=password
//*
//*********************************************************************
//* INVOKE THE TMP..                                                  *
//*********************************************************************
// EXEC PGM=IKJEFT01,DYNAMNBR=100,REGION=2048K
//*
//*********************************************************************
//* DESCRIPTION OF FORTRAN PROGRAM DATA SETS                          *
//*********************************************************************
//FT06F001 DD SYSOUT=*
//SYSUDUMP DD SYSOUT=*
//*
//*********************************************************************
//* DESCRIPTION OF DEBUGGER DATA SETS                                 *
//*********************************************************************
//AFFON  DD DUMMY                                  /* INCLUDE FILE    */
//AFFPRINT DD SYSOUT=*                             /* PRINT FILE      */
//AFFOUT DD DSN=log-name,DISP=OLD                  /* OUTPUT FROM IAD */
//AFFIN DD DSN=restart-name,DISP=SHR               /* DEBUG SCRIPT    */
//*
//*********************************************************************
//* LOCATION OF FORTRAN PROGRAM AND LIBRARIES                         *
//*********************************************************************
//STEPLIB DD DSN=SYS1.VSF2FORT,DISP=SHR             /* FORTRAN LIB    */
//*
//*********************************************************************
//* DESCRIPTION OF TSO SESSION INPUT AND OUTPUT                       *
//*********************************************************************
//SYSTSPRT DD SYSOUT=*,
// DCB=(RECFM=F,LRECL=255,BLKSIZE=255)
//SYSTSIN DD *
 /* INVOKE FORTRAN PROGRAM AND PASS DEBUG PARAMETER */
    CALL 'userid.TESTCASE.LOAD(program)' 'DEBUG'
/*
```

Figure 20. Sample JCL for a Batch Debugging Session under MVS with TSO

If you are not running under TSO, your JCL for the batch job would be similar to the sample shown in Figure 21 on page 55.

```
//SAMPLE   JOB (accounting-information),'programmer-name',
// MSGLEVEL=1,MSGCLASS=Z,USER=userid,
// TIME=(0,5),NOTIFY=userid,CLASS=A,
// PASSWORD=password
//*
//*********************************************************************
//* INVOKE THE APPLICATION PROGRAM                                    *
//*********************************************************************
// EXEC PGM=member,PARM='DEBUG',REGION=2048K
//*
//*********************************************************************
//* DESCRIPTION OF FORTRAN PROGRAM DATA SETS                          *
//*********************************************************************
//FT06F001 DD SYSOUT=*
//SYSUDUMP DD SYSOUT=*
//*
//*********************************************************************
//* DESCRIPTION OF DEBUGGER DATA SETS                                 *
//*********************************************************************
//AFFON  DD DUMMY                                 /* INCLUDE FILE    */
//AFFPRINT DD SYSOUT=*                            /* PRINT FILE      */
//AFFOUT DD DSN=log-name,DISP=OLD                 /* OUTPUT FROM IAD */
//AFFIN DD DSN=restart-name,DISP=SHR              /* DEBUG SCRIPT    */
//*
//*********************************************************************
//* LOCATION OF FORTRAN PROGRAM AND LIBRARIES                         *
//*********************************************************************
//STEPLIB DD DSN=SYS1.VSF2FORT,DISP=SHR            /* FORTRAN LIB    */
//        DD DSN=userid.TESTCASE.LOAD,DISP=SHR
//*
```

Figure 21. Sample JCL for a Batch Debugging Session under MVS without TSO

The JCL in Figure 20 on page 54 and in Figure 21 define AFFPRINT, AFFOUT, and AFFIN data sets. No AFFON data set is used. You can modify these DD cards for the combination of data sets you need.

Wherever lowercase variables, such as `programmer-name` or `log-name` are shown, you need to substitute the appropriate information. (You can usually substitute an asterisk (*) for your user ID and password if you prefer.) You will need to set up your own accounting information, and DD cards with appropriate names for your program data sets. You also need to designate the location and name of your program, and the libraries needed.

If AFFIN contains any SYSCMD or TSO commands, you must run the TSO terminal monitor program.

Running a Batch Debugging Session

In batch mode, all Interactive Debug prompts are suppressed. Whenever a simulated terminal input line is read, it is echoed to the simulated terminal output, prefixed with an equal sign and asterisk (=*). If the AFFIN file does not exist or the AFFOUT file is not defined, the program is terminated.

Standard corrective action is taken for all VS FORTRAN errors (unless the VS FORTRAN program calls ERRSET to change them). This will cause the program to terminate for unrecoverable errors.

Note: Interactive Debug avoids any real terminal interaction in batch mode. However, it cannot guard against interactions required by the application program or by SYSCMD commands. It is your responsibility to restrict the use of interactive commands during batch sessions.

Specifying the Input File or Data Set (AFFIN)

When running Interactive Debug in batch mode, all commands to be executed during the batch debugging session must be entered in a single input file. The input file or data set must be defined as AFFIN. It must exist prior to executing the VS FORTRAN program.

Remember that you cannot enter any interactive commands, such as HELP. SYSCMD and CMS or TSO commands are not permitted without operands. You must specify the command and any required operands on a single line; you will not be prompted interactively for command operands. Also, do not specify any commands that require interaction.

Under MVS, SYSCMD or TSO commands are permitted only if the TSO Terminal Monitor Program is running.

The file will be read until end-of-file or a QUIT command is encountered. (If end-of-file is reached, a QUIT command is forced.)

The AFFIN file must have a RECFM of F or FB and an LRECL of 80. It has the same format as an output file, and may even be a previous output file. If it is a previous output file, Interactive Debug will ignore any output contained in the file (anything preceded by only an equal sign, or only an asterisk.) It accepts any line preceded by =* , or by nothing, as input.

Note: Sequence numbers are **not** allowed in columns 73 through 80.

Although it is possible in principle to use the log file from a full screen debugging session as AFFIN input for a batch debugging session, you should be aware that there are some differences.

- TERMIO is likely to have different effects in batch. Under MVS it is not possible to connect a data set to a terminal device in batch. However, you can use the DEBUNIT execution-time option to specify one or more units that will be treated as terminals. It is then possible, for example, for you to use your output from a full screen session as a restart file, with TERMIO set to IAD.

  In CMS, you can still use TERMINAL in a FILEDEF command, but you should be careful not to issue TERMIO LIBRARY if there will be any terminal input, because your program would then attempt to actually get terminal input and the batch job would stall.

You may want to add TERMIO MSG and NOMSG commands in order to get some notification as the job progresses. (This should normally be used with restraint.)

- Error handling is different in batch. In order to avoid unplanned interactions, the debugger always forces standard fixup for errors in batch mode. Thus ERROR EXIT will never cause an exit to be taken. Error limits are in effect as if Interactive Debug were not present.

- SYSCMD commands with no system command specified are considered an error in batch mode (but cause no harm otherwise). You should also avoid any system commands that might themselves require interaction.

- Prompts are suppressed in batch. For example, GO with a statement identifier does not prompt for confirmation in batch mode, even if an optimized program unit is being debugged.

Including Program Input in AFFIN

When TERMIO IAD is in effect, all terminal input is obtained from AFFIN, preceded with %, and interspersed with the IAD comments.

In batch mode under MVS, you do not have a terminal available, so it is therefore impossible to connect VS FORTRAN files to a terminal. However, by specifying the DEBUNIT execution-time option, which specifies a device to be treated as a terminal, you can use the TERMIO command with MVS batch.

The DEBUNIT option may already be set up for you as a local default whenever you specify the DEBUG option, or you may need to specify it at execution-time when you run your program. If you need to specify it, the format is as follows:

```
DEBUNIT(S1[,S2,...])
```

where S is a unit number (such as 5), or an inclusive range of unit numbers (such as 35-40).

Note that in CMS, the commas must be replaced by blanks, unless the program will be invoked by an EXEC2 or REXX EXEC and uses the extended PLIST facilities. If I/O is to be issued to the units, the job stream must also include a FILEDEF (in CMS) or a DD card (in TSO) for each unit.

Remember that, when running in batch, the program must get its input from real data sets, and must send its output to real data sets. In CMS batch, you should not specify TERMIO LIBRARY if there will be any terminal input, because this requests interaction and will cause the batch job to fail.

For more information on TERMIO, see "Entering Terminal Input" on page 91 and "TERMIO Command" on page 207.

Specifying an Output File or Data Set (AFFOUT)

Interactive Debug creates a log of its activity for you to examine; you can view this output after the batch debugging session has completed. The output data set is referred to as the AFFOUT data set or the AFFOUT file. AFFOUT does not have to exist prior to executing the VS FORTRAN program, but must be defined. The DD name identifies the file to Interactive Debug.

The output file has the following characteristics:

- All Interactive Debug I/O is logged, along with all VS FORTRAN terminal I/O that occurs while TERMIO IAD is in effect.
- The file is created with a RECFM of FB and an LRECL of 80 (and a BLKSIZE of 800 for TSO).
- Each line is preceded with an equal sign (=).
- Each line of input is preceded by an asterisk (*) following the equal sign.

After Ending a Debugging Session

It is your responsibility to make sure that AFFOUT and any other batch output files are returned to you or sent to an appropriate printer. The batch output files are:

- AFFOUT for debug output
- AFFPRINT for debug print output
- Any output files that your VS FORTRAN program writes.

Using Optional Debugging Files or Data Sets

The following sections discuss two optional files (or data sets), and how to define them when invoking Interactive Debug in batch mode.

Specifying a Print File or Data Set (AFFPRINT)

Certain VS FORTRAN Version 2 Interactive Debug commands allow you to specify a PRINT keyword. You can use the PRINT keyword to send output to a print data set rather than AFFOUT. This may prove useful, for example, when you are listing the contents of a large array and want to keep that output separate from normal debug output. This print data set is referred to as the AFFPRINT data set or the AFFPRINT file.

Because AFFPRINT is an output file, it does not need to exist prior to executing your VS FORTRAN program. However, it must be defined with a FILEDEF or DD statement. The DD name identifies the file to Interactive Debug.

Specifying Program Units to be Debugged (AFFON)

When Interactive Debug is invoked, you can set breakpoints in any debuggable program units. Because there is some overhead associated with this ability, it will be more efficient to exclude from debugging those program units that are known to be error free. You can also exclude part of a program unit, which may be helpful if there are heavily-executed sections that are error free.

The AFFON file or AFFON data set is a sequential file containing a list of program unit names, up to 31 characters in length, to be debugged. AFFON is read by Interactive Debug; it must exist before execution of the VS FORTRAN program, and must be accessible to the batch job.

AFFON contains a list of program unit names to be selected for debugging, and may contain comment entries identified by an asterisk (*) in the first record position. Program unit names are assumed to be the first character string in the record (terminated by a blank), and may appear after initial leading blanks.

Each program unit name can be followed by the data set name containing the source listing for a program unit as well as a list of statement numbers (ISNs or sequence numbers), or statement number ranges. (ISNs are the default, unless you specified SDUMP(SEQ) when you compiled the program unit.) If any statement numbers are specified, only statements falling within the specified ranges, or included in the list of statement numbers, are selected for debugging. If no statement numbers are specified, all executable statements are selected. Note that statement labels are **not** allowed.

The syntax of each entry in the AFFON file is as follows:

```
unitname ['dataset[(member)]'][n[:n]]...
```

or

```
unitname ['dataset[(member)]'][ENTRY]
```

or

```
unitname ['dataset[(member)]'][NONE]
```

Blanks or commas separate entries in the list of statement number ranges. The program unit name and the list of statement number ranges must all fit within one record. The ENTRY form indicates that only entry and exit hooks are to be placed in the program unit. This is especially helpful if you want to increase the accuracy of timing information. The NONE parameter allows data set names to be specified, but by passes inclusion of debugging hooks.

For example, to specify that you want to place debugging hooks at the ENTRY and EXIT, on the statements in SUB1 whose statement numbers fall between 6 and 16, and at statement 18, you could make the following entry, under CMS, in your AFFON file:

```
Sub1     'SUB1 Listing *'            6:16  18
```

In the above example, SUB1 LISTING * is defined as the default listing. ENTRY and EXIT hooks are always placed in the program unit whenever any statement number ranges are specified.

An equivalent example for TSO would be:

```
Sub1     'userid.sub1.list'          6:16  18
```

Note that there may be executable statements in the specified ranges that cannot have debugging hooks placed on them because of optimization or vectorization. To find out which statements have had hooks placed on them, you can use the LISTFREQ command.

Program units that do not meet the requirements for debugging will not have program hooks inserted, even though they may be in the AFFON list.

AFFON can have any record format; entries can be in lower case. The logical record length can be any value up to 255 bytes. If the file exists but has incorrect attributes, you will receive an error message stating that the AFFON file cannot be read. AFFON can contain references to sequence numbers for VS FORTRAN programs compiled with SDUMP(SEQ), but AFFON itself cannot be sequenced.

If the AFFON data set is found and contains at least one entry that is not a comment, only the valid program unit names found in the data set or the program unit compiled with the test option will be considered for debugging. Otherwise, all VS FORTRAN program units will be considered for debugging.

For compiler and library restrictions that may make a program unit nondebuggable, see "Interactive Debug Programming Requirements" on page 5.

Chapter 5. A Sample Debugging Session

This section is intended for those who are unfamiliar with VS FORTRAN Version 2 Interactive Debug. Through a sample debugging session, it introduces you to some basic concepts and several commands.

Sample Program

Assume you want to debug the VS FORTRAN program shown in Figure 22. The program includes a subroutine, DIVIDE, which divides an array used as the dividend by a second array used as the divisor, and returns the result back to the main program in a third array. The three integer arrays each have 10 elements. The main program, named SAMPLE, uses a DO loop to assign values to the arrays for the dividend and divisor, named A1 and A2 respectively.

If you would like to follow this sample session on your terminal, enter the program as shown in Figure 22.

```
@PROCESS
      PROGRAM SAMPLE
      INTEGER A1(10),A2(10),A3(10)
      DO 20 I=1,10
        A1(I)=I+1
        A2(I)=I-1
20    CONTINUE
      CALL DIVIDE (A1,A2,A3)
      WRITE (6,30)(A3(I),I=1,10)
30    FORMAT (' ',I5)
      STOP
      END
@PROCESS
      SUBROUTINE DIVIDE (DIVEND, DIV, RES)
      INTEGER DIVEND(10),DIV(10),RES(10)
      DO 10 I=1,10
        RES(I)=DIVEND(I)/DIV(I)
10    CONTINUE
      RETURN
      END
```

Figure 22. Program Source File for SAMPLE Program

Now compile the program with the SDUMP and OPT(0) options, and execute the program under the control of Interactive Debug by specifying the execution-time option, DEBUG. If you are using ISPF, refer to Chapter 2, "Using Interactive Debug with ISPF" on page 9, for more detail. If you are operating in line mode, refer to Chapter 3, "Using Interactive Debug in Line Mode" on page 41.

VS FORTRAN Version 2 Interactive Debug will be invoked, allowing you to begin a debugging session. Figure 23 on page 63 shows how the compiler listing for the SAMPLE program might appear. During the debugging session, this listing will be useful for determining statement identifiers for breakpoints. A statement identifier can be an ISN, a sequence number in columns 73 through 80, or a statement label.

The compiler assigns a number known as the internal statement number (ISN) to each statement in the program. For example, the first executable statement in Figure 23 has an ISN of 3. To reference that statement in a VS FORTRAN Version 2 Interactive Debug command, you would normally use this ISN. You can use a qualifier to distinguish ISNs with the same number in different program units.

If you specify `SDUMP(SEQ)` when you compile your program, you must always refer to the sequence numbers in columns 73 through 80 instead of the compiler-generated ISNs. However, for this sample program, we will assume you are using ISNs as your statement numbers.

Statements that have a user-specified statement label in columns 1 through 5, such as the CONTINUE statement in Figure 22 on page 61, can also be referenced by that statement label. When a statement label is used, the number must be preceded by a slash (/). The CONTINUE statement in SAMPLE may be referenced as `/20` as well as `6`.

```
LEVEL 2.2.0 (JUNE 1987)        VS FORTRAN            OCT 09, 1987  10:42:15
REQUESTED OPTIONS (PROCESS):
OPTIONS IN EFFECT: NOLIST NOMAP NOXREF NOGOSTMT NODECK SOURCE TERM OBJECT FIXED
                   TRMFLG SRCFLG NOSYM NORENT SDUMP(ISN) AUTODBL(NONE) NOSXM
                   NOVECTOR IL NOTEST NODC NOICA NODIRECTIVE OPT(0) LANGLVL(77)
                   NOFIPS  FLAG(I) NAME(MAIN) LINECOUNT(60) CHARLEN(500)

 IF DO   ISN    *....*...1.........2.........3.........4.........5.........6.....
           1          PROGRAM SAMPLE
           2          INTEGER A1(10),A2(10),A3(10)
           3          DO 20 I=1,10
     1     4            A1(I)=I+1
     1     5            A2(I)=I-1
     1     6    20      CONTINUE
           7          CALL DIVIDE (A1,A2,A3)
           8          WRITE (6,30)(A3(I),I=1,10)
           9    30    FORMAT (' ',I5)
          10          STOP
          11          END
*STATISTICS*    SOURCE STATEMENTS = 11, PROGRAM SIZE = 964 BYTES,
                PROGRAM NAME = SAMPLE     PAGE:     1.
*STATISTICS*     NO DIAGNOSTICS GENERATED.
**SAMPLE** END OF COMPILATION 1 ******
LEVEL 2.2.0 (xxxx 1987)        VS FORTRAN            JAN 09, 1987  10:42:15
REQUESTED OPTIONS (PROCESS):
OPTIONS IN EFFECT: NOLIST NOMAP NOXREF NOGOSTMT NODECK SOURCE TERM OBJECT FIXED
                   TRMFLG SRCFLG NOSYM NORENT SDUMP(ISN) AUTODBL(NONE) NOSXM
                   NOVECTOR IL NOTEST NODC NOICA NODIRECTIVE OPT(0) LANGLVL(77)
                   NOFIPS  FLAG(I) NAME(MAIN) LINECOUNT(60) CHARLEN(500)
 IF DO   ISN    *....*...1.........2.........3.........4.........5.........6.....
           1          SUBROUTINE DIVIDE (DIVEND,DIV,RES)
           2          INTEGER DIVEND(10),DIV(10),RES(10)
           3          DO 10 I=1,10
     1     4            RES(I)=DIVEND(I)/DIV(I)
     1     5    10      CONTINUE
           6          RETURN
           7          END
*STATISTICS*    SOURCE STATEMENTS = 7, PROGRAM SIZE = 828 BYTES,
                PROGRAM NAME = DIVIDE     PAGE:     2.
*STATISTICS*     NO DIAGNOSTICS GENERATED.
**DIVIDE** END OF COMPILATION 2 ******
LEVEL 2.2.0 (xxxx 1987)        VS FORTRAN            JAN 09, 1987  10:42:15
SUMMARY OF MESSAGES AND STATISTICS FOR ALL COMPILATIONS
*STATISTICS*    SOURCE STATEMENTS = 11, PROGRAM SIZE = 964 BYTES,
                PROGRAM NAME = SAMPLE     PAGE:     1.
*STATISTICS*     NO DIAGNOSTICS GENERATED.
**SAMPLE** END OF COMPILATION 1 ******
*STATISTICS*    SOURCE STATEMENTS = 7, PROGRAM SIZE = 828 BYTES,
                PROGRAM NAME = DIVIDE     PAGE:     2.
*STATISTICS*     NO DIAGNOSTICS GENERATED.
**DIVIDE** END OF COMPILATION 2 ******
******* SUMMARY STATISTICS ******* 0 DIAGNOSTICS GENERATED.
                                   HIGHEST SEVERITY CODE IS 0.
```

Figure 23. Program Source Listing for SAMPLE Program

Sample Debugging Session

Now that you have compiled the program and invoked Interactive Debug, you are ready to begin the debugging session. Before executing the first statement, VS FORTRAN Version 2 Interactive Debug suspends execution and allows you to enter commands. The log tells you that execution is suspended at ISN 3:

```
WHERE: SAMPLE.3
```

Interactive Debug will prompt you for commands wherever execution is suspended. You can enter commands in upper- or lowercase letters, but all system responses will appear in uppercase letters.

To begin our debugging session, let's list the program units available for debugging:

```
listsubs
```

The response to LISTSUBS should look like this:

```
PROGRAM UNIT     COMPILER    OPT    HOOKED    TIMING
SAMPLE          VSF 2.2.0     0       YES       OFF
DIVIDE          VSF 2.2.0     0       YES       OFF
```

Now let's try executing the program. The GO command begins execution at the next executable statement. Because we are not aware of any errors in the program, let's try executing it without setting any breakpoints.

```
go
```

Unfortunately, there is an error in the program. You should receive an error message that looks like this:

```
ERRMSG=> AFB209I VFNTH : PROGRAM INTERRUPT - FIXED-POINT DIVIDE EXCEP
ERRMSG=> TION
ERRMSG=>          VFNTH : PSW 4009A20205DC
ERRMSG=>          VFNTH : LAST EXECUTED FORTRAN STATEMENT IN PROGRAM D
ERRMSG=> IVIDE AT ISN 4 (OFFSET 000214).
INFMSG=> USER ERROR CORRECTIVE ROUTINE ENTERED.
ERROR EXIT: ERROR 209 AT DIVIDE.4
```

Now we know that there is a problem at ISN 4 in program unit DIVIDE. You may want to look at ISN 4 in the source listing while continuing to debug.

There are several ways to look at the source listing without exiting the debugging session, depending on your operating environment. Under ISPF Version 2, the source listing will automatically be shown in a window if you have defined the window size (using the WINDOW command) and have identified the source listing file on the listings data set specification panel. To see the listings data set specification panel, enter the LISTINGS command or type a question mark (?) in the first character of the Q field of the execution panel header.

If you are using ISPF with PDF, you can split the screen with the SPLIT function of ISPF, and use PDF's BROWSE or EDIT facilities. For those not running under ISPF and PDF, the VS FORTRAN Version 2 Interactive Debug SYSCMD

command can be used to invoke an editor. For example:

```
syscmd edit sample.for (TSO)

syscmd xedit sample fortran a (CMS)
```

Because ISN 4 contains several variables, it might help to look at the values of the variables there. We can do this using the LIST command. (We are already in program unit DIVIDE, so we do not need to qualify any of the variables.)

```
list (i,divend(i),div(i),res(i))
```

In the log file, we see the value of I, and the first elements of DIVEND, DIV, and RES:

```
DIVIDE.I          =         1
DIVIDE.DIVEND(1)  =         2
DIVIDE.DIV(1)     =         0
DIVIDE.RES(1)     =         0
```

Look at the value of our divisor, DIV. At this point in the program, DIV is 0, an invalid value for a division operation. But how did the value become 0? To find out, we need to restart the program and check what it does at earlier points. Under ISPF Version 2, you can simply type RESTART, but let's use a longer method that will also work in line mode. We will tell Interactive Debug to go back to ISN 3 in the main program, SAMPLE.

Because the GO command cannot switch from one program unit to another, we must first get back to SAMPLE before we can GO to ISN 3. On our listing, we see that ISN 8 is the first statement in SAMPLE after returning from the call to DIVIDE. Let's set a breakpoint there. To suspend execution of the program just before the WRITE statement is executed, use the AT command. We need to qualify the ISN with the name of the main program because the currently qualified unit is still DIVIDE:

```
at sample.8
```

Now resume execution:

```
go
```

At the WRITE statement, execution will be suspended and you will receive the following messages:

```
INFMSG=> STANDARD CORRECTIVE ACTION TAKEN. EXECUTION CONTINUING.
AT: SAMPLE.8
```

The error in DIVIDE has been temporarily corrected, and execution has resumed up to ISN 8. The AT message indicates where execution is now suspended by displaying the ISN of the statement and the name of its program unit (in this case, SAMPLE, the main program).

By using the GO command with a statement identifier, the initialization loop can be executed again. We can check the values in the three arrays just before DIVIDE is called. To do this, we should set a breakpoint at the statement labeled 20, and use a command list to list the values of the variables at this point:

```
at /20 (list (i,a1(i),a2(i),a3(i)))
```

Before resuming execution, we can verify that we set the correct breakpoints by listing them:

```
listbrks
```

The output from LISTBRKS should be:

```
CURRENT BREAKPOINTS:
  SAMPLE.6/20
  SAMPLE.8
CURRENT WHEN CONDITIONS:
  NONE
CURRENT HALT STATUS: OFF
```

We see that a breakpoint is indeed set at ISN 6, which is also statement label 20, in SAMPLE. So we are ready to resume execution at the beginning of the array initialization loop, ISN 3.

```
go 3
```

Execution will be suspended at ISN 6, and the values of the variables listed:

```
AT: SAMPLE.6/20
SAMPLE.I      =           1
SAMPLE.A1(1)  =           2
SAMPLE.A2(1)  =           0
SAMPLE.A3(1)  =           2
```

Again, we can see that the value in the second array, A2, which is referenced indirectly by subroutine DIVIDE as DIV, is 0. For now, let's temporarily correct this value. We can use the SET command to change the value for the rest of the debugging session, but later we will need to modify the actual program.

```
set a2(1) = 1
```

To execute the rest of the program with the new value and end the debugging session, enter:

```
off *
go
```

Now the SAMPLE program will produce the following output to the terminal:

```
FT06F001       2
FT06F001       3
FT06F001       2
FT06F001       1
FT06F001       1
FT06F001       1
FT06F001       1
FT06F001       1
FT06F001       1
FT06F001       1
PROGRAM HAS TERMINATED; RC = (0)
```

The last message indicates that the program has completed execution, in this case as the result of the STOP statement. Because the arrays are integer, the values are truncated. The output is correct up to the truncated value, so you can assume that the value of A2 was the bug in the program.

You can terminate the debugging session by entering the QUIT command, which will return you to the system (ISPF, CMS or TSO) from which you initiated debugging. You may terminate the debugging session at any time by entering the QUIT command.

To correct the problem, you must **edit the source program** so that no value of A2 (or DIV) is ever 0. Then save the new version of the program. To run the edited version, you must terminate the current debug session, recompile the program, and then reexecute it.

Chapter 6. Using Some Common Interactive Debug Commands

This section describes some of the common debugging tasks you may need to perform, and explains the Interactive Debug commands that will help you perform them.

Examples of the following commands are used in this chapter:

ANNOTATE	LISTSAMP
AT	LISTSUBS
BACKSPACE	LISTTIME
CLOSE	NEXT
DESCRIBE	OFFWN
ENDDEBUG	QUALIFY
ENDFILE	RECONNECT
ERROR	REWIND
FIXUP	SYSCMD
HALT	TERMIO
LIST	TIMER
LISTBRKS	TRACE
LISTFREQ	WHEN
	WHERE

For the complete syntax of all commands and more detail, see Chapter 8, "Interactive Debug Commands" on page 113.

Displaying Information about Debuggable Program Units

Most debugging activities, such as displaying variables, can only be performed on program units that are considered debuggable by Interactive Debug. To be debuggable, a program unit must be compiled with the SDUMP option. In addition, it must be in storage at the time the VS FORTRAN Library is initialized.

Note: The reentrant part of a program unit compiled with the RENT option need not be in storage at this time.

If you want to see which program units are debuggable, you can use the LISTSUBS command.

The following is a sample of the output produced by LISTSUBS:

```
PROGRAM UNIT      COMPILER     OPT  HOOKED  TIMING
MAINLINE         VSF  2.2.0   V2     YES     ON
SUBBUILD         VSF  1.4.0    3      NO     OFF  RENT NOT LOADED
SUBDOWN          VSF  (TEST)   0     YES     OFF
SUBREFIT         VSF  1.3.1    1      NO     ON
```

In this example, we see that the program unit MAINLINE was compiled using VS FORTRAN Version 2, Release 2, identified as VSF 2.2.0. SUBBUILD was compiled with Release 4 of VS FORTRAN Version 1, and SUBREFIT with Release 3.1 of VS FORTRAN Version 1. `VSF (TEST)` tells us that SUBDOWN was compiled prior to VS FORTRAN Version 1 Release 4, and the TEST option was specified. In this case, it is not possible to determine the VS FORTRAN release level.

Notice that, for MAINLINE, the OPT column specifies `V2`. This indicates vectorization level 2. When the OPT column shows a vectorization level (V1 or V2), the optimization level is always 3.

In the HOOKED column, "YES" means that hooks are installed at entry and exit points and possibly at some or all statement boundaries as well. You can set breakpoints only in program units that have hooks. The hook settings are controlled by the AFFON file. "NO" in the HOOKED column indicates that no hooks are installed in the program unit. The TIMING column indicates whether the TIMER command has been activated for each program unit listed. This column may also be followed by an indication of the load status for reentrant programs. In our example, SUBBUILD indicates `RENT NOT LOADED`, meaning that the program unit has not yet been called, and has not been located (although it may actually be in storage).

Referring to Statements or Variables in Other Program Units

Programs often contain more than one program unit. A program unit is defined as a main program, a function subprogram, or a subroutine subprogram. Each of these units has its own set of variables, but variables in different program units may have the same names.

A similar situation exists with statement identification; two statements in different program units may have the same statement label or be assigned the same ISN by the VS FORTRAN compiler. On Interactive Debug commands that refer to statements or variables, you can specify the program unit as a qualifier (in the form of the program unit name followed by a period). If no qualifier is specified in a command that references statements or variables, Interactive Debug resolves these references using the current program qualification.

The current program qualification is normally the program unit that is executing (or in which execution is suspended). However, you can change the current qualification by issuing the QUALIFY command.

Note: Each time execution is resumed, Interactive Debug will reset the qualification to the program unit currently executing.

Displaying the Current Program Qualification

To see which program unit is currently qualified, enter the QUALIFY command with no arguments:

```
qualify
```

The response will be something like:

```
QUALIFICATION IS MAIN
```

Changing the Current Program Qualification

If the current qualification is MAIN, unqualified statement identifiers and variable names will refer to statements and variables in MAIN. Issuing the command AT /10 will set a breakpoint at the statement labeled 10 in MAIN. If your program has a subroutine (or function) named SUB1 and you wanted to set a breakpoint in that subroutine, you could do so by using an explicit qualifier. For example:

```
AT sub1./10
```

You could also change the current qualification. For example:

```
qualify sub1
```

Now entering:

```
at /10
```

sets a breakpoint at the statement labeled 10 in your subroutine SUB1. To check current breakpoint settings, entering the LISTBRKS command:

```
listbrks
```

would produce output similar to that shown below.

```
CURRENT BREAKPOINTS:
  MAIN.25/10
  SUB1.32/10
CURRENT WHEN CONDITIONS:
  NONE
CURRENT HALT STATUS: OFF
```

Breakpoint settings are displayed for both the main program and the subroutine SUB1.

You can display all the variables in the currently qualified program unit with the LIST command by entering:

```
list *
```

To issue commands without explicit qualification that reference statements or variables in MAIN, change the currently qualified program unit back to the main program by entering:

```
qualify main
```

Now enter LIST * again to display all the variables in MAIN. If, on resuming execution, the breakpoint you set in SUB1 was reached, the currently qualified program unit would be set to SUB1. For example:

```
go
AT SUB1.32/10
list *
```

would list all the variables in SUB1.

Explicit Qualification of Individual Variables

You can qualify individual variables without a QUALIFY command by preceding the variable name with the name of the program unit that it belongs to and a period. For example, you can refer to variable x from MAIN as:

```
main.x
```

You can refer to an array element data(10) from sub2 as:

```
sub2.data(10)
```

When qualifying an array element with a symbolic subscript, remember to also qualify the subscript. For example, to display array element data(I) from sub2 while execution is suspended in MAIN, enter:

```
list sub2.data(sub2.i)
```

If you omit the second SUB2, and the current qualification is MAIN, Interactive Debug will look for a value of I in MAIN. If it finds one, it will give you that element of DATA instead of the one you want in SUB2. You must qualify each variable that is not in the currently qualified program unit.

You can reference variables outside the currently executing program unit in any command dealing with VS FORTRAN variables. For example:

```
list (a,b,sub1.alpha,sub2.beta,x,y,z)
set a=sub1.value
when over (sub1.rchg=5.)
```

Setting Breakpoints at Debugging Hooks

The AT command sets breakpoints at specific statements. Breakpoints can be set only at statements that have *debugging hooks*. A hook gives temporary control to Interactive Debug at a specific point within a program (usually at the beginning of an executable statement).

For VS FORTRAN code compiled with the TEST option, hooks are placed in the object code by the compiler. For VS FORTRAN code compiled with the NOTEST and SDUMP options, hooks can be inserted into the object code by IAD at execution time.

When using the AT command, you can identify the statement either by its statement number or by the statement label, if it has one. Normally, the statement

number refers to the Internal Statement Number (ISN) generated by the compiler (see "Statement Identifier Conventions" on page 114 for more details). However, you can specify at compile time that you want to use the sequence numbers in columns 73 through 80 as the statement numbers for your debugging session instead of the ISNs.

As an example, assume that your main program has a write statement labeled 10 and that the compiler has assigned an ISN of 6 to this statement. The listing might show:

```
ISN     6       10      WRITE(*,*)  'Example Program'
```

You can set a breakpoint at this statement by issuing either of the following two commands:

```
at 6
```

or

```
at /10
```

Statement labels are preceded with a slash to distinguish them from ISNs or sequence numbers. Remember that, if no qualifier is specified, Interactive Debug uses the current qualification to determine which program unit this statement is located in. (MAIN is assumed here.) When the statement is reached, execution is suspended and the following message is displayed:

```
AT: MAIN.6/10
```

You may specify a list of statements or a range of statements with the AT command. For example, the following command:

```
at ( 6 /15 14 3)
```

sets breakpoints at statement numbers 6, 14, and 3, and also at statement label 15, providing they are all executable statements and have debugging hooks.

```
at (6:/15)
```

sets breakpoints at every hooked executable statement between the statement whose ISN is 6 and the statement labeled 15. Both statements specified in a range must be executable, and the statement on the left must appear in the program before the statement on the right.

Breakpoints cannot be set if optimization or vectorization causes the statement to be *collapsed*. A collapsed statement is an executable source statement that occupies no object code because of the effects of optimization or vectorization. (The code was either moved to a new location, or eliminated.) For further explanation, see "Debugging Optimized and Vectorized Code" on page 98.

If the statement is nonexecutable, you cannot set a breakpoint.

If you have specified statement ranges for one or more program units in the AFFON control file, breakpoints can only be set in the specified statement ranges for those program units. Multiple units and ranges can be specified. See Chapters on ISPF, line mode, or batch mode for more information about specifying statement ranges in your environment.

You cannot suspend execution at the "trailer" statement following a logical IF under any of these conditions:

- The program was compiled prior to VS FORTRAN Version 1 Release 3.1.
- The trailer statement is a GOTO statement.
- Sequence numbers were used instead of ISNs.

Controlling Program Execution

Note: The following section refers only to statements that have hooks. You cannot suspend execution at a statement that does not have a hook.

The HALT, NEXT, and WHEN commands provide a number of different ways to control program execution. These commands allow you to suspend execution:

- At every executable statement
- At the next executable statement (without knowing which it is)
- At every apparent program branch
- At every entry to and exit from a program unit
- Whenever a user-defined condition is met
- Whenever a specific variable is modified

The HALT command allows you to suspend execution under certain specified conditions. For example:

```
halt stmt
```

suspends execution at every executable statement. This allows you to single step through your program, which can be helpful in finding errors related to the processing flow.

```
halt goto
```

suspends execution at every apparent program branch. Halting can occur for several reasons, including a GOTO, a DO group, and an IF statement.

```
halt entry
```

suspends execution at every entry to or exit from a debuggable program unit.

The HALT command remains in effect until you cancel it with:

```
halt off
```

The NEXT command requests that execution be suspended at the next executable statement. It is similar to the HALT STMT command, except that the NEXT command is temporary and does not remain in effect after execution is suspended.

The WHEN command allows you to suspend execution every time a particular condition is met. You define the condition and supply its name. Later you can refer to the condition by name without redefining it. With WHEN, you can monitor:

- An arithmetic relationship between two variables or between a variable and a constant
- The status of a logical variable
- A change in the value of a variable

For example, to cause execution to be suspended when variable SMITH equals 30, define a condition, such as the one below named RDS. (The name can be one to four alphameric characters, the first character alphabetic.)

```
when rds (smith = 30)
```

Execution is suspended at the first possible statement following the point at which the condition becomes true. For example, if condition RDS was found to be satisfied at the beginning of statement 46 in program unit MAIN, you would receive the following:

```
WHEN:  "RDS" SATISFIED;
CURRENTLY AT MAIN.46
```

The first line tells you which condition was satisfied; the second tells you where execution is suspended. To detect when SMITH changes value, enter:

```
when rds smith
```

Notice that when you want to define a condition that monitors any change in the value of a variable, the variable name is not enclosed in parentheses.

If the value of SMITH is continually being changed, and SMITH changes initially from 2 to 3, you are notified. If SMITH changes to 4, you are notified again.

Examples:

```
when rds1 (smith = md)

when rds2 (smith .lt. 4.7)

when rds3 (rich)
```

In the final example, RICH is a logical variable. In this case, you get control when RICH is true. If the parentheses were omitted, you would get control whenever RICH was modified.

Note: Interactive Debug cannot tell you exactly which statement changed the variable being monitored. (It might have occurred in a section of code that has no debugging hooks.) However, you can get a list of the last ten branches known to the debugger by entering `WHERE FLOW`. This should help you deduce which statement actually caused the change.

To turn off WHEN condition monitoring, use the OFFWN command. For example, to turn off condition RDS, enter:

```
offwn rds
```

To turn off condition DJV along with condition RDS, enter:

```
offwn (djv,rds)
```

To stop all condition monitoring, enter:

```
offwn *
```

WHEN condition monitoring is not automatically turned off when a condition is satisfied. If you do not want Interactive Debug to continue monitoring the same condition, you must issue the OFFWN command after the condition has been satisfied. If you want to reactivate a condition after it has been deactivated by an OFFWN command, enter WHEN with the condition name. For example:

```
when rds
```

Using Command Lists

As part of an AT command, you can specify a list of commands to be executed whenever a breakpoint is reached. This allows you to conditionally suspend execution at a specific statement or to specify a list of commands to be executed there. When you specify a command list, you can control whether Interactive Debug will wait for a command, or whether it will continue execution without the need for intervention.

In this example, the value of variable A will be displayed each time sequence number 10 is reached, and execution will then continue.

```
at 10 (list a %go)
```

This is useful for observing how the value of a variable changes in a loop. Here, sequence number 10 could be at the end of a loop and the value of variable A would be displayed at each iteration of the loop.

Note: The percent sign (%) is used to separate commands within the list.

To conditionally suspend execution at a particular statement, you can use the IF and HALT commands within an AT command list. The HALT command will suspend the execution of the command list. For example, entering:

```
at 10 (list (a,b) %if (a.lt.b) halt %go)
```

will cause execution to be suspended at sequence number 10 only if A is less than B. Otherwise, the GO command will cause execution to continue. In either case, the values of A and B will be displayed.

A command in an AT command list that causes execution to resume or halt will cause the remainder of the command list to be ignored. In this example:

```
at /200 (if (a=0) go /10 %if (b=0) go /10 %go /300)
```

if the value of A or B is equal to 0, execution will resume at the statement labeled 10; otherwise, execution will resume at the statement labeled 300.

The following commands cannot be used in a command list:

COLOR	POSITION
FIXUP	PREVDISP
HELP	PROFILE
LISTINGS	SEARCH
MOVECURS	WINDOW

Displaying the Data Types of Scalar Variables and Arrays

To display the data types of scalar variables and arrays, use the DESCRIBE command. DESCRIBE also displays dimension information for arrays. This command can be useful for checking the attributes of variables and arrays when it is inconvenient to search the source listing for their declarations. It is particularly useful for displaying the dimensions that were passed for dummy array arguments.

For example, to see the data type of the variable "a" in program unit "sub1," enter this command:

```
describe sub1.a
```

By entering an asterisk (*), you can request a display of the type of every variable in the currently qualified program unit. To see a list of all the names in the current program unit, with their data types, enter this:

```
describe *
```

Let's say you have a program that uses a mixture of scalars and arrays, and you would like to display the type of a specific group of them. You can specify the scalars and arrays in a name list, for example:

```
describe (i,k,sub2.dumchr,r8ary,sub2.r4dumy,l1aymn)
```

The output might look like this:

```
SUB3.I:                                INTEGER*4
SUB3.K:                                INTEGER*4              DUMMY
SUB2.DUMCHR:                           CHARACTER*(*)          DUMMY
SUB3.R8ARY:                            REAL*8
   RANK = 2,  SIZE = 49 ELEMENTS
   DIM 1:     EXTENT = 7, LBOUND = (1), UBOUND = (7)
   DIM 2:     EXTENT = 7, LBOUND = (1), UBOUND = (7)
SUB2.R4DUMY:                           REAL*4                 DUMMY
   RANK = 3;  DUMMY ARRAY ARGUMENT OF INACTIVE SUBPROGRAM OR
              ALTERNATE ENTRY POINT;
              DIMENSION INFORMATION NOT AVAILABLE
SUB3.L1AYMN:                            LOGICAL*1             DUMMY
   RANK = 2,  SIZE = *  ELEMENTS
   DIM 1:     EXTENT = 5, LBOUND =(-2), UBOUND = (2)
   DIM 2:     EXTENT = *, LBOUND = (1), UBOUND = (*)
```

Dummy arguments are identified by DUMMY in the right-hand column. The length of DUMCHR could not be determined because it is a dummy argument in an inactive program unit, so the length is displayed as CHARACTER*(*).

Dimension information is not available for R4DUMY. This information is never displayed for arrays that are dummy arguments in an inactive program unit, or that are defined only when entered by some other entry point.

L1AYMN is an assumed-size array. The upper bound for the last dimension of such arrays is displayed as an asterisk (*), and the size is indicated as follows:

```
* ELEMENTS
```

Determining Statement Execution Frequency

The LISTFREQ command displays the number of times each statement has been executed. For example, to see how often the statement whose statement number is 100 has been executed, enter:

```
listfreq 100
```

You can specify a statement label or a statement number (an ISN or sequence number), a list of statement labels or statement numbers (in parentheses), or a range of statements. For example, to see how often each statement between sequence numbers 45 and 52 has been executed, enter:

```
listfreq 45:52
```

To list the number of times every statement in the currently qualified program unit has been executed, enter:

```
listfreq
```

This produces output similar to the following:

```
STATEMENT                                FREQUENCY
MAIN.ENTRY                                NO HOOK
MAIN.EXIT                                 NO HOOK
MAIN.14/80                                6
MAIN.15                                   6
MAIN.16                                   90
MAIN.17                                   90
MAIN.18                                   12
MAIN.19                                   90
MAIN.20/50                                90
MAIN.21                                   6
MAIN.22                                   5
```

The LISTFREQ output tells you which statements have debugging hooks. Those that do will have an execution count in the FREQUENCY column. Those that do not will be identified as either COLLAPSED STMT or NO HOOK. COLLAPSED STMT indicates that, because of optimization or vectorization, there is no code at this location. NO HOOK indicates that the statement was not in the AFFON statement range list, or that it is an ENTRY/EXIT to the main program unit.

You can also use LISTFREQ to list statements that have never been executed. The ZEROFREQ keyword is used to create such a list. The PRINT keyword can be added to obtain a listing file or a print data set.

```
listfreq 11:74 zerofreq print
```

Interactive Debug will list to the print file all the statements between statement numbers 11 and 74 that have never been executed.

Program Sampling

Program sampling can help you identify the portions of your program that are using the most CPU time, without using the resources required when using debugging hooks. The information developed by program sampling can be displayed at your terminal using the LISTSAMP command, or reported as printed output using the ANNOTATE command.

Initiating Program Sampling

You can initiate program sampling by issuing the ENDDEBUG command with the SAMPLE option. (See "ENDDEBUG Command" on page 138.) This will cause interactive debug to interrupt your program's execution periodically to collect sampling data. Data is collected for each statement of every debuggable program unit and for each entry point in every nondebuggable program unit. The sampling data is recorded in two counters for each of the statements and entry points, as follows:

- DIRECT counter

  If the interruption occurs in the code of a debuggable program unit, the DIRECT counter for the interrupted statement is incremented. If the interruption occurs while the VS FORTRAN library is active, the DIRECT counter for the library entry point is also incremented.

- CALLED counter

  If sampling is initiated with the CALLED option of ENDDEBUG, the CALLED counter is incremented for each statement (or entry) included in the sequence of calling program units that lead to the interrupted statement. This is done by tracing the register save area chain back to the main program. In addition, if an I/O operation is in process when an interruption occurs, the CALLED counter is incremented for the statement (or entry) that requested the operation.

All interruptions that cannot be associated with any statement or entry are recorded in the *UNKNOWN DIRECT counter.[1] This count is incremented if an interruption occurs in a program that has no entry identifier or in system code servicing an asynchronous interrupt. The *UNKNOWN CALLED counter is incremented when the save area chain cannot be successfully traced back to the main program.

An additional counter, *LIBRARY DIRECT, shows the sampling count for all VS FORTRAN library modules, other than the mathematical functions and the Error

---

[1] In order to be properly identified, nondebuggable modules must follow standard MVS linkage conventions.

Monitor. This includes lower-level calls to system services. The *LIBRARY CALLED counter is never incremented.

Displaying Program Sampling Statistics

When your program has completed, you can report the program sampling statistics in three ways:

1. **Terminal display**: Using LISTSAMP, you can show information either for selected statement ranges or summarized by program unit. Additionally, LISTSAMP allows you to list only those statements or program units having the highest sampling counts, using the TOP(n) option.

2. **Printout**: Using ANNOTATE, you can copy the VS FORTRAN source listings to AFFPRINT. Sampling data is added to the right of each statement and is summarized by program unit. All program units and nondebuggable entries that were encountered are included in the summary. Page number references are shown for program units whose listings were annotated.

3. **Bar chart**: Under ISPF Version 2, you can have the source listing window overlaid with a bar chart that displays the frequency or sampling data for each statement in the listing. You control this feature by your choice of options for the ANNOTATE command, specifying whether you want the bar chart to be in terms of "frequency" (total executions) or "sample" (timer interrupts).

   The bar chart adjusts to the size of the window so that 100% would cover the full width. On a seven color terminal, the bar charts are shown by simply changing the color (reverse video is assumed). Otherwise, asterisks (*) are displayed.

The ANNOTATE and LISTSAMP commands provide options to let you display the DIRECT counts, the CALLED counts, or the sum of the two (ALL) in your terminal or printed output.

The following examples illustrate some forms of the ANNOTATE and LISTSAMP commands.

Example 1

Copy the source listing for SUB2 to AFFPRINT, annotating it with sampling information.

```
annotate sub2
```

Example 2

Overlay the source listing window with a bar chart showing the sum of the DIRECT and CALLED counters.

```
annotate on all
```

Example 3

Display a summary of the sampling counts for all program units

```
listsamp * summary

PROGRAM SAMPLING INTERVAL WAS 10 MS; TOTAL NUMBER OF SAMPLES
        WAS 2698.
DIRECT SAMPLES:
 PROGRAM UNIT     SAMPLES %TOTAL
 MAIN                  90   3.34 *
 INIT                   0   0.00
 FUN1                 177   6.56 *
 SUB1                 752  27.87 ******
 S#QRT                 61   2.26
 S#IN                   4   0.15
 A#LOG                  5   0.19
 *LIBRARY            1609  59.64 ************
```

(MAIN, INIT, FUN1, and SUB1 are VS FORTRAN program units; S#QRT, S#IN, and A#LOG are VS FORTRAN math library entry points; *LIBRARY contains the counts of sampling occurrences in VS FORTRAN non-math library routines.)

Example 4

Display the sampling counts for SUB1, including the CALLED counts (Sampling counts will include interruptions which occurred in the code of a statement as well as in any lower-level routines called by the statement.)

```
listsamp sub1.* all

PROGRAM SAMPLING INTERVAL WAS 10 MS; TOTAL NUMBER OF SAMPLES
        WAS 2698.
SUM OF DIRECT AND CALLED SAMPLES:
STATEMENT            SAMPLES     %UNIT    %TOTAL
 SUB1.ENTRY/EXIT          52      6.47      1.93
 SUB1.8                    7      0.87      0.26
 SUB1.9                   34      4.23      1.26
 SUB1.10                 561     69.78     20.79 ****
 SUB1.11/10              146     18.16      5.41 *
 SUB1.12                   4      0.50      0.15
 SUB1.13                   0      0.00      0.00
```

Limitations Using Program Sampling

There are some limitations that you should keep in mind with regard to program sampling:

1. Use of sampling will cancel any active timer interval at the start of execution.
2. In CMS, the BLIP will be turned off during sampling.
3. Program performance will be slightly degraded due to the execution of sampling code at each interrupt.[2] Use of the CALLED option executes additional code in order to trace back through the call chain.
4. Code generated by VS FORTRAN currently does not follow MVS standard linkage conventions. Specifically, it loads register 13 with the address of the new SAVE area before chaining the new SAVE area to the old SAVE area. In rare situations, this can cause errors in the CALLED counts.
5. Interruptions that occur in math library routines called by program units compiled with NOSDUMP will be attributed to the program unit instead of the library routine. You can eliminate this inaccuracy by recompiling with the SDUMP option.
6. The accuracy of the STIMER macro, used to provide periodic interruptions on VM and MVS, is sensitive to system activity. Thus, sampling may occur less often than the interval you specified in ENDDEBUG. If your CPU has the virtual interval timer assist facility, and you are using CMS, you may be able to improve the timing values. You can turn on this facility by issuing:

   ```
   SYSCMD CP SET ASSIST ON TMR
   ```

7. If you use the STIMER macro in your program, sampling will be discontinued.

Displaying Timing Information

If you want to time one or more program units and then see the timing information produced by Interactive Debug, use the TIMER and LISTTIME commands.

The TIMER command turns timing on and off, or resets the activation count and time to zero for a specified program unit. When you want to turn timing on for a program unit named LOOP, for example, you must issue the command:

```
TIMER LOOP
```

After your program has executed, you can issue the LISTTIME command to see both the cumulative values and the percent of total execution time for each timed program unit and for each entry point. (The timing information is presented by entry point, although timing is controlled by the program unit name.) LISTTIME shows values only for those program units for which timing has been turned on

---

[2] On the order of one millisecond per timer interruption.

(using TIMER). To get a printed listing of the LISTTIME information, enter the command:

```
LISTTIME PRINT
```

To see the information at your terminal, omit the PRINT keyword.

If you want to increase the accuracy of timing information for subroutines, you must minimize the overhead caused by debugging hooks in your program. To do this, use the AFFON selection file to specify hooks only on entry and exit points. Your entry in the AFFON file might look like this for the program unit LOOP:

```
LOOP ENTRY
```

Next use the TIMER command to turn timing on for the subroutine you are interested in. If the subroutine calls any other routines, be sure to turn timing on for them also. If you do not, the time spent in any called routines is included in the measurement for the calling routine.

When you ask for your LISTTIME display, you should see a fairly accurate execution time for that subroutine.

Tracing Program Execution

Two commands provide trace information: TRACE and WHERE.

The TRACE command traces control transfers within your program as it executes. To trace each entry to and exit from any subprogram as it occurs, enter:

```
trace entry
```

This produces output similar to the following:

```
TRACE:  FROM MAIN.14 TO SUB1.ENTRY
```

To trace the origin and destination of every apparent branch within the program (including entry to and exit from subprograms) listed by statement identifier, enter:

```
trace goto
```

This produces output similar to the following:

```
TRACE:  FROM SUB1.150/20 TO SUB1.210/40
```

If you don't need to examine your TRACE output right away, you can add PRINT to the TRACE command and send the output to the print data set.

```
trace goto print
```

To stop tracing, enter:

```
trace off
```

The WHERE command shows you the number of the statement at which execution is suspended. This statement will normally be the one executed next. For example,

if MAIN calls subroutine TAD at sequence number 150, but a breakpoint is set at sequence number 20 in TAD, a WHERE command produces:

```
WHERE:  TAD.20
```

WHERE has a TRBACK keyword that gives you a trace of the calls that got you to your current location. For example, if your program MAIN calls subroutine TAD and execution is suspended, entering:

```
where trback
```

might produce the following:

```
WHERE:  TAD.20
TAD CALLED AT MAIN.150
```

TRBACK output is limited to the transfers between debuggable program units.

WHERE also has a FLOW keyword that gives you a trace of the last ten program transfers executed. For example, if you specify the FLOW keyword:

```
where flow
```

you will receive output similar to that in Figure 24.

```
WHERE: MAIN.92                    (WHERE response)
TO: MAIN.80     FROM: MAIN.85     (FLOW response, prev. branch)
TO: MAIN.65     FROM: MAIN.70     (Next most recent branch)
TO: MAIN.51     FROM: MAIN.53
TO: MAIN.49     FROM: MAIN.53     (Loop in MAIN)
TO: MAIN.49     FROM: MAIN.53
TO: MAIN.47     FROM: MAIN.40
TO: MAIN.38     FROM: MAIN.20
TO: MAIN.15     FROM: MAIN.10
TO: MAIN.3      FROM: MAIN.10     (Loop in MAIN)
TO: MAIN.3      FROM: MAIN.10
```

Figure 24. Example of WHERE FLOW Output

Note that Interactive Debug can only keep track of statements that have debugging hooks. If there is a block of code that has no hooks, it appears to Interactive Debug as if there was a branch from the statement before the block to the statement after it.

The PRINT keyword can be used to send WHERE information to the print data set. For example, you can record the contents of a 100-element array, "ar," at several points in your program by issuing the following sequence of commands each time you want "ar" recorded:

```
where print
list ar(1):ar(100) print
```

This produces a record on your print data set both of the array "ar" and of the exact program location where "ar" had that particular content.

Displaying Formatted Variable and Array Values

To display values, use the LIST command. To display the value of variable A, enter:

```
list a
```

To display the values of variables A, B, C, D, enter:

```
list (a,b,c,d)
```

Similarly, to display array elements ARY(1,1) through ARY(3,4), enter:

```
list ary(1,1):ary(3,4)
```

Output from the LIST command can be very long and you may not want it displayed at the terminal. The LIST command has a PRINT keyword that sends its output to a print data set.

```
list a(1):a(100) print
```

The value of each variable in a LIST command is normally displayed in its correct VS FORTRAN data type and precision: integer, real, complex, and so on. If you want to display the values in a different format, you can use the FORMAT or DUMP keyword. To display the hexadecimal values of some variables, enter:

```
list (a,i,n,r) format(x)
```

To display the values as if they were character strings, enter:

```
list (a,i,n,r) format(a)
```

The DUMP keyword is similar to the FORMAT keyword, but shows the hexadecimal storage location instead of the name. For example, to display the storage location and the hexadecimal value of variable A, enter:

```
list (a) dump(x)
```

The FORMAT and DUMP keywords and codes are described in the reference section in Figure 26 on page 127.

Note: For the LIST command and other commands that allow array element references, the subscripts must use simple arithmetic expressions no more complex than the form "variable plus (or minus) a constant." For example,

```
A(I), A(3), ARY(I+3) or ARY(I-3)
```

are valid.

Handling Execution-Time Errors

VS FORTRAN Version 2 Interactive Debug allows you to control the action taken when execution errors are encountered. The ERROR command allows you to specify whether to take corrective action or to suspend execution when an error occurs. If you choose the latter, corrective action can be specified by the FIXUP command. It also allows you to suppress the VS FORTRAN library execution-time error messages for specific errors.

Initially, whenever an execution-time error occurs, execution is suspended and VS FORTRAN library execution-time error messages are displayed. The ERROR command allows you to change these initial error settings. There are two pairs of keywords for the ERROR command:

MSG/NOMSG — specifies whether or not the VS FORTRAN library execution-time messages are to be displayed.

EXIT/NOEXIT — specifies whether execution is to be suspended or corrective action is to be performed. If you specify that corrective action is to be taken (NOEXIT), standard corrective action will normally be taken. However, if you have specified your own corrective action by calling ERRSET in your program, whatever action you specified will be taken.

Note: Calling ERRSET in your program will cause the NOEXIT and MSG settings to be in effect for the specified errors. You can issue the ERROR command afterward to change this if you want.

The MSG and EXIT keywords are the defaults.

Identifying Errors

The ERROR command uses the identification numbers from the VS FORTRAN library to identify execution-time errors. (You can find these error numbers in *VS FORTRAN Version 2: Language and Library Reference.*)

The following examples illustrate the use of the ERROR command.

```
error 215 noexit msg
```

causes corrective action to be taken and a full diagnostic message to be displayed for error AFB215I.

```
error 215 noexit
```

has the same effect as the previous command, because MSG is the default.

```
error 215 noexit nomsg
```

causes corrective action to be taken and will suppresses the diagnostic message.

After entering this command you will not be notified if error AFB215I occurs again. You can change the error settings back to their original settings by entering any one of the following commands:

```
error 215
error 215 exit
error 215 msg
error 215 msg exit
```

All these commands have the same effect because of keyword defaults.

You may also specify a list of errors or a range of errors. For example:

```
error (215 243 247 289) noexit
```

will cause corrective action to be taken, and full diagnostic messages to be displayed for errors AFB215I, AFB243I, AFB247I, and AFB289I.

```
error (215:218 290) nomsg
```

will cause execution to be suspended and suppress diagnostic messages for errors AFB215I, AFB216I, AFB217I, AFB218I, and AFB290I.

Note: When executing under VS FORTRAN Version 2 Interactive Debug (except in batch mode), the VS FORTRAN library does not update the execution-time error occurrence counts; therefore, settings in the VS FORTRAN Version 2 error option table that depend on these counts have no effect. This permits unlimited occurrences of errors and messages regardless of the settings in the error option table.

In batch mode, the error counts are updated and tested just as if running without Interactive Debug.

Performing Corrective Action

When execution is suspended because of an error, Interactive Debug displays a message like this:

```
ERROR EXIT: ERROR 243 AT MAIN.6/15
```

and waits for you to enter a command. You may not enter a GO command with a statement identifier. Entering the FIXUP command with no arguments, or a GO command with no statement identification, will cause standard corrective action to be taken and execution to resume. If the error is caused by an incorrect value passed to a VS FORTRAN library mathematical routine, you may use the FIXUP command to specify corrected values to be used to recalculate the function.

With FIXUP, you can assign values to the first, second, or both arguments of a function. The following examples illustrate some of the uses of the FIXUP command:

```
ERRMSG=> AFB241I FIXPI : INTEGER BASE=0, INTEGER EXPONENT=0, LESS THAN
ERRMSG=> OR EQUAL TO ZERO
ERRMSG=>          FIXPI : LAST EXECUTED FORTRAN STATEMENT IN PROGRAM
ERRMSG=> MAIN AT ISN 44 (OFFSET 000954).
INFMSG=>  USER ERROR CORRECTIVE ROUTINE ENTERED.
ERROR EXIT: ERROR 241 AT MAIN.44
IAD/E
fixup arg1(2) arg2(2)
INFMSG=> USER CORRECTIVE ACTION TAKEN. EXECUTION CONTINUING.
```

The function has been reevaluated using both arguments, and execution continues.

```
ERRMSG=> AFB242I FRXPI : REAL*4 BASE=0.0, INTEGER EXPONENT = 0, LESS
ERRMSG=> THAN OR EQUAL TO ZERO
ERRMSG=>          FRXPI : LAST EXECUTED FORTRAN STATEMENT IN PROGRAM
ERRMSG=> MAIN AT ISN 51 (OFFSET 00099E).
INFMSG=>  USER ERROR CORRECTIVE ROUTINE ENTERED.
ERROR EXIT: ERROR 242 AT MAIN.51
IAD/E
fixup arg1(2.0)
INFMSG=> USER CORRECTIVE ACTION TAKEN. EXECUTION CONTINUING.
```

A new value is given for the first argument (base) only; the second argument (exponent) remains unchanged.

```
ERRMSG=> AFB244I FRXPR : REAL*4 BASE=0.0, REAL*4 EXPONENT= 0.0000000E
ERRMSG=> +00,LESS THAN OR EQUAL TO ZERO
ERRMSG=>          FRXPR : LAST EXECUTED FORTRAN STATEMENT IN PROGRAM MAIN
ERRMSG=> AT ISN 61 (OFFSET 000A0E).
INFMSG=> USER ERROR CORRECTIVE ROUTINE ENTERED.
ERROR EXIT: ERROR 244 AT MAIN.61
IAD/E
fixup arg2(1.0e)
INFMSG=> USER CORRECTIVE ACTION TAKEN. EXECUTION CONTINUING.
```

The function has been reevaluated by changing the second argument; the first argument is unchanged.

```
ERRMSG=> AFB243I FDXPI : REAL*8 BASE=0.0, INTEGER EXPONENT = 0, LESS
ERRMSG=> THAN OR EQUAL TO ZERO
ERRMSG=>         FDXPI : LAST EXECUTED FORTRAN STATEMENT IN PROGRAM
ERRMSG=> MAIN AT ISN 56 (OFFSET 0009D6).
INFMSG=> USER ERROR CORRECTIVE ROUTINE ENTERED.
ERROR EXIT: ERROR 243 AT MAIN.56
IAD/E
f
INFMSG=> STANDARD CORRECTIVE ACTION TAKEN. EXECUTION CONTINUING.
```

The abbreviation F for FIXUP is used with no arguments. Standard corrective action is taken. The same action would have been taken if GO had been entered instead.

To determine what standard corrective action is applied for a particular error or for more information about mathematical functions and their arguments, see *VS FORTRAN Version 2: Language and Library Reference.*

Processing External Files

VS FORTRAN Version 2 Interactive Debug allows you to manipulate external files used by a VS FORTRAN program through a set of commands that are similar to corresponding VS FORTRAN statements.

The BACKSPACE command positions a sequentially accessed external file at the beginning of the previous record. For example:

```
backspace 8
```

positions the file connected to I/O unit 8 at the beginning of the last record written or read, allowing it to be written or read again.

The REWIND command positions a sequentially accessed external file at the beginning of the first record of the file. For example:

```
rewind 4
```

positions the file connected to I/O unit 4 at the beginning. This permits you to perform I/O operations as though the file had just been opened. VS FORTRAN supports multiple files under the same I/O unit. The REWIND command sets the VS FORTRAN file name to the first in the sequence of files for the specified I/O unit. For example, if you were currently processing file FT08F003 on I/O unit 8 and entered:

```
rewind 8
```

I/O unit 8 would be connected to file FT08F001, which would be positioned at the beginning of the first record.

The ENDFILE command writes an end-of-file record on a sequentially accessed external file. This causes subsequent I/O operations to be performed on the next

file for the specified I/O unit. For example, if you are currently processing file FT05F001 on I/O unit 5, entering:

```
endfile 5
```

causes subsequent I/O operations to be performed, using file FT05F002. If REWIND were issued for I/O unit 5, the filename would be set back to FT05F001.

The CLOSE command disconnects an external file from an I/O unit. For example:

```
close 1
```

disconnects the VS FORTRAN file connected to I/O unit 1, allowing you to associate a different CMS file or TSO data set with that I/O unit, if desired, or to examine the file contents using an editor or browser.

The RECONNECT command resets a file to its original (preconnected) condition. For example, if unit 8 has been closed, you can make it possible for the program to perform additional I/O on unit 8 (without executing an OPEN) by issuing

```
reconnect 8
```

to reconnect unit 8 to file FT08F001. This is necessary only if the OCSTATUS run-time option is in effect.

If you neglected to allocate a file that is needed by your program, you will receive an error message when the program attempts to access that file. If the program is debuggable and ERROR EXIT is in effect, you can recover from this condition by issuing the following sequence:

```
NEXT
GO
SYSCMD ALLOCATE...(or FILEDEF...)
GO n
```

where n is the statement identifier for the I/O statement. The ALLOCATE or FILEDEF command must be completed as appropriate for allocating the required file. This procedure will work whether or not the OCSTATUS run-time option is in effect.

Using System Commands

Using SYSCMD, you can issue a system command while debugging. For example, if you wanted to begin processing a different CMS file or TSO data set using I/O unit 8, you could enter the following:

CMS Example

```
close 8
syscmd filedef 8 disk example data a
```

TSO Example

```
close 8
syscmd allocate file (ft08f001) da(example.data) reuse
```

SYSCMD has many uses; the ability to view your VS FORTRAN source or listing files is particularly useful in the line mode environment. For example:

CMS Example

```
syscmd xedit example fortran a
syscmd type example listing a
```

TSO Example

```
syscmd edit example.fort
syscmd list example.list
```

Another possible use is to show which CMS files are currently defined or which TSO data sets are currently allocated. For example:

CMS Example

```
syscmd filedef
```

TSO Example

```
syscmd listalc status
```

Entering Terminal Input

When a VS FORTRAN program attempts to perform any I/O to or from the terminal, either the VS FORTRAN Version 2 Interactive Debug I/O routines or the VS FORTRAN library I/O routines can be used. The TERMIO setting determines which set of routines is used. Using the Interactive Debug routines gives you the advantage of being told which unit is being read, and you have the ability to issue Interactive Debug commands while the read is pending. In addition, when operating in full screen mode, the Interactive Debug routines remain in full screen mode for the read or write.

When executing under Interactive Debug, the initial TERMIO setting is IAD, specifying that the Interactive Debug routines should be used for terminal I/O. The TERMIO command allows you to change the setting to either IAD or LIBRARY or to display the current TERMIO setting. When the option selected is LIBRARY, all terminal I/O is performed in line mode in the same manner as when Interactive Debug is not active.

When executing in batch mode on MVS, no actual terminal is available. However, the DEBUNIT execution option may be used to specify that certain I/O units are to be treated as if they were allocated to the terminal. These units then come under the control of TERMIO. For details about using DEBUNIT, see *VS FORTRAN Version 2: Programming Guide*.

When the TERMIO setting is IAD and your program attempts to read from the terminal, the following message is displayed:

```
FT05F001 INPUT: PRECEDE INPUT WITH % OR ENTER IAD COMMAND
```

where FT05F001 indicates the VS FORTRAN file being read. When this message is displayed, you may either:

- Issue a debugging command (other than STEP, GO, or ENDDEBUG).
- Enter the requested input, prefaced with a percent sign (%).

The percent sign is not passed to the VS FORTRAN program. For example:

```
FT05F001 INPUT: PRECEDE INPUT WITH % OR ENTER IAD COMMAND
%743
```

inputs the number 743 to your program. Alternatively, you could have first determined which statement in your program was issuing the read, as follows:

```
FT05F001 INPUT: PRECEDE INPUT WITH % OR ENTER IAD COMMAND
where
WHERE: MAIN.10
FT05F001 INPUT: PRECEDE INPUT WITH % OR ENTER IAD COMMAND
%743
```

When the TERMIO setting is IAD:

- The leading and trailing percent signs are removed before the input is sent to the VS FORTRAN program (the trailing percent sign is not required). Whenever you want to include a percent sign as part of your input, you must include a trailing percent sign. For example, to input the string 55% to a VS FORTRAN program, enter:

  ```
  %55%%
  ```

  To enter two percent signs (%%), enter:

  ```
  %%%%
  ```

- To signify an end-of-file from the terminal, use two percent signs. For example:

  ```
  %%
  ```

- All terminal input is padded on the right with blanks to the LRECL of the terminal file.
- All terminal input is converted to uppercase. To avoid this in environments that support mixed-case input, use the TERMIO LIBRARY setting.
- Both VS FORTRAN Version 2 Interactive Debug commands and terminal input can be continued on succeeding lines by ending each continued line with a hyphen (-). For example:

  ```
  FT05F001 INPUT: PRECEDE INPUT WITH % OR ENTER IAD COMMAND
  % hello, this input line is -
  broken across two lines.
  ```

- Under ISPF, leading blanks are stripped from the beginning of all continuation lines. To avoid this, precede the continuation line with a single quotation mark ("). The quotation mark will not be passed to the program. For example:

  ```
  FT05F001 INPUT: PRECEDE INPUT WITH % OR ENTER IAD COMMAND
  % hello, this input line is -
  "  broken across two lines.
  ```

- In CMS line mode, the maximum length of an input line is 131 characters. Any input longer than this is truncated.
- In TSO line mode, the maximum length of an input line is 251 characters. Any input longer than this is truncated.
- In ISPF, the maximum length of an input line is 251 characters. When this limit is exceeded, a message is displayed and the input is ignored.
- All output written from your program to the terminal is shown in the log, preceded by the name of the VS FORTRAN file it was written to, and is broken into lines that are 60 characters long.

Continuing Execution without Further Debugging

When you complete debugging and want to finish execution of the program, use the ENDDEBUG command. This command discontinues communication between the program and VS FORTRAN Version 2 Interactive Debug until the program terminates. Only use ENDDEBUG when you are certain that further debugging is not required.

Using ENDDEBUG causes the program to execute as if the DEBUG execution-time option had never been specified, except that attention interrupts are possible, and Interactive Debug will be reentered at termination.

After ENDDEBUG has been issued, all terminal I/O is handled by the VS FORTRAN library I/O routines (as if TERMIO LIBRARY had been issued).

In addition, the VS FORTRAN library begins updating the occurrence count for execution-time errors. All error handling is determined by settings in the VS FORTRAN Version 2 error option table. The error summary displayed when the program terminates reflects only errors occurring after ENDDEBUG was issued. WHERE information is not available.

When the program terminates after issuing an ENDDEBUG command, you may issue Interactive Debug commands. However, commands such as LISTFREQ and LISTTIME will display only the information that was current when ENDDEBUG was issued.

Attention can still be used to interrupt the program after ENDDEBUG is issued; however, no program information is available at that time. To resume execution after an attention interrupt, enter a null line. The only command allowed while in an attention exit after ENDDEBUG has been issued is QUIT, which terminates the program execution. You can then list the values of variables or issue other commands that do not require program execution.

Chapter 7. Special Considerations When Using Interactive Debug

This section describes various situations you may encounter while using Interactive Debug, and gives you more detail than the previous chapters. It presents methods of completing various tasks or handling common problems. The topics discussed here are:

- Issuing commands after program termination
- Handling loops in nondebuggable program units
- Specifying the default execution-time options
- Monitoring floating-point equalities
- Referring to unused FORTRAN variables
- Entering commands in an attention-interrupt exit
- Debugging optimized and vectorized code
- Improving performance when using Interactive Debug
- Recognizing some common errors when setting up a debugging session

For conventions when using any Interactive Debug command with statement labels or statement numbers, see "Statement Identifier Conventions" on page 114.

Issuing Commands after Termination of a VS FORTRAN Program

You can issue the following Interactive Debug commands after your program finishes execution:

ANNOTATE	LISTFREQ	QUIT
AUTOLIST	LISTINGS	REFRESH
BACKSPACE	LISTSAMP	RESTART
CLOSE	LISTSUBS	REWIND
COLOR	LISTTIME	SEARCH
comment	MOVECURS	SET
DESCRIBE	POSITION	SYSCMD
ENDFILE	PREVDISP	TERMIO
HELP	PROFILE	WHERE
LIST	PURGE	WINDOW
LISTBRKS	QUALIFY	

The following commands cannot be entered after program termination:

AT	HALT	RECONNECT
ENDDEBUG	IF	STEP
ERROR	NEXT	TIMER
FIXUP	OFF	TRACE
GO	OFFWN	WHEN

Handling Loops in Nondebuggable Program Units

Many programs may contain units that cannot be debugged. These may be subroutines coded in some language other than VS FORTRAN Version 1 or VS FORTRAN Version 2, a VS FORTRAN module that is not debuggable, and so forth. It is possible for execution to get caught in a loop in such a nondebuggable program unit. When this happens, the only way to suspend the program is by issuing an attention interrupt, and the only accepted command that will affect program execution is QUIT. When QUIT is entered following an attention interrupt, VS FORTRAN Version 2 Interactive Debug terminates the program and allows you to list the values of variables or issue other Interactive Debug commands that do not require further execution. When QUIT is issued again, the debugging session is terminated.

Specifying Default Execution-Time Options

You may occasionally need to use a program compiled with a compiler other than VS FORTRAN (COBOL, for example), which then invokes a VS FORTRAN program. However, when you have no FORTRAN main program to accept a parameter, there is no way to explicitly specify DEBUG or any other execution-time option. One way to solve this problem is to override the normal VS FORTRAN default options by including a local parameter specification module (AFBVLPRM) when you link-edit your module, or by using a customized library containing a global parameter specification module (AFBVGPRM). You can

create either a local or a global parameter specification program by assembling a small program that invokes a macro supplied with the VS FORTRAN Version 2 library. For details, see *VS FORTRAN Version 2: Programming Guide.*

Monitoring Floating-Point Equalities

Requesting Interactive Debug to monitor equal conditions for floating-point numbers requires caution. Equality comparisons are performed on a bit-by-bit basis. Numbers may appear to be equal in a program designed to be insensitive to minor differences, but may differ by a single bit and not be considered equal when compared by Interactive Debug. Because only simple relations are supported, there is no way to monitor equality to a given precision. See "WHEN Command" on page 213 for more information.

Referring to Unused VS FORTRAN Variables

If you define a variable in your VS FORTRAN program without assigning an initial value, and never refer to it in your program, no storage is allocated for the variable and it does not appear in the symbol table used by Interactive Debug. If you try to refer to the variable in your debug session, you will get an error message stating that the variable does not exist.

Therefore, if there are VS FORTRAN variables that you may want to reference in Interactive Debug but which are not referenced in the VS FORTRAN program, you should assign initial values to them.

Entering Commands in an Attention-Interrupt Exit

You can interrupt processing in full screen or line mode by issuing an attention interrupt signal. You can use this signal to gain control if your program appears to be looping, or if an Interactive Debug command is producing excessive output.

The way you issue an attention interrupt signal varies with the operating system and with the type of terminal. A line mode terminal typically has a BREAK key, or one marked ATTN. On a 3270-type terminal, you can use the PA1 key. Under VM, pressing ENTER may signal attention.

When you enter an attention interrupt, Interactive Debug issues the *attention prompt* (`IAD/A`), and temporarily suspends your program. Your program is now in an *attention exit*. You can either:

- Enter a null line, which will cause execution to continue. (You will leave the attention exit.)
- Enter an Interactive Debug command.
- Enter QUIT, which will terminate your program. (However, you will still have a chance to issue any Interactive Debug command that is valid after program termination.)

If you issue any of the following commands while in an attention exit, the command will be executed and you will remain in the attention exit:

PURGE	To terminate excessive output, such as from a LIST command.
WHERE	To identify the last statement that was begun in a debuggable program unit (parameters are not honored).
NEXT	To request a pause at the next executable statement in a debuggable program unit. However, if your program is looping in a nondebuggable program unit, you may never get back to a debuggable unit.
*** or " (comment)**	To enter a comment. (However, the comment will not be logged.)

If you issue a QUIT command while the program is executing, the program will be terminated and Interactive Debug will accept debugging commands that are not related to execution.

If you issue any other non-fullscreen Interactive Debug command while the program is executing, the command will be saved, the attention routine will be exited, and execution will continue. The saved command is deferred until an executable statement in a debuggable program unit is reached (which may not happen). Note that if the program is looping in a nondebuggable routine, the saved command will never be executed. Under ISPF Version 2, the full screen commands cannot be issued from an attention exit.

Debugging Optimized and Vectorized Code

VS FORTRAN Version 2 Interactive Debug allows you to debug VS FORTRAN programs compiled at any optimization level available in the VS FORTRAN compilers (0, 1, 2, or 3), or at any vectorization level (0, 1, or 2). Vectorization levels 1 and 2 require optimization level 3, so debugging vectorized code always involves debugging optimized code.

Debugging is least complicated for programs compiled at optimization level 0 with no vectorization. Optimization level 0 provides object code that most closely follows the source code, so a bug found in the executing object code can be most easily and directly traced to its corresponding source statement.

Debugging optimized or vectorized programs may be necessary, but it requires careful, informed interpretation of the results. You must recognize certain actions taken by the compiler for optimization level 1, 2, or 3, and vectorization level 1 or 2. These actions are:

Register optimization
Retaining values in registers instead of in storage.

Common expression elimination
Eliminating duplicated instructions by retaining subexpression values for later use.

Strength reduction
Replacing an operation by a faster one to improve execution of DO-loops.

Code motion
Altering the placement of calculations, usually by moving instructions from inside a loop to outside

Vectorization
Executing certain computations in DO-loops with vector instructions rather than scalar instructions.

Because of these actions, debugging optimized or vectorized programs can sometimes be difficult and confusing. Interactive Debug will not be able to counter all the effects of optimization or vectorization, but will issue a warning message to inform you that optimized or vectorized code is being debugged. These messages are issued at your first attempt to reference a variable in an optimized or vectorized program unit. Rather than appearing every time an affected command is used, such messages appear only once for each program unit.

If you must debug optimized or vectorized code, you can determine how optimization or vectorization has affected your program by looking at the listing and vector report.

Optimization Levels and Functions

When debugging optimized code, you should expect to get different results from Interactive Debug commands than with nonoptimized code. These unexpected results are actually a result of the optimization process.

Statements that have been eliminated or moved as a result of optimization or vectorization occupy no storage and consequently cannot have debugging hooks. If you try to set a breakpoint at such a statement, a message will indicate that it is "collapsed," which means no machine code has been generated for the statement. Setting or listing variables that are carried in registers across statement boundaries produces unpredictable results because Interactive Debug references the storage locations for such variables, and the register versions (which are the "real" values) will not be displayed or set.

Examples of how these possibly misleading results occur and can be recognized are given in this section.

Optimization Level 0

Optimization level 0 is the recommended level of optimization when a program is to be debugged, or is being compiled to check syntax. It provides the fastest compile time, although it has the least efficient execution time.

The compiler may perform some minor optimizations, even at optimization level 0. However, this is suppressed if the SDUMP option is in effect. Because SDUMP is required for debugging, you can be sure there was no optimization if you are debugging a program unit compiled with optimization level 0.

Even at optimization level 0, variables that are declared but never used in a program have no storage allocated to hold their values. If Interactive Debug claims that a certain variable does not exist, check to see that the variable is actually used in the program.

Optimization Level 1

This option performs register and branch optimization. Variables are retained in registers where possible and time is saved by eliminating unnecessary loads and stores. Branching is improved by the use of RX format branch instructions. This provides a moderate level of optimization for programs that do not have nested loops. Loop structure is not considered.

At OPT(1), all the statements in a block are considered when attempting to use a value already in a register. (Think of a block as a section of code that must always be executed sequentially.) For purposes of debugging, it can be assumed that OPT(1) attempts to use variables and constants placed in registers earlier in the block.

For example:

```
29              NUM = 1
30      10      SUM = SUM + NEXT(NUM)
31              NUM = NUM + 1
32              AVG = SUM / NUM
33              TABLE (NUM) = NEXT(NUM)
34              IF (NUM.LT.100) GOTO 10
36              NUM = NUM * 2
```

Because the flow of control is sequential in the block of statements from ISN 30 to ISN 34, the optimizer attempts to retain variables in registers over the entire span of the block. For example, SUM may be placed in a hardware register at ISN 30 and that register may be used again at ISN 32.

The register used for NUM at ISN 29 might not be used for NUM at ISN 30 because ISN 30 has a label. The label indicates that ISN 30 is a possible target for a branch. Therefore, the value of NUM must be obtained from storage in ISN 30.

Interactive Debug accesses the storage location associated with a variable whenever processing a command that refers to the variable. When code is optimized, the actual value of a variable may be kept in a register; therefore, altering or displaying variables can result in misleading results. Before altering a variable at a breakpoint, examine the statements preceding and following the statement at which the breakpoint was set. Look for a block that has a sequential execution path. In the example above, you should not alter NUM while stopped at a breakpoint at ISN 32, and then expect the computation at ISN 32 to use the new value.

A label begins a new block. A branch, which may be explicit as in the IF statement, or implicit as in the DO statement, terminates a block. Altering NUM while at ISN 30 should produce normal results.

At OPT(1), the results of an assignment statement are always stored. Be careful about subsequent statements that use the variable again if the variable has been altered at an intervening breakpoint.

Optimization Levels 2 and 3

Optimization levels 2 and 3 are both designed to produce as much optimization as possible. The algorithms used are sophisticated and effective, but the general principles are relatively easy to understand.

With levels 2 and 3, control and data flow analysis is done for the entire program. This analysis allows optimizations such as common expression elimination, strength reduction, code motion, and global register assignment. Particular attention is paid to innermost loops.

VS FORTRAN Version 2 performs a very detailed control flow analysis and data flow analysis at OPT(2) and OPT(3). A control flow graph of the program is constructed. A data flow analysis is performed based on the flow graph used in conjunction with the "set and fetch" information for the program's data.

Optimization Level 2: This option performs full text and register assignment. It is identical to OPTIMIZE(3) except that certain optimizations are suppressed in order to provide *interruption localizing*. The basic rule in interruption localizing is: Do not move any code out of a loop if it might cause an interruption that would not occur without optimization. (Note that common expression elimination may result in apparent code motion **within** a loop, but if evaluation of the expression could cause an interruption, it will not be moved outside the loop.)

An example of such an interruption would be a possible division by 0 within a loop. If the division were to be moved out of the loop by the optimizer, it could be moved to a position where it is no longer under the control of an IF statement that checks for the 0 condition.

For example, in the loop

```
      DO 2 J=1,N
        IF(K.NE.0)M(J)=N/K
2     CONTINUE
```

the code to evaluate the expression N/K could be moved outside the loop, because it is invariant for each iteration of the loop. However, at OPT(2), it will not be moved.

Invariant computations involving floating point arithmetic or integer division (including the MOD function), or intrinsic function calls with invariant arguments, will not be moved out of a loop.

Optimization Level 3: This is the highest level of optimization. Invariant computations are moved outside loops wherever possible. This may result in unanticipated interruptions, but incorrect answers will not be generated from a legal program. The only difference from OPT(2) is that an extra error signal is possible.

If the preceding example were compiled at OPT(3), the invariant computation N/K would be moved outside the loop as follows:

```
      itemp=N/K
      DO 2 J=1,N
        IF(K.NE.0)M(J)=itemp
2     CONTINUE
```

where `itemp` is a compiler-generated temporary.

If K is zero, an unanticipated interruption (for integer division by zero) will occur in calculating `itemp`. However, the values stored in the elements of the M array will be the same as if the computation had not been moved outside the loop.

Vectorization Levels and Functions

When debugging vectorized code, you should expect that you may get different results from Interactive Debug commands than when debugging nonvectorized code. These results are caused by the vectorization process. Examples of how these different results occur and can be recognized are given in this section.

If you must debug fully vectorized programs, keep these points in mind:

- You may not be able to suspend the program inside a vectorized loop because there may not be any hooks there. This means that you cannot set breakpoints or examine variables inside a vectorized loop.
- There may be multiple instances of the same ISN if vectorization has split a loop. In this case, breakpoints can be set only at the first instance.
- Keep in mind that optimization is still in effect even if you set breakpoints outside the vectorized loop.

Vectorization Level 1

This option requests vectorization on a loop-by-loop basis. VS FORTRAN Version 2 must be able to vectorize every statement in the selected loop with no reordering; otherwise none of the statements in the loop will be vectorized. Optimization level 3 must be in effect. If the level is not explicitly specified, it will be forced to level 3 by the compiler.

When you specify vectorization level 1, the VS FORTRAN Version 2 compiler attempts to convert complete DO-loops into vector object code. Nests of DO-loops are analyzed, and loops are marked as vectorizable or not. If any statement within the loop is not vectorizable, the loop is not selected for vectorization.

For example, the following nest of loops is vectorizable because all statements within it can be vectorized.

```
      DO 1 K=1, N
        DO 1 J=1, N
          DO 1 I=1, N
            A(I,J,K)=B(I,J,K)+P(J,K)*Q(J,K)
1           E(K,J,I)=F(K,J,I)+X(I,J)*Y(I,J)
```

Vectorization level 1 does not itself cause code motion. However, optimization level 3, which is required, might.

Vectorization Level 2

This option requests vectorization on a statement-by-statement basis. As much of a loop as possible will be translated into vector object code, and the remainder will be translated into scalar object code. Optimization level 3 must be in effect. If it is not explicitly specified, it will be forced to level 3 by the compiler.

When you specify vectorization level 2, the VS FORTRAN Version 2 compiler attempts to convert as many statements as possible into vector object code. Nests of DO-loops are analyzed, and certain groups of statements in nests are marked as vectorizable or not. The compiler selects for vectorization only those statements it knows will run faster when compiled into vector instructions.

For example, the following DO-loop is not vectorizable under vectorization level 1 because one of the statements (`A(I)=A(I+1)+A(I-1)`) within the loop is not vectorizable:

```
      DO 300 I=1,100
        A(I)=A(I+1)+A(I-1)
        C(I)=D(I)
300   CONTINUE
```

However, this program can be vectorized at level 2, by statement, by creating a separate scalar loop for the unvectorizable statement. After vectorization, the loop will look like this in your vector report listing:

```
ISN   FLAG NESTING *....*...1.........2.........3.........4...
---   ---- -------

.
.
.
0005 SCAL +-------      DO 300 I=1,100
0006      |_______      A(I) = A(I+1) + A(I-1)

0005 VECT +-------      DO 300 I=1,100
0007      |_______      C(I)=D(I)
0009                  STOP
0010                  END
```

Using vectorization level 2, it is possible for duplicate ISN's to be assigned by the compiler. If this occurs, you can only assign a breakpoint to the first occurrence. The LISTFREQ command allows you to list all statements, including any duplicates.

Some Practical Examples: Optimization

Now that we have discussed optimization and vectorization levels, we can look at a few simple examples. Remember that vectorized programs are always optimized at OPT(3), so problems in debugging associated with optimization occur in vectorization as well. Examples that apply only to vectorization are explained in "Some Practical Examples: Vectorization" on page 105.

Constant Propagation: Constant propagation is a calculation done at compile time. The precalculated results are used at execution time. For example:

```
II = 10
JJ = II          is replaced by          JJ = 10
```

Results may be unexpected when you change the value of II at a breakpoint and later look at the value of JJ.

Common Expression Elimination: Common expressions are those in which the result of a calculation is available because of a previous calculation. In the example below, the calculation `J + K` is performed twice:

```
20        I  = J + K
21        II = 10
22        JJ = J + K      can be replaced by      JJ = I
```

Note the opportunity for unexpected results when you change the value of J or K while at ISN 21. Because the recalculation of the common expression J+K has been eliminated, JJ will be equal to I, not the new sum of J + K. If there is a breakpoint set at ISN 21 and the value of K is altered, it will have no effect on the value assigned to JJ.

Backward Movement: If the variables involved in an expression are not changed in a loop, it may be possible to move an expression outside of the loop. In the example below, A = B could be moved out of the loop:

```
20        X = Y
21        DO 10 I = 1,10
22        ARR(I) = ARR (I) + I
23        A = B
24   10   CONTINUE
25        WRITE(6,*) X,A
```

The program then becomes:

```
20        X = Y
21        A = B
          DO 10 I = 1,10
22        ARR(I) = ARR (I) + I
23
24   10   CONTINUE
25        WRITE(6,*) X,A
```

The optimizer recognizes that both A and B are loop invariant, and the computation can therefore be moved outside the loop. Note that ISN 23 is still there, but is empty, or "collapsed." An attempt to set a breakpoint at ISN 23 would result in an Interactive Debug error message.

Strength Reduction: Strength reduction replaces one operation by a faster one to improve execution of DO loops.

Let's assume that elements of an array such as ARR, in the example below, are 4 bytes long.

```
21        DO 10 I = 1,10
22        ARR(I) = ARR(I) + B
23   10   CONTINUE
```

Since each element is 4 bytes long, the subscript I must be internally multiplied by 4 to obtain the proper offset from the start of the array. The addresses of elements in this array actually increase in steps of 4, rather than in the steps of 1 implied by the DO loop. Thus, the offset takes on values of 4, 8, 12, and so on, as the subscript I takes on values of 1, 2, 3, and so on.

Strength reduction replaces one operation (in this case, the internal multiplication) by a faster one (in this case, addition). The loop code generated for the loop would contain instructions to add 4 to each iteration of an internally maintained offset, rather than keeping the subscript I and multiplying it by 4 on each iteration.

If strength reduction occurred in the example above and you set a breakpoint at the statement following CONTINUE to display the value of I, the value would be 1. This is a misleading value. I is never really used in the optimized loop. A compiler-generated temporary is used instead to calculate the offsets.

Global Register Assignment: Global register assignment is another way to improve the generated code. The example below illustrates the principle involved.

```
21        DO 10 I = 1,10
22        J = J + I
23   10   CONTINUE
```

results in machine code that performs these actions:

```
            load 1 into a register
            store register at I
          * add register to J
            add 1 to register
            compare register contents with 10
            branch (to *) if compare is less than or equal
            store final value at J
```

This is not an exact representation of the machine code, but illustrates the principle of register assignment. Instructions that use registers are faster than instructions that reference storage. For the duration of this loop, I is assigned to a register.

If you were to set a breakpoint at ISN 22 and display the variable I, the value would be 1.

If a program works correctly at OPT(0) and fails when optimized at a higher level, check for variables assigned to registers, or uninitialized variables. Frequently, a variable kept in storage at OPT(0) is assigned to a register under OPT(3). This is only one of the effects of debugging an optimized program.

Some Practical Examples: Vectorization

Grouping of Data Elements: When a DO-loop is vectorized, individual loops that operate on single elements are modified to produce less-frequently executed loops that operate on *groups* of elements. When you try to set a breakpoint at a particular statement in a program unit that has been vectorized, you may find that the loop index and/or the contents of storage have unexpected values. For example, examine the following code:

```
1           REAL A(100), B(100)
2           DO 2 K = 1, N
3     2       A(K) = B(K)
```

The single loop in the above example may actually look like this in the vectorized code:

```
      DO 2 K  = 1, N, Z
      DO 2 KK = K, K+MIN(N-K,Z-1)
2       A(KK) = B(KK)
```

where Z is the size of the group, and the inner "loop" is actually performed by vector instructions.

After vectorization, the single loop over individual elements becomes a loop over groups of elements. This loop contains a second "conceptual" loop, executed in vector instructions, over elements in the group.

The original statement labeled "2" (ISN 3) no longer exists. It has been placed under ISN 2 in a different form. If you try to set a breakpoint there, Interactive Debug may issue a message saying that it cannot find the statement.

Statement Reordering: To vectorize certain loops, the order of execution of statements may have to be modified. When you later try to debug a statement in this loop, you may find that it doesn't appear where you expected to find it.

For example, you may have coded the following:

```
1           DIMENSION A(100),B(100),C(100)
2           DO 500 I = 2,100
3                A(I) = B(I-1) * 3.0
4                B(I) = C(I) * 3.0
5     500   CONTINUE
```

But your vector report listing after vectorization at level 2 may look like this:

```
ISN   FLAG NESTING *....*...1.........2.........3.........4....
____  ____ _______
0001                      DIMENSION A(100),B(100),C(100)
0002 VECT +-------        DO 500 I = 2,100
0004      |                    B(I) = C(I) * 3.0
0003      |_______             A(I) = B(I-1) * 3.0
0006                      END
```

In this case, ISNs 3, 4, and 5 no longer exist. They have been placed under ISN 2 in a different form. You will not be able to set breakpoints at these statements.

Loop Distribution: During vectorization, statements coded within a single DO loop are sometimes distributed to separate loops during vectorization.

For example, if you code this loop:

```
1            REAL A(200), B(200)
2            DO 20 I = 2,100,2
3              A(I) = A(I) + 2
4              B(I+2) = B(I) + 2
5     20     CONTINUE
```

your vector report listing might look like this after vectorization:

```
ISN  FLAG NESTING *....*...1.........2.........3.........4....
+___  ____ _______
0001                      REAL A(200),B(200)
0002 VECT +-------        DO 20 I = 2,100,2
0003      |_______              A(I) = A(I) + 2

0002 SCAL +-------        DO 20 I = 2,100,2
0004      |_______             B(I+2) = B(I) + 2
0006                      END
```

You will not be able to set breakpoints inside the original loop. If you attempt to debug statements within the original loop, you will find that ISNs 3, 4, and 5 have disappeared. They have been placed under ISN 2 in a different form.

Commands Affected by Optimization and Vectorization

A number of Interactive Debug commands are affected by the processes of optimization and vectorization. These commands may produce unexpected or inaccurate results when used in debugging optimized or vectorized code. This section describes the commands that may function differently when used with optimized or vectorized code.

AT Command

Use of the AT command with optimized or vectorized code can be confusing if some statements have been relocated or removed. If a statement has been moved or completely eliminated, Interactive Debug will issue a message telling you that no breakpoint can be established at that statement.

AUTOLIST Command

The AUTOLIST command displays the contents of the storage area that the compiler assigned to a specified variable. When the program is optimized or vectorized, it is possible that the current value of a variable, or of an array element, is only in a register. As a result, what is displayed by the AUTOLIST command may not be the current value of the variable or variables.

GO Command

You will receive a message if you issue a GO command that refers to a statement identifier in a program unit compiled with an optimization level greater than 0 (including vectorization levels 1 or 2). The message indicates that results are unpredictable, and requires your confirmation before proceeding. The VS FORTRAN Version 2 optimizer produces code assuming that the possible paths through a module are known—for example, that a sequence of ten assignment statements will always be executed in order. Based on this assumption, a register may be loaded once at the beginning, and used by subsequent statements without being reloaded. If you issue a GO command referring to a statement in the middle of that sequence, you may be bypassing code that causes an important register to be loaded. Results are unpredictable when a statement using that register is subsequently executed.

IF Command

IF specifies a condition to be tested. This involves examining the value assigned to a variable, and may not produce the correct results if the value is being kept in a register.

LIST Command

The LIST command displays the contents of the storage area that the compiler assigned to a specified variable. When the program is optimized or vectorized, it is possible that the current value of a variable, or of an array element, is only in a register. As a result, what is displayed by the LIST command may not be the current value of the variable or variables.

LISTFREQ Command

The LISTFREQ command lists statements even though they have been moved, optimized away, or vectorized away, but indicates that these are "collapsed statements" in the "frequency" field. Execution counts cannot be maintained for these statements.

SET Command

Like LIST, the SET command may produce confusing results. The SET command alters the contents of a storage location assigned to a variable by the compiler. However, optimized or vectorized code may not be using this storage location to contain the current value. Even if the optimized or vectorized code has stored the value in this location, VS FORTRAN Version 2 may use a copy of the register value or values (without reloading it from storage) for the next use of the variable. Furthermore, subsequent instructions may store the values from the registers into storage, thereby overwriting the value just stored by the SET command. Under either of these circumstances, changing the value in storage will not have the desired effect.

WHEN Command

The WHEN command uses the storage value of a variable, and may not produce correct results if the value is being kept in a register.

Warning Messages

On the first AUTOLIST, IF, LIST, SET, or WHEN command in any program unit compiled with OPT(1), OPT(2), or OPT(3), or with VECTOR(LEV(1)) or VECTOR(LEV(2)), a warning message will appear, stating that the program unit contains optimized or vectorized code. Subsequent commands involving the same unit will not result in further messages.

Improving Performance when Using Interactive Debug

When you use VS FORTRAN Version 2 Interactive Debug with your program, the program runs slower. This is typical of high-level language debuggers that get control at every statement boundary to check for breakpoints or other conditions.

The greatest impact on performance occurs when you are debugging a program that executes many very simple VS FORTRAN statements, with debugging hooks at all (or most) of them. Having very small subroutines will also degrade performance.

Techniques to Improve Performance

For normal debugging of typical programs, you may find that performance is satisfactory. By limiting the amount of input to be processed (which you would probably do anyway when debugging), you may avoid any problems with performance. However, if you find performance seriously affected when running under Interactive Debug, there are a number of things you can do to improve it.

- Try to limit the number of program units being debugged at one time. The "include" file (AFFON) can be used to tell IAD which program units you want to debug, allowing the rest to run at full speed. This is especially important if some subroutines are called often. You can debug some of the program units in one debugging session, and others in another debugging session.

- If possible, insert only entry/exit hooks in heavily-used subroutines. This may be all you need to decide whether the subroutine is incorrectly changing a variable, for example. If you find that a such a subroutine is producing errors, you may be able to use the AT command with a command list to temporarily generate correct values while you are debugging other subroutines. Later, you can restart the debugging session with a different AFFON file and concentrate on the other subroutines that are generating incorrect results.

- If possible, avoid putting hooks in heavily executed code, especially if it consists of many very simple statements. You can use the "include" file (AFFON) to specify which parts of each program unit are to have hooks inserted in them.

Recognizing Some Common Errors when Setting up a Debugging Session

If you try to begin a debugging session without success, or if you try to debug a program unit that Interactive Debug considers nondebuggable, look through the following list of common errors to find a possible solution:

Problem	Solution
All program units are nondebuggable.	Check your AFFON file for incorrectly coded AFFON entries. Insure that the AFFON file matches the program being run. Make necessary corrections try again.
One or more program units are nondebuggable.	Check to see whether the program unit was compiled without SDUMP, or was compiled with a FORTRAN compiler level that is not supported. If so, recompile the program unit.
Interactive Debug is not loaded and a debugging session is not initialized.	Be sure you have specified the DEBUG execution-time option, or have taken steps to override the default option. Be sure you are running your program with a level of the VS FORTRAN library that supports Interactive Debug. If a program was link-edited with an older library, re-link-edit it with the current library.
Under ISPF, you receive the message "LOG FILE NOT FOUND," and Interactive Debug never runs.	Be sure that the VS FORTRAN Version 2 libraries are accessible and that you have allocated the FT06F001 file, then rerun the program. Also be sure you have allocated enough storage.

Part 2. VS FORTRAN Version 2 Interactive Debug Reference

This section contains:

- VS FORTRAN Version 2 Interactive Debug syntax conventions
- VS FORTRAN Version 2 Interactive Debug statement identifier conventions
- A functional summary of Interactive Debug commands
- Descriptions of all Interactive Debug commands

The commands are in alphabetic order. Each command description includes the command syntax, function, and examples.

Chapter 8. Interactive Debug Commands

This chapter summarizes all VS FORTRAN Version 2 Interactive Debug commands.

Syntax Conventions

Symbols used to describe the syntax of VS FORTRAN Version 2 Interactive Debug commands are the same as in other VS FORTRAN Version 2 publications.

- The syntax for each command is shown in a box, with any allowable abbreviations for the command name shown above the box.
- Lowercase character strings within the box indicate information you must supply.
- Uppercase character strings indicate Interactive Debug keywords.
- Square brackets ([ ]) set off optional portions of a command.
- Braces ({ }) indicate a choice. One or more vertical lines (|) within the braces separate the choices.

  If an optional portion consists entirely of a choice ([{... | ... | ... | ...}]), the braces are omitted.
- Where optional keywords are listed, the default is underlined. An explanation of the default is provided in the text below the box.
- Special characters (for example, colons, parentheses, and asterisks), must be entered as shown.

Keyword operands can be abbreviated to any unique shortened form. For example, NOTIFY can be abbreviated to NOT (but not to NO, to distinguish it from NONOTIFY). Command names can be abbreviated only with the explicit abbreviation shown in this manual for the command.

Statement Identifier Conventions

Follow these conventions when using any Interactive Debug command with VS FORTRAN statement labels, ISNs, or sequence numbers:

- All statement labels (numbers in columns 1 through 5 of the source program) must be preceded with a slash (for example, /100).
- The type of statement number is determined by the compiler and the options used when the program was compiled. A statement number is either an ISN or a sequence number.
  - If the program was compiled with VS FORTRAN Version 2, you can use either the ISN or the sequence number. ISNs are the default; to use sequence numbers, specify the compiler option `SDUMP(SEQ)`.
  - If the program was compiled with VS FORTRAN Version 1, and if both TEST and NOSDUMP were in effect or if VS FORTRAN Release 2 was used, and if the first record has a number in columns 73 through 80, use the sequence numbers. If the first record does not have a number in positions 73 through 80, use the internal statement number (ISN) provided in the compiler listing. VS FORTRAN Version 1 sequence numbers are not supported in full screen mode.
  - For all other combinations of VS FORTRAN Version 1 options, use the ISN.

Commands

Figure 25 lists the commands by function.

Functions	Commands
Controlling Program Execution	AT GO ENDDEBUG HALT LISTBRKS NEXT OFF OFFWN RESTART STEP WHEN
Monitoring and Modifying Variables	AUTOLIST DESCRIBE IF LIST QUALIFY SET
Processing Sequential Files	BACKSPACE CLOSE ENDFILE RECONNECT REWIND
Controlling Full screen Display	COLOR LISTINGS MOVECURS POSITION PREVDISP PROFILE REFRESH SEARCH WINDOW
Tracing and Timing	ANNOTATE LISTFREQ LISTSAMP LISTSUBS LISTTIME TIMER TRACE WHERE

Figure 25 (Part 1 of 2). Functional Summary of Interactive Debug Commands

Correcting Errors	ERROR FIXUP
General	* or " (comment) HELP PURGE QUIT SYSCMD TERMIO

Figure 25 (Part 2 of 2). Functional Summary of Interactive Debug Commands

The commands are presented below in alphabetic order. A summary of the commands appears in Appendix B, "Interactive Debug Command Summary" on page 225.

* or " (Comments)

An asterisk (*) or a quotation mark (") can be used to insert comments into the log of debugging activity. When running in an ISPF environment or in batch, a log of debugging activity is maintained. A log is also available in a CMS line-mode environment, if you specify `CP SPOOL CONSOLE START`. Comments may then be used to identify items for later examination.

Abbreviation: None

Syntax

{* | "} [comment]

comment
is any character string. This character string will appear in the log of the debugging session.

Notes:

1. *When running in a CMS environment, the use of a quotation mark (") to identify a comment command may be inhibited because the (") mark is normally assigned to be the CMS escape character. In this case, a double quotation mark is required to enter a comment. For example:*

   ```
   "" This is a comment.
   ```

2. *Comments entered in a command list as part of an AT command are ignored. They are also ignored if entered as the command portion on the IF command, or if entered in an attention exit.*

Example

```
*    This is how comments are inserted into the log.
"    A quotation mark also works.
```

ANNOTATE Command

The ANNOTATE command provides program sampling information in either of two forms:

1. As a source listing to the AFFPRINT file, showing sampling or frequency data.
2. As a bar chart overlay on the source listing window, showing the sampling or frequency data

Abbreviation: AN

Format 1

Syntax for Copying Source Listings to a Print File

```
ANNOTATE {unit | (unit list) | * }
 [SAMPLING [DIRECT | CALLED | ALL] | FREQUENCY]
```

Format 2

Syntax for Providing a Bar Chart on the Source Listing Window

```
ANNOTATE [ON | OFF | TOGGLE]
 [SAMPLING[DIRECT | CALLED | ALL] | FREQUENCY]
```

unit
specifies the name of a program unit whose listing is to be copied to AFFPRINT with sampling or frequency information added. The program unit must be a VS FORTRAN program unit compiled with the SDUMP option. Listing files must be identified either by specification in AFFON or on the listings data set specification panel.

unit list
specifies a list of program units whose listings are to be copied to AFFPRINT with sampling or frequency information added. (See restriction under *unit*, above.)

*****
specifies that all available program unit listings are to be copied to AFFPRINT with sampling or frequency information added. Listings will be copied only for VS FORTRAN program units compiled with the SDUMP option. Listing files must be identified either by specification in AFFON or on the listings data set specification panel.

ON
specifies that the source listing window is to be shown with overlaid sampling or frequency bar charts (valid in ISPF only). Note that ON cannot be abbreviated.

OFF
specifies that overlaid sampling or frequency charts are NOT to be shown in the source listing window (valid in ISPF only). Note that OFF cannot be abbreviated.

TOGGLE
specifies that overlaying of the source listing window with sampling or frequency bar charts is to be changed to ON if currently OFF, or to OFF if currently ON (valid in ISPF only). This form is intended for assignment to a PF key. Note that TOGGLE cannot be abbreviated.

SAMPLING
requests sampling information annotation. For Format 1, SAMPLING is the default when neither SAMPLING or FREQUENCY is specified.

DIRECT
counts interruptions occurring in the code. DIRECT is the default when SAMPLING is specified or defaulted.

CALLED
counts interruptions occurring in lower-level routines. This option is valid only if sampling was initiated with the CALLED option.

ALL
indicates that sampling counts are to be the sum of the DIRECT and CALLED counts.

FREQUENCY
requests statement frequency annotation information

Notes:

1. *Annotated information listings show a summary of all program units and entries for which sampling counts have been accumulated. This summary includes non-FORTRAN units, FORTRAN units compiled with NOSDUMP, and program units not specified in the unit list operand of the command.*

2. *When SAMPLING is specified along with the ON, OFF, or TOGGLE options, the current ANNOTATE settings are unchanged if FREQUENCY, DIRECT, CALLED, or ALL are not specified. The initial ANNOTATE settings are OFF and FREQUENCY. If SAMPLING is done, the ON and DIRECT options are turned on, turning off FREQUENCY.*

3. *If no operands are specified, the current ANNOTATE settings are displayed.*

4. *Only the first three options (unit, unit list, *) may be used in a restart file or in batch mode. Annotation is then limited to listings identified in the AFFON file.*

5. *In the histogram bars that are shown in the annotated source listings, each asterisk (*) represents 2% (rounded). In order to prevent the lines from getting too wide, any histogram bar that would exceed 17 characters (34%) is truncated to 17 characters, and the rightmost character is shown as a plus sign (+) instead of an asterisk.*

Example 1

Overlay the source listing window with bar charts indicating the sum of both DIRECT and CALLED sampling.

```
annotate on all
```

Example 2

Remove the bar chart overlays from the source listing window.

```
annotate off
```

Example 3

Query the ANNOTATE settings

```
annotate
```

Example 4

Copy the VS FORTRAN source listings (if known) for all program units to AFFPRINT, annotating them with DIRECT sampling information.

```
annotate *
```

Example 5

Copy the VS FORTRAN source listings for MAIN and SUB1 to AFFPRINT, annotating them with frequency information.

```
annotate (main,sub1) frequency
```

AT Command

The AT command sets breakpoints in a VS FORTRAN program at executable statements, entry points, or exit points. When defining a breakpoint, you may specify a list of Interactive Debug commands to be executed whenever the breakpoint is reached. Execution will be suspended **before** the specified statement is executed. For examples of how to use the AT command, see "Controlling Program Execution" on page 74 and "Using Command Lists" on page 76.

The AT command can also be used without parameters as one of the cursor commands under ISPF Version 2. For more information, see "Using Interactive Debug Features under ISPF Version 2" on page 24.

Abbreviation: None

Syntax

```
AT [qual.]{number [:[qual.]number] | ENTRY | EXIT}

 | (number/ENTRY/EXIT list)

 [(command list)] [COUNT(n)][NOTIFY | NONOTIFY]
```

qual
specifies a program unit name prefix to temporarily override the current qualifier for the prefixed operand only.

number
specifies the statement label, ISN, or sequence number of an executable VS FORTRAN statement. Precede a statement label with a slash to distinguish it from an ISN or sequence number. The type of statement number used (ISN or sequence number) is determined by the compiler and the options used. See "Statement Identifier Conventions" on page 114.

number:[qual.]number
specifies a range of statement labels and/or statement numbers (either ISNs or sequence numbers). A breakpoint is set at each executable statement within the range. Statement labels can be combined with ISNs or sequence numbers in the range, but the first and last must both be executable statements. Precede each statement label with a slash.

Statement identifiers can be qualified with a program unit name. The default program unit for the first identifier is the current qualifier. The default program unit for the second identifier in a range is the program unit specified or defaulted for the first identifier. Both identifiers must have the same program unit in effect.

For example, `A.5:6` is the same as `A.5:A.6`. However, `A.5:B.6` is invalid.

ENTRY
specifies that an entry breakpoint is to be set. The breakpoint occurs in the prolog of the program unit. The subroutine or function is not yet active, so dummy arguments are not accessible, and a `GO` with a statement identifier

cannot be issued. The breakpoint occurs regardless of which entry point of the program unit is entered.

EXIT
specifies that an exit breakpoint is to be set. The breakpoint occurs after the last FORTRAN statement in the program unit has been executed.

(number/ENTRY/EXIT list)
specifies a list of statement labels, ISNs, sequence numbers, entry breakpoints, exit breakpoints, and/or ranges of numbers. A breakpoint is set at each executable statement specified. Enclose the list in parentheses, and separate entries with commas or blanks. Precede each statement label with a slash.

Note: If a statement that does not have a debugging hook is specified in the number list, an error message is issued but breakpoints are still set at the remaining statements.

(command list)
specifies a list of commands to be executed at the specified statement. If more than one statement is listed, the command list is executed at every statement listed. Enclose the command list in parentheses, and separate commands with percent signs (%).

The qualification in effect when the AT command is issued is used as the default for any variables and statements referenced in the command list.

COUNT(n)
specifies that a breakpoint occurs only every nth time the specified statement is reached. Specify n as an integer. If more than one statement is specified, the iteration count is applied independently to each listed statement.

NOTIFY
specifies that the location of every breakpoint is to be displayed when it is reached. This includes breakpoints that cause the program to resume without user intervention. This is the default action.

NONOTIFY
specifies that no notification is given when the AT command list contains a command, such as GO, that causes execution to resume.
NOTIFY/NONOTIFY has no effect on the notification that is done when a breakpoint is reached and execution is suspended. You cannot turn off notification when execution is suspended.

Notes:

1. *Unless you have attached a command list containing a GO, ENDDEBUG, STEP, RESTART, or QUIT command, execution is always suspended when a breakpoint is reached.*

2. *You may include any VS FORTRAN Version 2 Interactive Debug command in the command list except HELP, FIXUP, LISTSAMP or any full screen display commands.*

3. *A new valid AT command overrides any previous AT command in effect for a specific statement.*

4. *HALT, NEXT, WHEN conditions, and AT breakpoints all cause execution to be suspended. When execution is suspended for multiple reasons, messages are issued for all the reasons.*

5. *At ENTRY, the following restrictions apply:*

   - *The GO command with a statement ID is not permitted.*
   - *Variables in a dynamic common cannot be referenced.*
   - *Dummy arguments are not accessible.*

6. *You can associate a remark with a breakpoint by using the LIST command to display a quoted string as part of the command list for the breakpoint. For more information, see "LIST Command" on page 158.*

7. *You can set a breakpoint only on statements that have a* **debugging hook**. *You cannot set a breakpoint on statements outside the AFFON statement list, on statements that are collapsed, on the ENTRY of a VS FORTRAN main program, or on the EXIT of a VS FORTRAN main program. For more information on debugging hooks, see "Setting Breakpoints at Debugging Hooks" on page 72.*

8. *A command in an AT command list that causes execution to resume will cause the remainder of the command list to be ignored. These commands are GO, STEP, ENDDEBUG, and RESTART. QUIT and HALT IMMED also cause the remainder of the command list to be ignored.*

9. *AT is not permitted after the VS FORTRAN program has terminated.*

Example 1

Stop execution every 10th time the program reaches the beginning of a loop that begins at statement label 65.

```
at /65 count(10)
```

Example 2

Set breakpoints at ISNs 180 and 220 in the currently qualified program unit, at every executable statement in program unit SUB1 between ISN 10 and statement label 50, and at entry to program unit CHECK. Execution is suspended at each of these points.

```
at (180 220 sub1.10:/50 check.entry)
```

Example 3

At ISN 140, list the value of variable A, set variable I to 10, and continue execution. Except for listing the value of A, no notification is given.

```
at 140 (list a%set i=10%go) nonotify
```

Example 4

At ISN 5 in program unit STUB, set variable ANSWER to 100 and exit the subroutine.

```
at stub.5 (set stub.answer=100%go stub.exit)
```

AUTOLIST Command

You can use the AUTOLIST command in full screen mode to specify up to ten lines of variable and array information. The information is then displayed at the top of the screen each time the program is suspended, or when the display is refreshed during animation.

The containing program unit name is shown with all variable or array names. Array elements may be displayed outside the defined dimensions. You can remove the display by issuing the AUTOLIST command with no operands.

You cannot use AUTOLIST in line mode or batch mode. If you try to, Interactive Debug issues an error message.

Abbreviation: AL

Syntax

```
AUTOLIST [{[qual.]name[:[qual.]name ] | *
 | 'string' | number | (specification list)}
  [FORMAT [(code)] | DUMP [(code)]]]
```

[qual.]name
specifies the name of a variable, array, or array element used in the program. If a qualifier is specified, it overrides the current qualifier for the specified name. Substring notation may be used with string variables.

[qual.]name:[qual.]name
specifies a range of variable, array, or array element names used in the program. If a qualifier is specified, it overrides the current qualifier for the specified name.

AUTOLIST displays all storage locations between the two variables. Unless FORMAT or DUMP is specified, the format of the displayed storage is the same as the type of the first variable.

*****
specifies that a list of all the variables and arrays in the currently qualified program unit is desired. Unless FORMAT or DUMP is specified, each is displayed according to its own type.

'string'
specifies a character string to be displayed as a remark. You can use this operand to help identify the AUTOLIST display.

number
specifies an integer or real numeric constant to be displayed as a remark. This function can be useful in converting numbers, when used in conjunction with the FORMAT option.

specification list
identifies a list of individual specifications. The list must be enclosed in parentheses. Individual entries must be separated by commas or blanks.

FORMAT [(code)] or DUMP [(code)]
specifies a particular data format:

- FORMAT displays the names listed and their values in the specified format.
- DUMP displays the address in storage of the names listed and their values in the specified format.
- (code) specifies the format or dump code for the names to be listed. The default format code is X. The default dump code is Z.
- FORMAT and DUMP are mutually exclusive.

  FORMAT and DUMP codes for the AUTOLIST command are shown in Figure 26 on page 127.

Code	Output
L1	Logical*1
L4	Logical*4
I2	Integer*2
I4	Integer*4
R4	Real*4
R8	Real*8
R16	Real*16
C8	Complex*8
C16	Complex*16
C32	Complex*32
L	Logical with size closest to internal data size
I	Integer with size closest to internal data size
R	Real with size closest to internal data size
C	Complex with size closest to internal data size
X[nnn]	Hexadecimal with nnn bytes per data item (default to internal data size)
Z[nnn]	Hexadecimal with nnn bytes per data item (default to Z4)
A[nnn]	Character with nnn bytes per data item (default to internal data size)
H[nnn]	Character with nnn bytes per data item (default to continuous full line output)

Figure 26. DUMP and FORMAT Codes for the AUTOLIST Command

Notes:

1. *Each time you issue the AUTOLIST command, the set of variables to be listed is redefined. The lists are not cumulative. To turn off the AUTOLIST display, specify AUTOLIST with no operands.*

   For example, if you type `AL (X,Y)`*, then X and Y are displayed. If you later type* `AL Z`*, only Z will be displayed.*

2. *The output is displayed in the first ten lines of the full screen panel. No AUTOLIST output is shown in the scrollable log. If more than ten lines of output are requested, the excess is not displayed.*

3. *Dummy arguments can only be displayed when the program unit in which they are defined is active. For example, when executing in MAIN before calling SUB1, entering:*

   ```
   autolist sub1.a
   ```

will cause a message to be produced if a *is a parameter. Results are unpredictable if you display a dummy argument that is not defined at the entry point called. (Note that a program unit is not yet active when suspended at entry.)*

4. *Variables in dynamic commons can only be displayed if the program unit used to qualify the variable has been activated at least once. (If not, you may receive an error message. However, if a variable has a large displacement in its dynamic common, Interactive Debug cannot detect that it is not initialized.)*

5. *When you request an individual name or list of names, the default format is determined by the type of each variable being displayed. When you request a range of names, the formatting of the values is determined by the format of the first name in the range. The locations of the listed names are not identified in the output. You may, however, specify a different format using the FORMAT or DUMP keyword.*

6. *VS FORTRAN defines storage layout only within arrays, variables in a common block (defined in a COMMON statement), and variables in equivalence groups (defined in an EQUIVALENCE statement). The relative positions of any other names in storage cannot be predicted. Names that you may expect to be adjacent in storage may be widely separated by other data. Therefore, a range specification for names other than array, equivalence, or common variables may produce unexpected results.*

7. *The DUMP option is not permitted with constant operands, including strings.*

8. *The length specification in a FORMAT or DUMP code may be entered with 1 through 3 digits. Thus, I4, I04, and I004 are equivalent.*

9. *A length specification of 0 in character and hexadecimal FORMAT and DUMP codes (for example, A0 or Z0) causes the data to be displayed as a continuous string, rather than split into pieces of some specified length.*

10. *If a FORMAT or DUMP code with no length specification is given for a range of variables or array elements, each variable or array element is displayed separately in the specified format. However, if a length specification is given, Interactive Debug will consider the entire storage area occupied by all the range of variables or array elements, or occupied by the entire array, as if it were broken into pieces, each with a length equal to that specified in the DUMP or FORMAT code, and will display each piece according to the specified format. For example, if PRIMES is a 2 x 3 array of INTEGER*4 values, then:*

    ```
    autolist primes format(x)
    ```

    will cause a display of 6 values, each corresponding to an element of the array. However:

    ```
    autolist primes format(x2)
    ```

    will cause a display of 12 values, each displaying the contents of successive 2-byte storage areas within the array.

11. *An assumed size array cannot be listed by just specifying the array name; the specific element or range of elements must be specified. (An assumed size array is an array with the last upper bound specified as an asterisk (*).) This restriction does not apply to arrays whose last dimension is "1" even though such arrays are otherwise treated as assumed size arrays. However, only the elements whose last subscript is "1" will be displayed if no subscripts are specified.*

12. *For array elements, subscripts may have values beyond the range of the corresponding array dimensions. A warning message will be issued except in the special case where the last dimension is "1" or "*" and only the last subscript is out of range.*

13. *Array subscripts may contain simple arithmetic expressions no more complex than the form "variable plus (or minus) a constant." For example,*

    ```
    autolist ARY(I), autolist ARY(3), autolist ARY(I + 3),
    or autolist ARY(I - 3)
    ```

    are valid forms.

Example 1

Display the value of the variable named NCOUNT whenever the program is suspended or the screen is refreshed during animation.

```
autolist ncount
```

Example 2

Obtain a hexadecimal dump (FORMAT(X)) showing values of array variables A(1,1) through A(2,3). The dump appears whenever the program is suspended or the screen is refreshed during animation.

```
autolist a(1,1):a(2,3) format
```

Example 3

Display several variables in hexadecimal, each with as many bytes as are appropriate for its data type. The information appears whenever the program is suspended, or the screen is refreshed during animation.

```
autolist (i,j,p,q,r) format
```

Example 4

Display an entire array (CHARAY) containing a series of 30-character alphabetic strings so that each character string is separated from the others. (If the array is declared in the program to be a CHARACTER*30 array, then the elements of the array will be separated from each other when the array is listed.) The information appears whenever the program is suspended, or the screen is refreshed during animation.

```
autolist charay format(a30)
```

Example 5

Display the value of the variable REAL1 in SUB1, ARRAY(I,J) in SUB2, and STRING, I, and J in the currently qualified program unit whenever the program is suspended, or the screen is refreshed during animation.

```
autolist (sub1.real1,sub2.array(i,j),string,i,j)
```

Example 6

Display the decimal number 12345 in hex whenever the program is suspended or the screen is refreshed during animation.

```
autolist 12345 f(x)
```

BACKSPACE Command

The BACKSPACE command positions a sequentially accessed external file to the beginning of the previous record. It is similar to the BACKSPACE statement in the VS FORTRAN Version 2 language. This command allows you to move backward in the file, for example to rewrite or reread a record.

Abbreviation: BACKSPAC, BACKS

Syntax

```
BACKSPACE {number | [qual.]integer-variable |
 [qual.]integer-array-element}
```

number
is the number of the I/O unit associated with the sequential file on which the backspace is to be performed.

[qual.]integer-variable
is the name of an integer variable in the VS FORTRAN program. This variable specifies the number of the I/O unit associated with the sequential file on which the backspace is to be performed.

[qual.]integer-array-element
is the name of an element of an integer array in the VS FORTRAN program. This element specifies the number of the I/O unit associated with the sequential file on which the backspace is to be performed.

Notes:

1. *"Number," "integer-variable," or "integer-array-element" must be specified; there is no default number.*

2. *This command may not be issued when I/O is currently active.*

3. *When referring to array elements, the subscripts must use simple arithmetic expressions no more complex than the form "variable plus (or minus) a constant." For example,*

   ```
   A(I), A(3), ARY(I+3) or ARY(I-3)
   ```

   are valid.

Example

Backspace the sequentially accessed external file associated with I/O unit 8 so that the last record written can be rewritten.

```
backspace 8
```

For additional examples of the BACKSPACE command, see "Processing External Files" on page 89.

CLOSE Command

The CLOSE command disconnects a VS FORTRAN external file from an input or output unit. Its usage is similar to that of the CLOSE statement in the VS FORTRAN Version 2 language. This command allows you to close an external file, for example to assign another file to the input or output unit, or to examine the contents of the file.

Abbreviation: None

Syntax

```
CLOSE {number | [qual.]integer-variable | [qual.]integer-array-element}
```

number
is the number of the I/O unit associated with the file that is to be closed.

[qual.]integer-variable
is the name of an integer-variable in the VS FORTRAN program. This variable specifies the number of the I/O unit associated with the file that is to be closed.

[qual.]integer-array-element
is the name of an element of an integer array in the VS FORTRAN program. This element specifies the number of the I/O unit associated with the file that is to be closed.

Notes:

1. *"Number," "integer-variable," or "integer-array-element" must be specified; there is no default number.*

2. *This command may not be issued when I/O is currently active.*

3. *Files used in the program but not explicitly closed will still be open when Interactive Debug gives you control at termination. If you want to examine such a file, you must CLOSE it first.*

4. *Under certain conditions, use of the CLOSE command may make it necessary for you to use the RECONNECT command before your program can perform additional I/O operations on the file. This situation occurs when the OCSTATUS run-time option is in effect, and the program cannot be made to execute an OPEN statement before doing more I/O to the file. (See "RECONNECT Command" on page 192.)*

5. *When referring to array elements, the subscripts must use simple arithmetic expressions no more complex than the form "variable plus (or minus) a constant." For example,*

   ```
   A(I), A(3), ARY(I+3) or ARY(I-3)
   ```

   are valid.

Example under CMS

Reallocate an external output file for I/O unit 8.

```
close 8
sys filedef 8 disk output file a
```

Example under TSO

Reallocate an external output file for I/O unit 8.

```
close 8
sys allocate file(ft08f001) dataset(output.file)
```

For additional examples of the CLOSE command, see "Processing External Files" on page 89.

COLOR Command

The COLOR command allows you to tailor the color and attribute setting of the full screen display under ISPF Version 2. This command cannot be issued in a command list, as the command portion of an IF command, in an attention exit, or a restart file.

The COLOR command displays a panel that allows you to select the color, highlighting and intensity of the various parts of the interactive debug panel. The panel characteristics can be customized for each user.

You can save your color specifications from session to session in the ISPF profile by entering SAVE on the command line when the color panel is displayed. If you do not save a color specification, you are assigned the default. To restore the default if you have changed a color setting, enter DEFAULT on the command line when the color panel is displayed.

Enter END or use the END PF key to return to the debugging session.

Abbreviation: None

Syntax

```
COLOR
```

Note: COLOR will operate as usual with a parameter list, but the panel will include the message "PARAMETERS IGNORED."

The color and attribute selection panel is shown in Figure 27 on page 135

```
   VS FORTRAN INTERACTIVE DEBUG  (COLOR and ATTRIBUTE settings)
COMMAND ===>

                        COLOR   HIGHLIGHT  INTENSITY
 Title : field headers  WHITE    NONE       HIGH      Enter:
         output fields  BLUE     NONE       LOW       SAVE to save your
         input fields   YELLOW   NONE       LOW           color settings.
 Auto  : autolist area  TURQ     NONE       LOW       DEFAULT to restore
 Source: listing area   WHITE    REVERSE    LOW           default settings.
         prefix area    TURQ     REVERSE    LOW
         suffix area    BLUE     REVERSE    LOW       END to return to
         current line   YELLOW   REVERSE    HIGH          debug session with
         breakpoints    RED      REVERSE    HIGH          current settings
         TOF & EOF line PINK     REVERSE    HIGH          in effect.
         bar graphs     GREEN    REVERSE    HIGH
 Log   : output lines   GREEN    NONE       LOW
         input lines    YELLOW   NONE       HIGH      COLOR and HIGHLIGHT
         line numbers   TURQ     NONE       HIGH      are valid only with
 Search target          TURQ     REVERSE    LOW       color terminals.
 Field headers          TURQ     NONE       HIGH

 Valid COLOR     :  White Yellow Blue Turq Green Pink Red
 Valid HIGHLIGHT :  Uscore Blink Reverse None
 Valid INTENSITY :  High Low
```

Figure 27. Color and Attribute Selection Panel under ISPF Version 2

DESCRIBE Command

The DESCRIBE command displays the data type of scalar variables or arrays, and also supplies dimension information for arrays.

Abbreviation: DE

Syntax

DESCRIBE {[qual.]name | * | (name list)} [PRINT]

[qual.]name
specifies the name of a variable, or array used in the program. The name can be qualified by the name of a program unit.

*****
specifies a list of all the names in the currently qualified program unit. The type of each is displayed.

(name list)
specifies a list of names. Enclose the list in parentheses and separate individual names with commas or blanks.

PRINT
specifies that the output be sent to the print data set instead of the terminal.

Notes:

1. *In your output, dummy arguments are identified in the last column with the word DUMMY.*

2. *Interactive Debug cannot determine the length of character variables that are dummy arguments in an inactive program unit. The length is displayed as CHARACTER*(*). (Note that at ENTRY, the program unit is not yet active.)*

3. *Dimension information is not displayed for arrays that are dummy arguments in an inactive program unit, or that are defined only when entered by some other entry point. (Note that at ENTRY, the program unit is not yet active.)*

4. *The upper bound for the last dimension of an assumed-size array is displayed as an asterisk, and the size is indicated as * ELEMENTS.*

Example

Display the data type information for the variables I and R4, for array CM8ARY, for the dummy array R8ASAR, for the dummy variable WHERE in program unit SUB1, and for dummy array L1AYMN in program unit SUB1.

```
describe (i,r4,cm8ary,r8asar,sub1.where,sub1.l1aymn)
```

The output from this command should look something like this:

```
SUB2.I:                              INTEGER*4
SUB2.R4:                             REAL*4
SUB2.CM8ARY:                         COMPLEX*8
   RANK = 2, SIZE = 21 ELEMENTS
   DIM 1:    EXTENT = 3,  LBOUND = (1),  UBOUND = (3)
   DIM 2:    EXTENT = 7,  LBOUND = (1),  UBOUND = (7)
SUB2.R8ASAR:                         REAL*8              DUMMY
   RANK = 1, SIZE = * ELEMENTS
   DIM 1:    EXTENT = *,  LBOUND = (-3),  UBOUND = (*)
SUB1.WHERE:                          CHARACTER*(*)       DUMMY
SUB1.L1AYMN:                         LOGICAL*1           DUMMY
   RANK = 1; DUMMY ARRAY ARGUMENT OF INACTIVE SUBPROGRAM OR
             ALTERNATE ENTRY POINT
             DIMENSION INFORMATION NOT AVAILABLE
```

For additional examples of the DESCRIBE command, see "Displaying the Data Types of Scalar Variables and Arrays" on page 77.

ENDDEBUG Command

The ENDDEBUG command provides two capabilities:

1. It allows you to discontinue debugging and run the program at full speed. Except for entering limited commands after the program has terminated, no debugging is available after using the ENDDEBUG command.

2. Using the SAMPLE option, you can initiate program sampling to obtain an approximation of relative CPU time.

Abbreviation: None

Syntax

```
ENDDEBUG [SAMPLE[(msecs)][MAXSAMP(n[,STOP])][CALLED ]]
```

SAMPLE [(msecs)]
indicates that sampling is to occur during subsequent execution. msecs is the time interval in milliseconds between sampling interruptions; the default is 10 milliseconds.

MAXSAMP(n[,STOP])
specifies the maximum number of sampling interruptions (n) that can occur. If STOP is specified, the program will be terminated when the specified number of samples is reached. Otherwise, the program will continue execution without sampling interruptions after the specified number of samples is reached. If MAXSAMP is not specified, sampling interruptions will continue until the program terminates. MAXSAMP is valid only if SAMPLE is specified.

CALLED
specifies that each sampling interruption is to be counted for each caller in the save chain. This option may cause an appreciable increase in overhead for the sampling function; however, it makes it much easier to determine when CPU usage is primarily due to called subroutines. CALLED is valid only if SAMPLE is specified.

If CALLED is not specified in the ENDDEBUG command, the CALLED option will not be permitted the ANNOTATE and LISTSAMP commands.

Notes:

1. *All breakpoints and WHEN conditions are removed and the HALT command status is set to OFF.*

2. *TERMIO is set to LIBRARY.*

3. *Timing is turned off for all program units.*

4. *The VS FORTRAN library will begin updating the occurrence count for execution-time errors. All error handling actions will be determined by the settings in the VS FORTRAN error option table.*

5. *The attention exit will no longer allow entry of Interactive Debug commands. The only Interactive Debug command allowed from an attention exit is QUIT, which will terminate the VS FORTRAN program. You will then be prompted for further Interactive Debug commands. Entering QUIT a second time will terminate the debugging session. To resume execution following an attention interrupt, enter a null line.*

6. *At the end of program execution, Interactive Debug will prompt for commands. LISTTIME and LISTFREQ will show information that was current when the ENDDEBUG command was issued. WHERE information cannot be determined after ENDDEBUG is issued.*

7. *Unless the program was compiled with the TEST option, or SAMPLE was specified, execution will proceed at the same speed at which it would run if the DEBUG execution-time option had not been specified.*

   If a program unit was compiled with the TEST option, Interactive Debug will still be called for each VS FORTRAN statement in that program unit. While this activity will not be apparent, there will be a slight increase in the time required to execute your program.

8. *If SAMPLE was specified, program speed will be reduced due to the overhead involved in recording the sampling information at each interrupt. However, this overhead should not be greater than 15%.*

9. *If SAMPLE is specified without CALLED, the ANNOTATE settings (for the source window) are changed to ON. If SAMPLE and CALLED are both specified, the settings are changed to ON and CALLED.*

10. *ENDDEBUG is not permitted after the VS FORTRAN program has terminated or while a read is pending. If issued in an error exit, standard corrective action is taken.*

11. *The BLIP is turned off when sampling is in operation and restored when sampling completes.*

Example 1

Continue executing a program from the current statement to the end of the program with no further debugging activity.

```
enddebug
```

Example 2

End debugging, but continue executing the program with sampling interruptions every 10 milliseconds.

```
enddebug sample
```

Example 3

End debugging, but continue executing the program with sampling interruptions every 20 milliseconds and CALLED counts accumulated.

```
enddebug sample(20) called
```

Example 4

End debugging, but continue executing the program with sampling interruptions occurring every 40 milliseconds. Continue execution of the program without interruptions if the number of interruptions exceeds 10,000.

```
enddebug sample(40) maxsamp(10000)
```

For additional explanation of the ENDDEBUG command, see "Continuing Execution without Further Debugging" on page 93.

ENDFILE Command

The ENDFILE command writes an end-of-file record on a sequentially accessed external file. Its usage is similar to that of the ENDFILE statement in the VS FORTRAN Version 2 language.

Abbreviation: ENDF

Syntax

```
ENDFILE {number | [qual.]integer-variable |
 [qual.]integer-array-element}
```

number
is the number of the I/O unit associated with the sequential file on which the end-of-file record is to be written.

[qual.]integer-variable
is the name of an integer-variable in the VS FORTRAN program. This variable specifies the number of the I/O unit associated with the sequential file on which the end-of-file record is to be written.

[qual.]integer-array-element
is the name of an element of an integer array in the VS FORTRAN program. This element specifies the number of the I/O unit associated with the sequential file on which the end-of-file record is to be written.

Notes:

1. *"number," "integer-variable," or "integer-array-element" must be specified; there is no default number.*
2. *This command may not be issued when I/O is currently active.*
3. *Writing an end-of-file record erases all records that may follow.*
4. *When referring to array elements, the subscripts must use simple arithmetic expressions no more complex than the form "variable plus (or minus) a constant." For example,*

   ```
   ARY(I), ARY(3), ARY(I+3) or ARY(I-3)
   ```

   are valid.

Example

Write an end-of-file record on the sequentially accessed external file associated with I/O unit 8.

```
endfile 8
```

For additional examples of the ENDFILE command, see "Processing External Files" on page 89.

ERROR Command

The ERROR command specifies whether corrective action is to be performed or execution is to be suspended, and whether messages are to be received for execution-time errors. If execution is suspended, you may specify the corrective action to be taken by issuing the FIXUP command. For more information on library error messages, see *VS FORTRAN Version 2: Language and Library Reference.*

Abbreviation: ER

Syntax

```
ERROR {error | error:error | (error list)}

 [MSG | NOMSG] [EXIT | NOEXIT]
```

error
specifies a single error. It is the identification number of the error as defined by the VS FORTRAN Version 2 or VS FORTRAN Version 1 library, or as defined by an auxiliary product.

error:error
specifies a range of error identification numbers to which all specified keywords apply.

(error list)
specifies a list of error numbers or ranges to which all specified keywords apply. Enclose the list in parentheses, and separate entries with commas or blanks.

MSG
specifies that, if an execution-time error occurs among those listed, a full diagnostic message is to be displayed. This is the default action.

NOMSG
specifies that, if an execution-time error occurs among those listed, no diagnostic message is to be displayed.

EXIT
specifies that program execution is to be suspended if any of the listed errors occur. This is the default action.

NOEXIT
specifies that the VS FORTRAN library is to take corrective action and execution is to continue if any of the listed errors occur.

Notes:

1. *The default options at the start of a debugging session are to provide all diagnostic messages (MSG) and to suspend execution of the program if any execution-time error occurs (EXIT). If your program calls ERRSET, this will change the settings for the specified errors to MSG and NOEXIT.*

2. *If the EXIT keyword is in effect for a particular error, you are notified of the location and error number when the error occurs. You can then trace the sequence of control that led to the error location by issuing the WHERE command with the TRBACK or FLOW keyword. If NOEXIT and NOMSG are both in effect, no notification of error location or error number is given. If MSG is in effect, the error number will be contained in the error message.*

3. *If you specify a large range of error numbers, the ERROR command may generate a large number of diagnostic messages. You can use the PURGE command in an attention exit to terminate ERROR if the messages are excessive.*

4. *If NOEXIT has been specified, standard corrective action will be taken unless you have defined user corrective action by calling ERRSET from your VS FORTRAN program. Execution will not be suspended in either case.*

5. *Not all VS FORTRAN library error numbers may be specified by the ERROR command. If you specify one that may not, you will receive the following message:*

   ```
   AFB198I VMOPP : ATTEMPT TO CHANGE UNMODIFIABLE MESSAGE TABLE
    ENTRY, MESSAGE NUMBER 240
   ```

6. *ERROR is not permitted after the VS FORTRAN program has terminated.*

Example 1

You have set a variable to a negative value to test a condition but then realize that the square root of the variable will be taken later. To avoid halting execution when this occurs, you want the library to perform standard corrective action and continue with no notification of the error. The error number for the square root of a negative number is 251.

```
error 251 nomsg noexit
```

Example 2

If any single or double precision arithmetic execution errors (logs, trigonometric functions, exponents) occur, you want Interactive Debug to provide full diagnostics and to also take standard corrective action. The error numbers are 241 through 285.

```
error 241:285 noexit
```

For additional examples of the ERROR command, see "Handling Execution-Time Errors" on page 86.

FIXUP Command

The FIXUP command may be used to specify corrected argument values when execution has been suspended because of errors in a VS FORTRAN library mathematical function. This command causes the function to be evaluated with new arguments and execution to be continued. A FIXUP command with no arguments causes standard corrective action to be taken.

Abbreviation: F

Syntax

```
FIXUP [ARG1(value)] [ARG2(value)]
```

ARG1(value)
specifies the value of the first argument of the function. The value can be a variable, an array element, or a constant.

ARG2(value)
specifies the value of the second argument of the function. The value can be a variable, an array element, or a constant.

Notes:

1. *FIXUP is permitted only when execution is suspended for a library function error.*
2. *If the library function for which FIXUP is to be performed has two arguments, you can specify a single value for either of the two arguments, or you can specify values for both arguments.*
3. *If you issue GO or FIXUP without arguments after an execution-time error, standard corrective action is performed and execution is resumed.*
4. *You cannot change the actual value of the variable in storage by using FIXUP.*
5. *When referring to array elements, the subscripts must use simple arithmetic expressions no more complex than the form "variable plus (or minus) a constant." For example,*

   ```
   A(I), A(3), ARY(I+3), or ARY(I-3)
   ```

   are valid.

Example 1

Your program stops because of an arithmetic error. You want the library to perform standard corrective action and to resume execution.

```
fixup
```

Example 2

Your program attempts to take the square root of a variable that is set to a negative value. Instead of taking standard corrective action, you reset the negative variable to a positive number and resume execution. (The value of the variable in storage is not changed.)

```
fixup arg1(36)
```

For additional examples and a discussion of error correction, see "Handling Execution-Time Errors" on page 86.

GO Command

The GO command resumes program execution.

Abbreviation: None

Syntax

```
GO [[qual.]{number | EXIT}]
```

qual
specifies a program unit name to temporarily override the current qualifier for this command. The qualifier must be the program unit that will be in execution when the GO is executed.

number
specifies the statement identifier of an executable VS FORTRAN statement at which execution is to be resumed. The statement identifier is a statement label, an ISN, or a sequence number in columns 73 through 80. The type of statement number used (ISN or sequence number) is determined by the compiler and the options used. (See "Statement Identifier Conventions" on page 114.) Precede a statement label with a slash. The statement must be an executable statement with a debugging hook.

EXIT
specifies that Interactive Debug will resume execution of the program unit at the exit point that corresponds to the entry point used.

Notes:

1. *The statement identifier or EXIT keyword is optional, and is used if execution is to resume at a specific point outside of the normal execution sequence. GO without an operand causes execution to be resumed at the currently suspended statement.*

2. *If execution has stopped because of an execution-time error, resuming it with GO produces standard corrective action to fix the error. (GO with an operand is not allowed in an error exit.)*

3. *Execution of the GO command is subject to the same language restrictions as the GOTO statement in VS FORTRAN. Branches into DO-loops and inner DO-loops (except from the extended range of the DO loop (LANGLVL(66) only)) will produce unpredictable results.*

4. *The number or EXIT operand is not allowed with the GO command:*

   - *In an error exit*
   - *When execution is suspended at the ENTRY point of a program unit*
   - *When execution is suspended for output*
   - *When execution is suspended during a terminal read*

5. *It is not advisable to specify a statement label or statement identifier when executing in a program unit that has been compiled with an optimization level higher than zero. (This is because optimized code is highly dependent on register contents.) Interactive Debug allows the operand, but will warn you (except in batch mode) that the program unit is optimized and ask if you wish to continue. If you type* YES, *the GO will be executed.*

6. *GO is not permitted after the VS FORTRAN program has terminated. It is also not permitted if execution is suspended during a terminal read.*

Example 1

Your program suspends execution at a breakpoint for debugging, and you now want to resume execution at the currently suspended statement.

```
go
```

Example 2

Your program suspends execution at a breakpoint for debugging, and you now want to skip ahead and resume execution at sequence number 410.

```
go 410
```

Example 3

Your program suspends execution at a breakpoint for debugging, and you have changed the qualification to display variables in another program unit. You now want to skip ahead and resume execution at the exit point of the program unit currently executing, which is SUB1.

```
go sub1.exit
```

Example 4

You know that program unit BUGGY produces incorrect results past the statement labeled 100. For now, you want to set RESULT to the value A, and exit whenever statement 100 is reached.

```
at buggy./100 (set result = a % go buggy.exit) nonotify
```

HALT Command

The HALT command causes execution to be suspended for every statement of a given class, or at a specific point in a command list. The classes of statements are: at the start of every statement, or after every branch, or at entry to and exit from a debuggable routine.

For more information on use of the HALT command under ISPF Version 2, see "Using Interactive Debug Features under ISPF Version 2" on page 24.

Abbreviation: None

Syntax

HALT [OFF | STMT | GOTO | ENTRY | **IMMED**]

OFF
specifies that the HALT setting in effect is to be terminated. This is the setting when Interactive Debug execution begins.

STMT
indicates that execution is to be suspended before every executable statement that has a debugging hook.

GOTO
indicates that execution is to be suspended whenever two consecutively executed debugging hooks are not on consecutively stored statements. This could occur for several reasons, including a GOTO, a DO group, a CALL, or an IF statement. It also might occur if one or more statements have been collapsed by optimization or vectorization, or because of hook restrictions specified for ranges in AFFON, or when debug packets occur in the code.

ENTRY
specifies that execution is to be suspended whenever any debuggable program unit is entered or exited. This could be as a result of a subroutine or function call or return from a subroutine or function.

HALT ENTRY causes suspension of execution at the same points as AT EXIT or AT ENTRY. At an entry breakpoint, the program unit is not yet active. Thus, dummy arguments or variables in a dynamic common cannot be accessed, and GO with an operand is not permitted.

IMMED
indicates that execution of an AT command list should be suspended immediately and a prompt should be issued. This is the default action when HALT is issued with no operand. In an attention exit, this will terminate the current AT command list if one is executing.

Notes:

1. *You can issue a HALT IMMED command to terminate a command list. You can also use HALT IMMED in an attention exit to terminate command loops. For example:* `AT 5 (GO 5)`.

2. *If a statement at which execution would normally be suspended by the current HALT setting has an AT breakpoint with a command list that causes execution to resume, the HALT setting will not suspend that statement.*

   For example, if HALT STMT is in effect but you have issued the command `AT 5 (SET A=10%GO)`, *execution will not be suspended at statement 5.*

3. *Any statement for which execution would be suspended by HALT GOTO will also have execution suspended by HALT STMT.*

   Any statement for which execution would be suspended by HALT ENTRY will also have execution suspended by HALT GOTO and HALT STMT.

4. *HALT is not permitted after the VS FORTRAN program has terminated.*

5. *To see the current HALT setting, type* `LISTBRKS`.

Example 1

Suspend execution at entry to or exit from all debuggable units.

```
halt entry
```

Example 2

Suspend execution prior to executing each statement with a debugging hook in the program.

```
halt stmt
```

Example 3

At statement identifier 10, halt execution if A is greater than B; otherwise, continue executing with no notification of the breakpoint.

```
at 10 (if (a .gt. b) halt%go) nonotify
```

For additional examples of the HALT command, see "Controlling Program Execution" on page 74.

HELP Command (ISPF Version)

The HELP command provides information about the description, format, and keywords of all Interactive Debug commands as well as a task-oriented tutorial. See Appendix A, "Using the Interactive Debug HELP Facility" on page 219 for a description of HELP capabilities, procedures, and examples. The following form of the command is intended for use in an ISPF environment.

Abbreviation: H (ISPF Version 2)

Syntax

```
HELP [command ]
```

command
specifies the name of a command, or one of the keywords TASK or MENU. If no command is specified, a HELP menu is displayed for further selection. You cannot use a command abbreviation.

Notes:

1. *If HELP is issued with no operand after an error has occurred in an Interactive Debug command, the HELP panel for that command is shown. Otherwise, the HELP main menu is displayed for you to select a specific help panel.*

2. *The HELP command cannot be used as part of an AT command list or with an IF command.*

3. *HELP TASK displays a menu of debugging tasks. From this menu, you may request further information for a specific task.*

4. *HELP or HELP MENU displays a menu of all commands available in the Interactive Debug HELP facility. From this menu, you may request further information for a specific command in the menu.*

Example 1

Display a list of all available HELP command topics.

```
help
```

Example 2

Display HELP information for the LIST command.

```
help list
```

HELP Command (CMS Version)

The HELP command provides information about the description, format, and keywords of all Interactive Debug commands as well as a task-oriented tutorial. See Appendix A, "Using the Interactive Debug HELP Facility" on page 219 for a description of HELP capabilities, procedures, and examples. The following form of the command is intended for use in a CMS line mode environment.

Abbreviation: H

Syntax

HELP [command [(ALL | (DESC | (PARM | (FORM]]

command
specifies the name of a command, or one of the keywords TASK or MENU. If no command is specified, no other option may be specified; a HELP menu is displayed for further selection. You cannot specify a command abbreviation.

ALL
requests information about keywords and syntax of the named command. This is the default action.

DESC
requests a description of the named command.

PARM
requests a description of the keywords of the named command.

FORM
requests a description of the syntax of the named command.

Notes:

1. *The HELP command cannot be used as part of an AT command list, with an IF command, or in batch mode.*

2. *In CMS line mode, HELP TASK displays a menu of debugging tasks. From this menu, you may request further information for a specific task.*

3. *In CMS line mode, HELP or HELP MENU displays a menu of all commands available in the Interactive Debug HELP facility. From this menu, you may request further information for a specific command in the menu.*

4. *In CMS line mode, if HELP is issued without any operands after an error has occurred in an Interactive Debug command, the HELP panel for that command will appear.*

Example 1

Display a list of all available HELP command topics.

```
help
```

Example 2

Display all HELP information for the IF command.

```
help if
```

Example 3

Display HELP information for all ERROR command keywords.

```
help error (parm
```

HELP Command (TSO Version)

The HELP command provides information about the function, syntax, and keywords of all Interactive Debug commands as well as a task-oriented tutorial. See on Appendix A, "Using the Interactive Debug HELP Facility" on page 219 for a description of HELP capabilities, procedures, and examples. The following form of the command is intended for use in a TSO line mode environment.

Abbreviation: H

Syntax

```
HELP [command][ALL][FUNCTION][SYNTAX]

 [OPERANDS [(keyword list)]]
```

command
specifies the name of a command, or one of the keywords TASK or IADMENU. If no command is specified, no other option may be specified; a list of commands is displayed for further selection. You cannot specify a command abbreviation.

ALL
requests information about the function, syntax, and operands of the named command. This is the default action.

FUNCTION
requests information about the function of the named command.

SYNTAX
requests information about the syntax of the named command.

OPERANDS [(keyword list)]
requests information about keywords for the named command. Enclose the list in parentheses and separate the keywords in the list with commas or blanks. If the list is omitted, all keywords, operands, notes, and examples for the named command are explained.

Notes:

1. *The HELP command cannot be used as part of an AT command list, with an IF command, or in batch mode.*

2. *In TSO line mode, HELP TASK displays a menu of debugging tasks. You may then issue another HELP command to request further information for a specific task.*

3. *In TSO line mode, HELP IADMENU displays a menu of all topics available in the Interactive Debug HELP facility. You may then issue another HELP command for a specific item in the menu.*

4. *In TSO line mode, if HELP is issued without any operands after an error has occurred in an Interactive Debug command, the HELP panel for that command appears.*

5. *In line mode, you can request that notes and examples be shown by specifying NOTES or EXAMPLES as a keyword with OPERANDS.*

Example 1

Display a list of all available HELP topics.

```
help
```

Example 2

Display all HELP information for the IF command.

```
help if
```

Example 3

Display ERROR command syntax and HELP information for the EXIT keyword.

```
help error syntax operands(exit)
```

Example 4

Display the notes and examples for the GO command in addition to the syntax.

```
help go syntax operands (notes, examples)
```

IF Command

The IF command is used, usually within an AT command list, to test a relational or a logical condition when the specified breakpoint is reached. If the condition is true, the command specified within the IF command is executed.

Abbreviation: None

Syntax

IF (condition) command

(condition)
is the condition to be tested. It can be either a relational or a logical condition.

command
is a single Interactive Debug command that is executed only if the specified condition is true.

Relational condition: A relational condition is a signed or unsigned variable or array element or constant, followed by a *relational operator*, followed by another signed or unsigned variable or array element or constant.

There are six relational operators that can be used to test relational conditions.

= or .EQ.

¬= or .NE.

> or .GT.

< or .LT.

>= or .GE.

<= or .LE.

Logical condition: A logical condition is a logical variable or a logical array element, optionally preceded by the negation operator (¬ or .NOT.). No other operators are permitted.

Notes:

1. *The following is an example of the relational condition form of the IF command:*

   ```
   IF (A .GT. B) HALT
   ```

   With the relational condition form of the IF command, the following is true:

   - *When either variable or constant is a logical or character type, both must be of that same type, and no sign is permitted preceding either variable or constant.*

- *When either variable or constant is a logical or complex type, only the relational operators .EQ. and .NE. (or = and ¬=) may be used.*

2. *Substring notation is permitted for string variables.*

3. *For array elements, subscripts may have values beyond the range of the corresponding array dimensions. A warning message will be issued except in the special case where the last dimension is "1" or "*" and only the last subscript is out of range.*

4. *When referring to array elements, the subscripts must use simple arithmetic expressions no more complex than the form "variable plus (or minus) a constant." For example,*

   ```
   ARY(I), ARY(3), ARY(I+3) or ARY(I-3)
   ```

   are valid forms.

5. *You cannot reference variables that are not currently defined, such as dummy arguments in an inactive subroutine.*

6. *Logical variables must have been set to VS FORTRAN logical constants for condition testing to produce predictable results.*

7. *Variables in different program units can be referenced by qualifying the variable names with program unit names; for example:*

   ```
   IF (MAIN.A .LT. SUB1.B) SET SUB1.B = MAIN.A
   ```

8. *The command specified with IF cannot be HELP, QUALIFY, or FIXUP. You also cannot specify any of the full screen display commands.*

9. *When character variables or constants of unequal length are compared, the shorter is considered to be extended with blanks during the comparison.*

10. *IF is not permitted after the VS FORTRAN program has terminated.*

Example 1

At the statement labeled 100, determine whether logical variable OVER is true, and if it is true, reinitialize counter I and resume processing; if OVER is false, continue processing without resetting I.

```
at /100 (if (over) set i=0%go)
```

Example 2

At statement number 10, test if variable A in subroutine SUB is zero; if it is, suspend execution; if A is not zero, continue processing.

```
at 10 (if (sub.a = 0.0) halt%go)
```

Example 3

At statement 5, test the following set of logical conditions; if either A or B is true, go to the statement labeled 10 and continue processing; if neither is true, continue processing at the next executable statement.

```
at 5 (if (a) go /10%if (b) go /10 %go)
```

Example 4

At statement 7, if the first five characters of the variable SOLAR are ABCDE, set counter `I` to 2

```
at 7 (if (solar(1:5)='ABCDE') set I=2)
```

LIST Command

The LIST command displays the values of specified scalar variables, arrays, array elements, or string constants at the terminal or in a print data set. Values can be displayed in a variety of formats. The specified or implied qualifier is shown with all variable or array names and the array elements may be displayed outside the defined dimensions. For more information on how to use LIST under ISPF Version 2, see "Using Interactive Debug Features under ISPF Version 2" on page 24. See also "Displaying Formatted Variable and Array Values" on page 85.

Abbreviation: L

Syntax

```
LIST {[qual.]name[:[qual.]name] | * | 'string' |
  number | (specification list)}
  [PRINT] [FORMAT [(code)] | DUMP [(code)]]
```

[qual.]name
specifies the name of a variable, array, or array element used in the program. If a qualifier is specified, it overrides the current qualifier for the specified name.

[qual.]name:[qual.]name
specifies a range of variable, array, or array element names used in the program. If a qualifier is specified, it overrides the current qualifier for the specified name.

LIST displays all storage locations between the two variables. Unless FORMAT or DUMP is specified, the format of the displayed variables is the same as the type of the first variable.

*****
specifies that a list of all the names in the currently qualified program unit is desired. Unless FORMAT or DUMP is specified, each is displayed according to its own type.

'string'
specifies a character string to be displayed as a remark. You can use this operand to help identify breakpoints.

number
specifies an integer or real numeric constant to be displayed as a remark. This function is useful for converting numbers, in conjunction with the FORMAT option.

specification list
specifies a list of individual specifications. Enclose the list in parentheses and separate entries with commas or blanks.

PRINT
specifies that output is to be sent to the print data set instead of to the terminal.

FORMAT [(code)] or DUMP [(code)]
specifies a particular data format:

- FORMAT displays the names listed and their values in the specified format.
- DUMP displays the address in storage of the names listed and their values in the specified format.
- (code) specifies the format or dump code for the names to be listed. The default format code is X. The default dump code is Z.
- FORMAT and DUMP are mutually exclusive.

  FORMAT and DUMP codes for the LIST command are the same as for the AUTOLIST command, and are shown again in Figure 28 on page 160.

Code	Output
L1	Logical*1
L4	Logical*4
I2	Integer*2
I4	Integer*4
R4	Real*4
R8	Real*8
R16	Real*16
C8	Complex*8
C16	Complex*16
C32	Complex*32
L	Logical with size closest to internal data size
I	Integer with size closest to internal data size
R	Real with size closest to internal data size
C	Complex with size closest to internal data size
X[nnn]	Hexadecimal with nnn bytes per data item (default to internal data size)
Z[nnn]	Hexadecimal with nnn bytes per data item (default to Z4)
A[nnn]	Character with nnn bytes per data item (default to internal data size)
H[nnn]	Character with nnn bytes per data item (default to continuous full line output)

Figure 28. DUMP and FORMAT Codes for the LIST Command

Notes:

1. *When you request an individual name or list of names, the default formatting of values is determined by the type of each name being displayed. When you request a range of names, the formatting of the values is determined by the format of the first name in the range. You may, however, specify a different format using the FORMAT or DUMP keyword. The locations of the listed names are identified in the output only if DUMP is specified.*

2. *VS FORTRAN defines storage layout only for arrays, variables in a common block (defined in a COMMON statement), and variables in equivalence groups (defined in an EQUIVALENCE statement). The relative positions of any other names in storage cannot be predicted. Names that you may expect to be adjacent in storage may be widely separated by other data. Therefore, a range specification for names other than array, equivalence, or common variables may produce unexpected results.*

3. *The length specification in a FORMAT or DUMP code may be entered with 1 through 3 digits. Thus, I4, I04, and I004 are equivalent.*

4. *A length specification of 0 in character and hexadecimal FORMAT and DUMP codes (for example, A0 or Z0) causes the data to be displayed as a continuous string, rather than split into pieces of some specified length.*

5. *If a FORMAT or DUMP code with no length specification is given for a range of variables or array elements, each variable or array element is displayed separately in the specified format. However, if a length specification is given, Interactive Debug will consider the entire storage area occupied by all the range of variables or array elements, or occupied by the entire array, as if it were broken into pieces, each with a length equal to that specified in the DUMP or FORMAT code, and will display each piece according to the specified format. For example, if PRIMES is a 2 x 3 array of INTEGER*4 values, then:*

   ```
   list primes format(x)
   ```

   will cause a display of 6 values, each corresponding to an element of the array. However:

   ```
   list primes format(x2)
   ```

   will cause a display of 12 values, each displaying the contents of successive 2-byte storage areas within the array.

6. *The DUMP option is not permitted with constant operands, including strings; using it will produce an error message.*

7. *An assumed size array cannot be listed by just specifying the array name; the specific element or range of elements must be specified. (An assumed size array is an array with the last upper bound declarator specified as an asterisk (*).) This restriction does not apply to arrays whose last dimension is "1," even though such arrays are otherwise treated as assumed size arrays. However, only the elements whose last subscript is "1" will be displayed if no subscripts are specified.*

8. *For array elements, subscripts may have values beyond the range of the corresponding array dimensions. A warning message will be issued except in the special case where the last dimension is "1" or "*" and only the last subscript is out of range.*

9. *Array subscripts must consist of simple arithmetic expressions no more complex than the form "variable plus (or minus) a constant." For example,*

   ```
   list ARY(I), list ARY(3), list ARY(I + 3) or list ARY(I - 3)
   ```

   are valid.

10. *Dummy arguments can only be displayed when the program unit in which they are defined is active. If not, an error message is issued. Results are unpredictable if you display a dummy argument that is not defined at the entry point called. Note that a program unit is not yet active when suspended at entry.*

11. *Variables in dynamic commons can only be displayed if the program unit used to qualify the variable has been activated at least once. (If not, an error message may be issued. However, if a variable has a large displacement in its dynamic common, Interactive Debug cannot detect that it is not initialized.)*

12. *Although a quoted string can be used as an operand on the LIST command, you cannot point the cursor at a quoted string in the source listing window (when using LIST as a cursor-oriented command).*

Example 1

Display at the terminal the value of the variable named NCOUNT.

```
list ncount
```

Example 2

Obtain a hexadecimal dump (FORMAT(X)) showing values of array variables A(1,1) through A(7,10). Have the display sent to the print data set.

```
list a(1,1):a(7,10) print format
```

Example 3

List the decimal number 12345 in hex.

```
list 12345 f(x)
```

Example 4

Display an entire array (CHARAY) containing a series of 30-character alphabetic strings so that each character string is separated from the others. (If the array is declared in the program to be a CHARACTER*30 array, the elements of the array will be separated from each other when the array is listed.)

```
list charay format(a30)
```

Example 5

Display the value of the variable named CTR in subroutine SUB1.

```
list sub1.ctr
```

Example 6

Display the message "inside loop" whenever ISN 100 is executed.

```
at 100 (list 'inside loop'%go)
```

Example 7

Display variable LONG__NAME__VAR in program unit SUB1, and variable SHORT in program unit SUB2. This illustrates the LIST format in support of long (31 character) names. Note that long names cause a line break to allow alignment of the "="s without excessive horizontal spread.

```
list (sub1.long_name_var,sub2.short)
SUB1.LONG_NAME_VAR
                =           10
SUB2.SHORT      =           25
```

Example 8

Display values of the second through sixth characters in the character variable PORT

```
list PORT(2:6)
```

For additional examples of the LIST command, see "Displaying Formatted Variable and Array Values" on page 85.

LISTBRKS Command

The LISTBRKS command provides the following information:

- All breakpoints that are currently set, including entry, exit, and statement breakpoints
- All WHEN conditions (both on and off) that are currently defined, and the condition being tested
- The current HALT status (OFF, STMT, GOTO, or ENTRY)

Abbreviation: LB

Syntax

```
LISTBRKS [PRINT]
```

PRINT
specifies that output is to be sent to the print data set instead of to the terminal.

Note: LISTBRKS lists breakpoints and WHEN conditions for all program units.

Example

The following is a sample of the output produced by LISTBRKS:

```
CURRENT BREAKPOINTS:
  MAIN.15/30
  SUB.ENTRY
  SUB.8/10
CURRENT WHEN CONDITIONS:
  ABCD ON (SUB.X > 5)
  EFGH ON (CH(1:2)='AB')
CURRENT HALT STATUS:  OFF
```

For additional examples of the LISTBRKS command, see "Referring to Statements or Variables in Other Program Units" on page 70.

LISTFREQ Command

The LISTFREQ command lists the number of times statements in the currently qualified program unit have been executed. This command can also be used to list the statements that have not been executed.

Abbreviation: LF

Syntax

```
LISTFREQ [[qual.]{number[:[qual.]number] | ENTRY | EXIT} |
 (number/ENTRY/EXIT list)]
 [ZEROFREQ] [PRINT]
```

qual
specifies a program unit name prefix to temporarily override the current qualifier for the prefixed operand only.

number
specifies the statement label, ISN, or sequence number of an executable VS FORTRAN statement whose execution counts are to be listed or checked for zero. Precede a statement label with a slash to distinguish it from an ISN or sequence number.

number:[qual.]number
specifies a range of statement labels or statement numbers (ISNs or sequence numbers) whose execution counts are to be listed or checked for zero. Statement labels can be combined with ISNs or sequence numbers in the range specification, but the first and last must be executable statements. Precede each statement label with a slash.

Statement identifiers can be qualified with a program unit name. The default program unit for the first identifier is the current qualifier. The default program unit for the second identifier is the program unit specified or defaulted for the first identifier. Both identifiers must have the same program unit in effect.

ENTRY
specifies entry points. The frequency of entry into the program unit is specified.

EXIT
specifies exit points. The frequency of exit from the program unit is specified.

(number/ENTRY/EXIT list)
requests a list of statement labels, ISNs or sequence numbers, entry points, exit points, and ranges. (Note that ENTRY and EXIT are not permitted in a range.) The frequency for each specified statement is listed. Enclose the list in parentheses, with individual entries separated by commas or blanks. Precede each statement label with a slash.

ZEROFREQ
requests a list of statements that have not been executed. All statements may be tested, or specific statements may be specified using the options discussed above.

PRINT
requests that the output go to a print data set instead of to the terminal.

Notes:

1. *If no operand is specified, the counts are displayed for ENTRY, EXIT, and all executable statements with debugging hooks in the currently qualified program unit.*

2. *If LISTFREQ is issued after an ENDDEBUG command, the execution counts displayed are those that existed when ENDDEBUG was issued.*

3. *Execution counts of unhooked or collapsed statements are not displayed. Instead, you will see the phrase "NO HOOK" or "COLLAPSED STMT."*

4. *Before a RENT program unit is first entered, all statements are considered to have no hook. Statements in a reentrant program unit that are excluded in the AFFON file will show "COLLAPSED STMT" on the LISTFREQ display.*

5. *After ENDDEBUG is issued, LISTFREQ displays counts for all noncollapsed statements.*

6. *Statements in a debug packet will be treated as collapsed in VS FORTRAN programs compiled prior to Version 1 Release 4.0. If the program is compiled with VS FORTRAN Version 1 Release 4.0 or later, the debug statements are inserted directly into the code and LISTFREQ will show duplicate statements in addition to the DEBUG packet code.*

7. *If you request ZEROFREQ, Interactive Debug displays only hooked statements that have* **not** *been executed.*

Example 1

On the print data set, list how many times each executable statement in the currently qualified program unit has been executed.

```
listfreq print
```

Your output might look something like this:

```
STATEMENT              FREQUENCY
MAIN.ENTRY              NO HOOK
MAIN.EXIT               NO HOOK
MAIN.6                  1
MAIN.7                  COLLAPSED STMT
MAIN.8                  COLLAPSED STMT
MAIN.9                  NO HOOK
MAIN.10                 3
MAIN.11                 3
MAIN.12                 NO HOOK
MAIN.13/10014           NO HOOK
```

Example 2

List the statements that have **not** been executed in the currently qualified program unit.

```
listfreq zerofreq
```

Example 3

List how many times some specific statements have been executed in the currently qualified program unit.

```
listfreq (10:/80,300,/95,/105,ENTRY)

listfreq (20:130 ENTRY 250 /1000)
```

Example 4

List how many times the executable statements 12 through 15 in subroutine SUB1 have been executed.

```
listfreq sub1.12:sub1.15
```

For additional examples of the LISTFREQ command, see "Determining Statement Execution Frequency" on page 78.

LISTINGS Command

The LISTINGS command displays the listings data set specification panel under ISPF version 2. This provides a command equivalent to typing "?" in the Q: field.

Abbreviation: None

Syntax

```
LISTINGS
```

Notes:

1. *LISTINGS cannot be issued in a command list, as the command portion of an IF command, in an attention exit, or a restart file.*

2. *LISTINGS will operate as usual with a parameter list, but the panel will indicate the message "PARAMETERS IGNORED."*

Example

When you issue the LISTINGS command, a Listings Data Set Specification Panel, similar to Figure 29, will be displayed.

```
                    VS FORTRAN INTERACTIVE DEBUG                    ROW 1 OF 4
COMMAND ===>                                                  SCROLL ===> HALF

       PROGRAM UNIT NAME            CMS FILE ID       SOURCE
MAIN_PROGRAM_____________________ FORTPROG LISTING  *_   NO     FILE NOT FOUND
SUB1_____________________________ SUB1     LISTING  *_   YES
SUB2_____________________________ SUB2     LISTING  *_   NO     FILE NOT USABLE
SUBROUTINE3______________________ SUB1     LISTING  *_   NO     PROGRAM NOT FOUND
****************************** BOTTOM OF DATA  ********************************
```

Figure 29. The Listings Data Set Specification Panel in CMS

The column labelled "PROGRAM UNIT NAME" identifies the debuggable VS FORTRAN program units. The second column, "CMS FILE ID," indicates the names of files where the listings are to be found. In the column labelled "SOURCE," YES indicates that the listing will be displayed in the Source Listing Window.

In the last column, the CMS message FILE NOT FOUND indicates that the specified file was not found on any of your accessible disks. The equivalent TSO message is DATA SET NOT FOUND. The message PROGRAM NOT FOUND indicates that the listing file was found, but did not contain the named program (in this case, SUBROUTINE3). The FILE NOT USABLE message indicates that the

data set was found but cannot be used as a listing in the source window. Under TSO, the message is DATA SET NOT USABLE.

After filling in the listings data set specification panel, you return to the execution panel by entering END, usually PF key 3.

LISTSAMP Command

The LISTSAMP command lists sampling counts by statement or by program unit. Percentage of program unit samples and percentage of total number of samples are also listed, along with a bar chart of the sampling counts.

Abbreviation: None

Format 1

Syntax for Listing Sampling Counts by Statement

```
LISTSAMP {[qual.]number[:[qual.]number]
  | [qual.]ENTRY | [qual.]* | (specification list) | *.*}
 [DIRECT][CALLED][ALL] [TOP[(n)]][PRINT]
```

Format 2

Syntax for Listing Sampling Counts by Program Unit

```
LISTSAMP {unit name | (unitname list) | *} SUMMARY
 [DIRECT][CALLED][ALL] [TOP[(n)]][PRINT]
```

[qual.]number
is the statement label, ISN, or sequence number of an executable statement whose sampling information is to be listed. Qualification is optional. A statement label must be prefixed with a slash (/).

[qual.]number:[qual.]number
specifies a range of statements in the program whose sampling information is to be listed. Qualification is optional. If the second qualifier is specified, it must be the same as that specified for the first qualifier.

[qual.]ENTRY
indicates that the sampling count for the entry and exit code of the specified or currently qualified program unit is to be listed. There is only one sampling count to cover both entry and exit code.

[qual.]*
indicates that all statements in the specified or currently qualified program unit are to be included.

specification list
specifies a list of individual specifications. Enclose the list in parentheses and separate entries using commas or blanks.

.
includes all statements in all programming units.

unitname
specifies the name of a program unit whose sampling summary is to be listed. This must be a VS FORTRAN unit compiled with SDUMP.

unitname list
specifies a list of program unit names separated by commas or blanks.

*****
indicates that all program units, debuggable or not, are to be included. In addition, there are two special names that are reported when "*" is specified:

*LIBRARY shows the sampling count accumulated for all VS FORTRAN Library modules other than the mathematical functions and the Error Monitor. This includes lower-level calls to system services.

*UNKNOWN shows the count of sampling interrupts that could not be assigned to any program unit.

SUMMARY
indicates that sampling counts are to be summarized by program unit. This keyword is only allowed in format 2.

DIRECT
indicates that interruptions occurring in the code are to be included in the sampling counts for that code. DIRECT is the default.

CALLED
indicates that interruptions occurring in lower-level routines are also to be included in the sampling counts of the code being sampled. This option is valid only when sampling is initiated with the CALLED option.

ALL
indicates that sampling counts are to be the sum of the DIRECT and CALLED counts.

TOP(n)
indicates that only the number of statements specified by n (or programming units, if the SUMMARY option was specified) having the highest counts are to be listed. The output is sorted in descending order by count. The default for n is "1"; the maximum value is "9999."

If TOP is not specified, the output is in the order shown in the specification list, and within that by statement table order.

PRINT
indicates that output is to be written to the print data set instead of to the terminal.

Notes:

1. *The LISTSAMP command is valid only when sampling has been performed.*

2. *Qualifiers and unit names are restricted to VS FORTRAN program units compiled with SDUMP.*

3. *Non-FORTRAN program units (and units compiled with NOSDUMP) are identified by the entry ID located using the value of GPR 15 saved in the savearea. Programs that do not follow MVS standards for entry identifiers will not be correctly identified. Note that the entire ID string is shown, up to 31 characters. This often includes blanks and additional information such as date and time of compilation.*

4. *Sampling counts for nondebuggable program units cannot be requested by name; however, the "*" operand with the SUMMARY option will show sampling counts for both debuggable and nondebuggable program units, including VS FORTRAN library counts.*

Example 1

Display a summary of the sampling counts for all program units

```
listsamp * summary

PROGRAM SAMPLING INTERVAL WAS 20 MS; TOTAL NUMBER OF SAMPLES
        WAS 9745.
DIRECT SAMPLES:
 PROGRAM UNIT     SAMPLES %TOTAL
 MAIN                 613   6.35 *
 SUB1                  15   0.15
 SUB2                5763  59.71 ************
 SUB3                1882  19.31 ***
 S#IN                1251  12.83 ***
 *LIBRARY             128   1.31
 *UNKNOWN              93   1.01
```

Example 2

Display a summary of the sampling counts for SUB1 and SUB2

```
listsamp (sub1,sub2) summary

PROGRAM SAMPLING INTERVAL WAS 20 MS; TOTAL NUMBER OF SAMPLES
        WAS 9745.
DIRECT SAMPLES:
 PROGRAM                SAMPLES %TOTAL
 SUB1                        15   0.15
 SUB2                      5763  59.71 ************
```

Example 3

Display the sampling counts for a range of statements in SUB2. (Sampling counts will include interruptions which occurred in the code of a statement as well as in any lower-level routines called by the statement.)

```
listsamp sub2.10:20 all

PROGRAM SAMPLING INTERVAL WAS 20 MS; TOTAL NUMBER OF SAMPLES
        WAS 9745.
SUM OF DIRECT AND CALLED SAMPLES:
 STATEMENT                SAMPLES  %UNIT  %TOTAL
 SUB2.10/10                  1142  73.15   12.53 ***************
 SUB2.11/12                   231  15.42    3.02 ***
 SUB2.13                       12   1.22    0.15
 SUB2.15                COLLAPSED
 SUB2.16                       22   2.38    0.32
 SUB2.18                       14   1.24    0.16
 SUB2.20                       46   5.17    0.77 *
```

Example 4

Display the four highest counts in SUB2. (Sampling counts will include interruptions which occurred in the code of a statement as well as in any lower-level routines called by the statement.)

```
listsamp sub2.* top(4) all

PROGRAM SAMPLING INTERVAL WAS 20 MS; TOTAL NUMBER OF SAMPLES
        WAS 9745.
SUM OF DIRECT AND CALLED SAMPLES:
 STATEMENT                SAMPLES  %UNIT  %TOTAL
 SUB2.10/920                 1142  73.15   12.53 ***************
 SUB2.11/930                  231  15.42    3.02 ***
 SUB2.20                       46   5.17    0.77 *
 SUB2.ENTRY/EXIT               42   4.92    0.72 *
```

LISTSUBS Command

The LISTSUBS command displays a list of all VS FORTRAN program units compiled with SDUMP, including those not listed in AFFON, with the following information for each:

- Compiler level used to produce the object code (if it can be determined)
- Optimization level
- Vectorization level
- Hook existence
- Timing status
- Load status for units compiled with RENT

Abbreviation: LS

Syntax

```
LISTSUBS [PRINT]
```

PRINT
specifies that output is to be sent to the print data set instead of to the terminal.

Example

The following is a sample of the output produced by LISTSUBS:

```
PROGRAM UNIT     COMPILER     OPT  HOOKED   TIMING
MAINLINE        VSF  2.2.0    V2    YES       ON
SUBBUILD        VSF  1.4.0     3    NO        OFF  RENT NOT LOADED
SUBDOWN         VSF  (TEST)    0    YES       OFF
SUBREFIT        VSF  1.3.1     1    NO        ON
```

In the sample output above, `VSF (TEST)` means that the program unit was compiled prior to VS FORTRAN Version 1 Release 4.0, and the TEST option was specified. In this case, it is not possible to determine the VS FORTRAN release level.

For nonvectorized programs, the "OPT" column displays the level of optimization: 0, 1, 2, or 3. For vectorized programs, the "OPT" column in the output displays one of the following values:

V1 VECTOR (LEVEL(1))

V2 VECTOR (LEVEL(2))

If the program unit has been vectorized, the optimization level is always 3, and is not shown.

YES means that hooks are installed at entry and exit points and possibly at some or all statement boundaries as well. The hook settings are controlled by the AFFON file. NO in the "HOOKED" column indicates that no hooks are installed in the program unit.

ON in the "TIMING" column indicates that timing has been activated for the program unit. The TIMER command is used to set timing ON or OFF.

The possible load status indications for RENT program units are:

RENT NOT LOADED The program unit has not yet been called, and has not been located (although it may actually be in storage).

RENT IN USER AREA The program unit has been called and is in user-owned storage.

RENT IN PROT AREA The program unit has been called and is in protected storage.

For additional examples of the LISTSUBS command, see "Displaying Information about Debuggable Program Units" on page 69.

LISTTIME Command

The LISTTIME command displays timings for all program units for which timing is active or was previously active. The TIMER command activates timing for a unit.

See the TIMER command for more information on activating timing.

Abbreviation: LT

Syntax

```
LISTTIME [PRINT]
```

PRINT
specifies that output is to be sent to the print data set instead of to the terminal.

Notes:

1. *The timing information provided by Interactive Debug may include overhead caused by the debugging hooks in your program. Thus, the timing information is not an accurate representation of the time it takes to execute* **without** *Interactive Debug.*

2. *Timing for very small subroutines may be erratic.*

3. *Timing is measured separately for each entry point in the program unit.*

4. *If LISTTIME is issued after an ENDDEBUG command, the times and activation counts are those that existed when ENDDEBUG was issued.*

Example

The following is a sample of the output produced by LISTTIME:

```
ENTRY POINT          TASK TIME (MIC)    PERCENT    INVOCATIONS
MAIN                           31680      48.48              1
SUBA1                           4128       6.32              2
SUBA2                          28544      43.68             15
SUBB                             942       1.52              1
```

For additional examples of the LISTTIME command, see "Displaying Timing Information" on page 82.

MOVECURS Command

The MOVECURS command is used under ISPF Version 2 to move the cursor between the source window and the command line. It can be used only in full screen mode under ISPF Version 2, and is invalid if used in any other environment. It cannot be issued in a command list, an IF command, an attention exit, or a restart file.

See "Using Interactive Debug Features under ISPF Version 2" on page 24 for more information.

Abbreviation: MC

Syntax

MOVECURS

Notes:

1. *If the cursor is anywhere outside the command line when you give the MOVECURS command, the cursor is returned to the command line. If the cursor is already on the command line, it will be positioned at the most recent position in the source window (if the position is known and available). Otherwise, it is positioned at the beginning of the source window.*

2. *You may find it convenient to redefine the ISPF "CURSOR" PF key to issue the MOVECURS command. To do this, enter the ISPF command* KEYS *on the ISPF command line. You will then see a list of all current PF key assignments. You can change the CURSOR key (usually PF 12) to MOVECURS by typing* MOVECURS *next to the appropriate PF key number and pressing ENTER.*

NEXT Command

The NEXT command suspends program execution at the next statement, entry, or exit with a debugging hook. Because some statements may not have been included in the AFFON list or may have been collapsed, execution need not necessarily be suspended at the next statement to be executed.

Abbreviation: N

Syntax

```
NEXT
```

Notes:

1. *There are no operands for the NEXT command.*
2. *You are notified of the point where execution is suspended because of NEXT.*
3. *NEXT suspension is not a breakpoint. It is not listed by LISTBRKS.*
4. *Certain commands are not allowed while I/O is active. When execution is suspended for output, the NEXT command can be issued to cause execution to be suspended again after the completion of the I/O operation. At the next statement boundary with a debugging hook, execution will be suspended, and you may issue other commands.*
5. *Under ISPF, when an Interactive Debug command (or commands in a command list) produces more lines of output than the output halt value, a NEXT is forced to ensure that the output is displayed. A message will appear in the ISPF message area of the terminal, stating:*

   ```
   "NEXT" FORCED FOR OUTPUT
   ```

   The default for the output halt value is 50 lines, but you can change this value on the PROFILE panel under ISPF Version 2.
6. *If the statement at which execution is to be suspended has a breakpoint set with an AT command, including a command list that causes execution to resume, the NEXT command will not cause execution to be suspended at that statement.*
7. *NEXT is not permitted after the VS FORTRAN program has terminated.*
8. *The STEP command can be issued to replace NEXT/GO combinations. See "STEP Command" on page 203 for more information.*

Example 1

Suspend execution after executing one statement.

```
AT: MAIN.10
next
go
NEXT: MAIN.11
next
go 40
NEXT: MAIN.40
```

Example 2

You are in an error exit, and want to suspend execution after having performed corrective action.

```
ERROR EXIT: ERROR 209 AT MAIN.11
next
fixup
STANDARD CORRECTIVE ACTION TAKEN. EXECUTION CONTINUING.
NEXT: MAIN.12
```

For additional examples of the NEXT command, see "Controlling Program Execution" on page 74.

OFF Command

The OFF command removes breakpoints in the currently qualified program unit.

OFF can also be used without parameters as one of the cursor commands under ISPF Version 2. For more information, see "Using Interactive Debug Features under ISPF Version 2" on page 24.

Abbreviation: None

Syntax

```
OFF [qual.] {number[:[qual.]number] | ENTRY | EXIT } | * |
 (number/ENTRY/EXIT list)
```

qual
specifies a program unit name prefix to temporarily override the current qualifier. The program unit name is used for the prefixed operand only.

number
specifies the statement label, ISN, or sequence number of a single breakpoint you want to remove. Precede a statement label with a slash to distinguish it from an ISN or sequence number.

number:[qual.]number
specifies a range of statement labels and/or statement numbers (ISNs or sequence numbers). Breakpoints set at any statement within the range are removed. Statement labels and statement numbers (ISNs or sequence numbers) can be combined in the range. Precede each statement label with a slash.

Statement identifiers can be qualified with a program unit name. The default program unit for the first identifier is the current qualifier. The default program unit for the second identifier is the program unit specified or defaulted for the first identifier. Both identifiers must have the same program unit in effect.

ENTRY
specifies that the entry breakpoint is to be removed.

EXIT
specifies that the exit breakpoint is to be removed.

*
specifies that all breakpoints will be removed from the qualified program unit.

(number/ENTRY/EXIT list)
specifies a list of statement labels, ISNs or sequence numbers, entry points, exit points, and ranges of numbers. Breakpoints set at each specified statement and within each range are removed. Enclose the list in

parentheses, and separate entries with commas or blanks. Precede each statement label with a slash.

If the number of a statement that does not have a debugging hook is entered in the number list, an error message is issued but breakpoints are still removed from the remaining statements.

Notes:

1. *OFF with no operands is only valid when used as a cursor command under ISPF Version 2.*

2. *OFF is not permitted after the VS FORTRAN program has terminated.*

Example 1

Remove all breakpoints in the currently qualified program unit.

```
off *
```

Example 2

Remove breakpoints at statement numbers 120 and 560 in program SUB1.

```
off (sub1.120 sub1.560)
```

Example 3

Remove specific breakpoints in the currently qualified program unit.

```
off (20:80 /100 ENTRY)
```

OFFWN Command

The OFFWN command turns off the monitoring of WHEN conditions.

Abbreviation: None

Syntax

OFFWN condition name | * | (condition name list)

condition name
specifies the 1- through 4-character name of a WHEN condition that is currently being monitored and that you want to stop monitoring.

*
turns off all WHEN condition monitoring.

(condition name list)
specifies a list of such WHEN condition names. Monitoring is stopped for all of them. Enclose the list in parentheses, with individual names separated by commas or blanks.

Notes:

1. *Use of OFFWN to turn off monitoring of a condition does not remove the definition of the condition. Any condition can be reactivated by using the WHEN command.*
2. *To see the currently defined conditions, use the LISTBRKS command.*
3. *OFFWN is not permitted after the VS FORTRAN program has terminated.*

Example 1

Stop monitoring all WHEN conditions.

```
offwn *
```

Example 2

Stop monitoring a certain WHEN condition called ABS.

```
offwn abs
```

For additional examples of the OFFWN command, see "Controlling Program Execution" on page 74.

POSITION Command

The POSITION command positions the cursor in the log file at a specified log line, or in the source window at a specified ISN or sequence number. This command is valid only under ISPF Version 2, and cannot be issued in a command list, an IF command, an attention exit, or a restart file.

Abbreviation: POS

Syntax

```
POSITION number
```

number
specifies either a statement number (an ISN or sequence number), or a log line number to be used as the target. If the cursor is within the source window, Interactive Debug interprets the number as an ISN or sequence number. Otherwise, it is interpreted as a log line number.

Notes:

1. *If the search is successful, the cursor is placed at the beginning of the ISN or sequence number, or at the log line number.*

2. *If the search is successful, the log or source listing is scrolled vertically so the target line is displayed as the first line on the screen (unless it is already displayed in the current ISPF panel).*

3. *If the statement number or log number is not found, you will receive the message "TARGET NOT FOUND" in the upper right corner of the screen.*

4. *Only the last 1000 lines of the log are available for display during the debugging session. A target located below the last 1000 lines will cause POSITION to respond with the message "TARGET NOT FOUND."*

5. *If you issue POSITION without a parameter, you receive the message "PARAMETER MISSING."*

6. *If you issue POSITION with a nonnumeric parameter, you receive the message "INVALID PARAMETER(S)."*

7. *Sequence numbers (in columns 73 through 80) can be used only for program units that were compiled with VS FORTRAN Version 2 with the SDUMP(SEQ) option. In all other cases, ISNs must be used.*

8. *If you have MOVECURS assigned to a PF key, you can type the POSITION command on the command line. Then, without pressing ENTER, press the MOVECURS PF key to move the cursor to the source window, and finally press ENTER to search the source listing.*

Example

The following command searches for ISN or log line 100, depending on cursor position.

```
position 100
```

PREVDISP Command

Under ISPF Version 2, the PREVDISP command redisplays the previous panel displayed by the application program (if ISPF was used). The panels are saved automatically by ISPF, and are redisplayed with an ISPF message "Saved Panel Display."

PREVDISP is valid only under ISPF Version 2, and cannot be issued in a command list, an IF command, an attention exit, or a restart file.

Warning: The variables in the program are actually reset to the values of the variables in the previously displayed panel, (unless the application program has used VDELETE for those variables). The information in a saved panel is redisplayed exactly as it appeared when the panel was originally displayed, and may no longer be correct. You will not receive a message to remind you of this, nor will the change in values be logged.

Abbreviation: PREV

Syntax

PREVDISP

Notes:

1. *There are no operands for the PREVDISP command.*

2. *If no previous panel exists, the first PREVDISP Help panel is displayed.*

   If the HELP panel is displayed, you cannot access the main menu or subsequent panels.

3. *The display of a previous panel is not an active display. For example, if you redisplay an EDIT session, you cannot edit the panel as if you were in an editing session.*

   You can, however, change the values of application variables on a saved panel display, but this is not recommended. The application program may not be prepared to have the variables changed at that point.

4. *GDDM must still be active to redisplay a panel with a graphic area. If GDDM is not active (for example, if GRTERM has been invoked), the graphic area will be empty.*

PROFILE Command

The PROFILE command displays a panel containing the settings for various parameters that affect the way your debugging session executes. Both the default profile settings and your current settings are shown for each parameter. This command is valid only under ISPF Version 2, and cannot be issued in a command list, an IF command, an attention exit, or a restart file.

Abbreviation: None

Syntax

```
PROFILE
```

Notes:

1. *Initially, the current setting for any of the parameters displayed will be the same as your profile setting. However, if you want to modify a parameter for the current session, you can type over that field on the display panel with a new value. If you modify the profile settings, the new values are saved and will be used whenever you begin a new debugging session.*

   Initially, the profile settings will have the values displayed in the example below. However, you can modify these values (in addition to modifying the current settings).

2. *The parameters displayed in the PROFILE panel are:*

   Step delay *Initially set to 50. This value controls the pace of animation, measured in hundredths of a second. For more information, see Figure 15 on page 32.*

   Frequency count display
   Initially set to YES. YES indicates that the statement execution counts will be shown within the source listing window.

   Window columns
   Initially set to 40. This value specifies the width of the source window listing, measured in characters. To make the window wider, increase this value.

   Window rows
   Initially set to 10. This value specifies how deep the source listing window will be, measured in lines. To see more than 10 lines at a time in the window, increase this value.

   Window on *Initially set to YES. This value indicates that a source listing window will automatically appear on your screen, provided that its dimensions have been defined and the listing file is available for the current program unit. Setting this value to YES is equivalent to entering the* `WINDOW` *or* `WINDOW ON` *command.*

Log line numbers

Initially set to YES. This value indicates that the log line numbers will be displayed in your scrollable log. Enter NO to inhibit the display of these numbers.

Output halt value

Initially set to 50. This is the number of output lines after which Interactive Debug will initiate a break to be sure output is displayed periodically. The break causes a NEXT FORCED FOR OUTPUT or HALTED FOR OUTPUT condition to occur.

Example

Display the `PROFILE` command to modify the value of some of your current settings.

```
PROFILE
```

The profile panel might look like this after you modify some of the current settings:

```
VS  FORTRAN INTERACTIVE DEBUG      (CURRENT AND PROFILE SETTINGS)

COMMAND ===>

                          CURRENT SETTING      PROFILE SETTING
                          ---------------      ---------------
Step delay (.01 sec)            10                   50
Frequency count display         NO                   YES
Window columns                  60                   40
Window rows                     10                   10
Window on                       NO                   YES
Log line numbers                YES                  YES
Output halt value               50                   50

Enter END or RETURN to go back to the IAD panel.
```

PURGE Command

The PURGE command terminates the output of a single Interactive Debug command after the output has been suspended by an attention interrupt. Subsequent commands in a command list are not affected. Following the PURGE command, resume execution by entering a null line.

Abbreviation: None

Syntax

PURGE

Notes:

1. *There are no operands for the PURGE command.*
2. *PURGE cannot be used to stop output being produced as a result of a VS FORTRAN WRITE statement or a command in a static DEBUG packet.*
3. *PURGE cannot be used to stop output from the HELP command.*
4. *The terminal displays the results of commands more slowly than the processor produces these results. This may cause the processor to begin processing the command(s) following the one displayed on the terminal at the same time the null line (ATTN) was issued. In this case, PURGE may not have the desired results.*
5. *PURGE has no effect outside an attention exit.*

Example

The following AT command list is being executed:

```
at 100 (list a% set i=0%list b% where% go)
```

A contains 1000 elements. When A starts to be displayed, stop the display before its completion.

```
Press ENTER (or ATTN)
purge
```

This sequence suppresses the output of A. When a null line is entered, execution will resume with the next command in the command list (set i=0).

QUALIFY Command

The QUALIFY command changes or displays the current qualification. This command allows you to change the default qualification that determines which program unit any unqualified statement or variable references apply to.

Abbreviation: Q

Syntax

QUALIFY [program]

program
specifies the name of the main program or subroutine, or the name of a function subprogram.

Notes:

1. *The QUALIFY command remains in effect until the next QUALIFY command, or until execution is resumed. When execution is resumed, the current qualification is reset to the executing program unit.*

2. *If a QUALIFY command is not entered, it is assumed that any Interactive Debug commands apply to the currently executing program unit (except for individually qualified operands).*

3. *A QUALIFY command without the program unit parameter will display the name of the currently qualified program unit.*

4. *QUALIFY is not permitted as the command specified in an IF command.*

5. *To qualify an individual VS FORTRAN variable, place the variable name after the name of the program unit.*

   ```
   list sub1.x
   ```

6. *To qualify an individual statement identifier, place the number or statement label after the name of the program unit. The statement label must be preceded by a slash.*

   ```
   at sub1./50
   ```

7. *If you type over the Q: field in ISPF Version 2, a QUALIFY command is generated and logged.*

Example 1

Display the value of all the variables in subroutine SUB1 while execution is suspended at a breakpoint in the main program.

```
qualify sub1
list *
```

Example 2

Display the name of the currently qualified program unit.

```
qualify
```

For additional examples of the QUALIFY command, see "Referring to Statements or Variables in Other Program Units" on page 70.

QUIT Command

The QUIT command exits Interactive Debug and returns program control to ISPF, CMS, or TSO.

Abbreviation: None

Syntax

QUIT

Notes:

1. *If QUIT is issued following an attention interrupt, your program will be terminated. You may then issue certain Interactive Debug commands (for example, LIST) before returning to ISPF, CMS, or TSO. You must issue another QUIT to terminate the session.*

2. *Commands following QUIT in a command list are ignored.*

3. *The ISPF END command cannot be used to terminate a debugging session. The QUIT command must be used. This is designed to avoid accidentally terminating a debugging session with PF key 3.*

Example

Discontinue debugging and return to ISPF, CMS, or TSO.

```
quit
```

RECONNECT Command

The RECONNECT command resets a file to its original (preconnected) condition. This may be necessary if you have used the CLOSE command or your program has executed a CLOSE, and you wish to make it possible for the program to do additional I/O to the preconnected file.

Abbreviation: RECONN, RECONNEC

Syntax

```
RECONNECT {number | [qual.]integer-variable |
 [qual.]integer-array-element}
```

number
is the number of the I/O unit associated with the sequential file on which the reconnect is to be performed.

[qual.]integer-variable
is the name of an integer variable in the VS FORTRAN program. This variable specifies the number of the I/O unit associated with the sequential file on which the reconnect is to be performed.

[qual.]integer-array-element
is the name of an element of an integer array in the VS FORTRAN program. This element specifies the number of the I/O unit associated with the sequential file on which the reconnect is to be performed.

Notes:

1. *"Number," "integer-variable," or "integer-array-element" must be specified; there is no default number.*

2. *This command may not be issued when I/O is currently active.*

3. *RECONNECT is only necessary if the OCSTATUS run-time option is in effect and you wish to allow your program to perform additional I/O on a file that has been closed, without executing another OPEN.*

4. *When referring to array elements, the subscripts must use simple arithmetic expressions no more complex than the form "variable plus (or minus) a constant." For example,*

   ```
   ARY(I), ARY(3), ARY(I+3) or ARY(I-3)
   ```

   are valid forms.

Example

Reconnect the sequentially accessed external file associated with I/O unit 8.

```
reconnect 8
```

REFRESH Command

The REFRESH command controls whether or not the Interactive Debug panel is completely refreshed when Interactive Debug panels are displayed.

If you have applications that do full screen I/O without using ISPF, there may be changes to the screen contents that ISPF is not aware of, and portions of the application display may remain on the screen. REFRESH is helpful in these situations.

REFRESH is not valid in line mode or batch mode.

Abbreviation: None

Syntax

```
REFRESH [ON | OFF]
```

ON
indicates that every display of an Interactive Debug panel should rewrite the entire screen.

OFF
indicates that ISPF does not need to rewrite portions of the screen that already seem to have the proper contents. This is the initial setting for REFRESH.

Notes:

1. *When refresh is on, animation is less smooth because the entire screen is refreshed for each step. The screen may appear to flash each time.*
2. *Response time may increase for remote terminals when refresh is on.*
3. *Entering* REFRESH *without a parameter queries the status of REFRESH.*

Example

Query the current status of REFRESH:

```
refresh
```

RESTART Command

The RESTART command allows you to restart a debugging session in full screen mode without clearing the log file. The variable values are all cleared unless they are in dynamic commons. The VS FORTRAN program restarts at the first executable statement. Any new log information is appended to the existing log file.

RESTART is not valid in line mode or batch mode. It is also not allowed in a restart file.

Abbreviation: None

Syntax

```
RESTART
```

Notes:

1. *There are no operands for the RESTART command.*
2. *Commands following RESTART in a command list are ignored.*
3. *When RESTART is issued, all Interactive Debug settings such as breakpoints, WHEN conditions, HALT status, AUTOLIST window, and so on are reset. The TIMER status is turned off and all times are cleared.*
4. *Under TSO, you can recompile a program unit using a split screen, then issue RESTART to restart the debugging session using the new object deck. (This is not possible under CMS.)*
5. *The AFFON and AFFIN files are re-read when you issue RESTART. If you have modified these files (for example, in a split-screen edit session), the new files are used.*

Example

Restart a debugging session in full screen mode, but retain the current log file.

```
restart
```

REWIND Command

The REWIND command positions a sequentially accessed external file at the beginning of the first record. Its usage is similar to that of the REWIND statement in the VS FORTRAN Version 2 language, allowing you to move to the beginning of the file.

Abbreviation: REW

Syntax

```
REWIND {number | [qual.]integer-variable | [qual.]integer-array-element}
```

number
is the number of the I/O unit associated with the sequential file that is to be rewound.

[qual.]integer-variable
is the name of an integer variable in the VS FORTRAN program. This variable specifies the number of the I/O unit associated with the sequential file that is to be rewound.

[qual.]integer-array-element
is the name of an element of an integer array in the VS FORTRAN program. This element specifies the number of the I/O unit associated with the sequential file that is to be rewound.

Notes:

1. *"number," "integer-variable," or "integer-array-element" must be specified; there is no default number.*

2. *When referring to array elements, the subscripts must use simple arithmetic expressions no more complex than the form "variable plus (or minus) a constant." For example,*

   ```
   ARY(I), ARY(3), ARY(I+3) or ARY(I-3)
   ```

   are valid forms.

3. *This command may not be issued when I/O is currently active.*

4. *VS FORTRAN Version 1 and VS FORTRAN Version 2 support multiple files under the same I/O unit. The REWIND command sets the VS FORTRAN file name to the first in the sequence of files for the specified I/O unit. For example, if you were currently processing file FT08F003 on I/O unit 8, and entered:*

   ```
   rewind 8
   ```

   I/O unit 8 would be connected to file FT08F001, which would be positioned at the beginning of the first record.

Example

Rewind the sequentially accessed external file associated with logical unit 4 so that it may be rewritten.

```
rewind 4
```

For additional examples of the REWIND command, see "Processing External Files" on page 89.

SEARCH Command

The SEARCH command searches the source listing window or the scrollable log for a given character string. SEARCH is valid only under ISPF Version 2, and cannot be issued in a command list, an IF command, or a restart file.

Abbreviation: None

Syntax

```
SEARCH /string[/]
```

string

specifies a character string to be searched for. The search is not case sensitive, so your string can be found in any combination of upper or lower case. If the cursor is within the source window, the source listing is searched; otherwise, the last 1000 lines of the log are searched. The character string can be up to 64 characters long.

You can use any nonblank character as a delimiter. A slash (/) is shown in the syntax box above.

The initial and final delimiter must be the same character. The final delimiter is required if the search string contains trailing blanks. In all other cases, it is optional.

Notes:

1. *If the search is successful, the log or source listing is scrolled vertically and horizontally so the target line is displayed as the first line on the screen with the string visible (unless it is contained in the current ISPF panel). The cursor is placed at the beginning of the string.*

2. *The source listing search is performed starting at the top line displayed on the screen. However, if the SEARCH arguments are identical and the last-found search argument is still on the current screen, the search begins at the location of the last-found search argument.*

3. *When searching the log, only the last 1000 lines of the log are available.*

4. *If the search requires the cursor to return to the beginning of the source listing or the log, the message "WRAPPED..." is displayed in the upper right corner of the screen.*

5. *If the search argument is not found, you will receive the message "TARGET NOT FOUND" in the upper right corner of the screen.*

6. *If you search without a parameter, the most recent character string used as a target is searched for again. If there is no previous string, you will receive the message "NO SEARCH ARGUMENT."*

7. *If you have MOVECURS assigned to a PF key, you can type the SEARCH command on the command line. Then, without pressing ENTER, press the MOVECURS PF key to move the cursor to the source window, and finally press ENTER to search the source listing.*

Example

Search for the character string VM/CMS, using a question mark as the delimiter.

```
search ?VM/CMS?
```

SET Command

The SET command changes the value of a variable, array, array element, or group of array elements.

Abbreviation: S

Syntax

```
SET [qual.]name=value[,value...]
```

qual
specifies a program unit name to temporarily override the current qualifier for the prefixed name only.

name
specifies the name of a variable, array, or array element.

value
is the value to be assigned to a single variable or single array element. A group of values (separated by commas or blanks) can be assigned to an entire array or part of an array. A value can also be another qualified variable name or array element, and can be prefixed by a numeric replication factor.

Notes:

1. *Valid SET command assignments for the different types of names are shown in Figure 30. All names can be qualified.*

Name	Set	Type of Value	Example
Real, Integer, or Complex scalar	=	Another scalar variable An array element A constant	ALPHA=BETA ALPHA=-BETA ALPHA=A(3) ALPHA=-A(3) NUM=7
Logical	=	Another logical variable An array element A logical value	LOG1=LOG2 LOG1=LOG(2) LOG=.TRUE.
Character	=	Another character variable An array element A character constant A substring	CHAR1=CHAR2 CHAR1=CHAR(2) MSG='HELLO' A(1:3)='ABC'

Figure 30 (Part 1 of 2). Valid SET Command Assignments

Name	Set	Type of Value	Example
Array element	=	Another array element A scalar variable A constant	A(4)=B(1) AR(2,2)=-AR(5,5) C(7)=RATE C(8)=-TIME D(I,J)=0.0
Contiguous array elements	=	Value,value,... (Values can be variables, array elements, or constants; multiple assignments of a value can be entered as n*value.)	A=3*1.0,4*0.0, 7.2,5.,ACCL, 8.5E9 B(J,K)='C', 3*'Q',2*,'X'

Figure 30 (Part 2 of 2). Valid SET Command Assignments

2. *When referring to array elements, the subscripts must use simple arithmetic expressions no more complex than the form "variable plus (or minus) a constant." For example,*

   ```
   ARY(I), ARY(3), ARY(I+3) or ARY(I-3)
   ```

   are valid forms.

3. *No arithmetic operations (except negation) are allowed in the value assignments; for example, SET A=B+4 is not allowed.*

4. *Upper- and lowercase character constants may be entered at the terminal. They must be enclosed in single quotation marks when entered, but the enclosing single quotation marks are removed before the assignment is performed. If a single quotation mark is to be assigned as part of the character constant, two single quotation marks must be entered; for example:*

   ```
   set c=''''
   ```

   sets C to a single quotation mark. Character strings are truncated or extended with blanks to match the length of the receiving character variable or array element.

5. *In assigning values to contiguous array elements, values may be repeated using the notation n*value. For example,*

   ```
   set ary=10*4
   ```

   sets the first 10 elements of the array ARY to 4.

6. *In assigning values to contiguous array elements, elements can be omitted by using the asterisk notation with no value following the asterisk. For example, a single omission is entered as 1*, and a multiple omission might be entered as 3* (this would leave three successive elements of the array unchanged).*

7. *Substring notation is permitted with string variables.*

8. *On the right side of the "=" sign, an array reference can have subscripts that exceed the bounds of the dimensions. A warning message will be issued except for the special case where only the last dimension is exceeded, and that dimension is "*" or "1."*

9. *An assumed size array is not set unless a specific element or range of elements is specified. An assumed size array is an array with the last upperbound declarator specified as an asterisk (*). Results are unpredictable if you SET array elements beyond the end of the original array.*

10. *Dummy arguments can only be used or set when the program unit in which they are defined is active. (Note that a program unit is not yet active when suspended at entry.) If this rule is violated, you will get an error message.*

11. *Variables in a dynamic common can only be used or set after the program unit used to qualify the variable has been activated at least once. If this rule is violated, you will get an error message.*

12. *When setting a variable, results are unpredictable if:*

    - *More values are specified than will fit in an assumed-size array or an array whose last dimension is "1."*
    - *The right hand side contains an array reference whose subscripts are not all within the array dimensions. You will get a warning message in this case.*
    - *The right-hand side of the statement contains an inaccessible variable. You will get an error message in this case.*

Example 1

Change the values of several variables.

```
set dan=8.9e+7
set m=-int
set a(3)=b(5)
set c(i,2)=4.1
set d(10)=xray
set a(i-1,3)=b(4,j+6)
set main.x=sub1.x
set quote='he said: ''bye, bye.'''
```

Example 2

Set the ten elements of array ALFIE to 0, 0, 0, .666, .21E-08, 1.0, 0, 0, 0, 0.

```
set alfie=3*0.0,.666,.21e-8,1.0,4*0.0
```

Example 3

Set the second element of array DATA to 0, leave the third element unchanged, set the fourth through eighth elements to 1.0, and leave all other elements unchanged.

```
set data(2)=0,1*,5*1.0
```

Example 4

Set the third through the fifth characters of character string DATA to 'ABC'.

```
set data(3:5)='ABC'
```

STEP Command

The STEP command executes one or more FORTRAN statements and then gives control back to you. STEP is similar to a series of NEXT and GO command pairs.

Under ISPF Version 2, STEP execution is automatically *animated* if the source listing is available to Interactive Debug. Animated execution means that the source listing window is refreshed at each statement boundary where a hook exists, and the current statement is highlighted. Highlighting is determined by the COLOR settings. If an AUTOLIST display is defined in full screen mode, it will also be updated, or *refreshed* as necessary at each debugging hook.

Abbreviation: ST

Syntax

```
STEP [number]
```

number
specifies the maximum number of hooked statements to be executed before execution stops. The number must be a positive integer. The default is one.

Notes:

1. *The STEP command itself causes your program to resume execution. You do not need to use a GO command.*

2. *If any of the following conditions occur, STEP execution will terminate before the STEP count runs out:*

   - *A breakpoint is encountered.*
   - *A WHEN condition is satisfied.*
   - *The HALT status is satisfied.*
   - *An error condition is detected.*
   - *Terminal input occurs with TERMIO IAD in effect.*
   - *Terminal output exceeds the output halt value in full screen mode.*
   - *An attention is issued.*
   - *The program terminates.*

3. *If STEP processing ends after finding one of the above conditions or after reaching the end of the step count, you cannot resume STEP. You must issue a new STEP command.*

4. *If a STEP command is terminated because of a terminal READ, a NEXT will be forced at the next hooked statement.*

5. *If STEP is issued within a command list, the remainder of the list is ignored.*

6. *STEP counts only those statements that have debugging hooks. Thus, STEP may execute many statements before stopping. Statements that have been collapsed and statements not included in the AFFON statement list are not counted and execution does not stop.*

7. *STEP is not permitted after the VS FORTRAN program has terminated, or while a READ is pending. If issued in an error exit, standard corrective action is taken.*

Example

Execute the next 12 hooked statements before suspending execution:

```
step 12
```

SYSCMD Command

The SYSCMD command executes system commands during an Interactive Debug session. SYSCMD can also be included in a command list and on the IF command.

Abbreviation: SYS, CMS, TSO

Syntax

```
SYSCMD [system–command]
```

system–command
is a CMS or TSO system command to be executed.

Notes:

1. *Under ISPF, the abbreviations CMS and TSO are recognized and executed, but the command entered will not appear in the session log.*

2. *Caution should be observed when issuing commands that would cause the currently executing program to be overlaid. For example, a CMS LOAD command with the CLEAR option could cause Interactive Debug and the VS FORTRAN application program to be erased from storage.*

3. *CMS Considerations: If the system command is omitted, the standard CMS SUBSET will be entered. In this mode, CMS commands may be issued and Interactive Debug will not regain control until the RETURN command is issued. CMS commands issued in this mode (or specified with the SYSCMD command) are limited to those commands allowed in CMS SUBSET mode.*

4. *TSO Considerations: If the system command is omitted, a special command entry mode will be entered. Interactive Debug will produce the message,*

   ```
   ENTER A TSO COMMAND OR A NULL LINE
   ```

   In this mode, TSO commands may be issued and Interactive Debug will pass them along to TSO until a null line is entered.

5. *In order to use SYSCMD in batch mode on MVS, it is necessary to run a TSO Terminal Monitor Program (TMP).*

6. *In batch mode, the system command must be specified, and must not be a command that requires interaction. Interactive Debug cannot guard against system commands that require interaction during a batch session; this is your responsibility.*

CMS Example

List the files that have been allocated:

```
syscmd q filedef
```

TSO Example

List the data sets that have been allocated:

```
syscmd listalc status
```

For additional examples of the SYSCMD command, see "Using System Commands" on page 90.

TERMIO Command

The TERMIO command allows you to select the I/O routines that you want to use for terminal I/O for your VS FORTRAN program. You can select either the VS FORTRAN Version 2 Interactive Debug I/O routines, or the VS FORTRAN library routines.

You can also use TERMIO to send a copy of batch mode output to a user as message text. To query the current settings, enter `TERMIO` with no operands.

Abbreviation: None

Syntax

```
TERMIO [IAD | LIBRARY] [MSG [(userid)] | NOMSG]
```

IAD
indicates that terminal I/O is to be performed using the VS FORTRAN Version 2 Interactive Debug I/O routines. The Interactive Debug I/O routines combine input and output from the VS FORTRAN program with Interactive Debug input and output. You must precede terminal input with a percent sign (%) to distinguish it from Interactive Debug commands.

This is the setting when Interactive Debug execution begins.

In full screen mode or batch mode, terminal input and output are included in the log file. This includes library error messages.

LIBRARY
indicates that terminal I/O is to be performed using the VS FORTRAN library routines. The library I/O routines cause output from the VS FORTRAN program to be displayed as it would be if the program were not being debugged.

Terminal input and output are not included in the log file. In full screen mode, a request for terminal input causes the screen to be cleared and the keyboard to be unlocked. Terminal output is written on a blank screen.

MSG
In batch mode only, this indicates that a copy of each line of Interactive Debug input and output is to be sent to the specified or defaulted user ID as message text.

userid
specifies the user ID to which message text is to be sent. If this operand is omitted, it defaults to the previously established user ID (if one exists), or to the submitter's user ID if available. The operand is required if the submitter's user ID is not available and no user ID has been previously established.

The default user ID is obtained from the JOB card information in MVS, if available. In CMS, no default user ID is available.

NOMSG
specifies that Interactive Debug input and output are not to be copied as message text. This is the initial setting.

Notes:

1. *When no operands are specified, the current TERMIO setting is displayed.*

2. *This command does not affect I/O operations other than those requested by the VS FORTRAN application program. This implies that, if Interactive Debug is executing under ISPF, a specification of LIBRARY will cause any application program terminal I/O operation to occur in line mode. At the completion of the I/O operation, full screen operation will resume.*

3. *When running in batch mode on MVS, a VS FORTRAN program has no real terminal inputs or outputs. You can simulate these terminal inputs and outputs by specifying one or more units on the DEBUNIT execution-time option for your VS FORTRAN program. If you do not specify the DEBUNIT option, the IAD/LIBRARY operand has no effect on program I/O.*

4. *Whenever character data is entered, Interactive Debug I/O routines change it to uppercase. If mixed-case input is required (and supported by the host system), the library I/O routines must be used.*

5. *If output cannot fit on one line in whatever mode Interactive Debug is operating, it is split across multiple lines. The lengths of these lines are 60 characters if Interactive Debug I/O routines are used. When the library routines are used, I/O operations produce the same results as would be produced if the program were executed without Interactive Debug.*

6. *When TERMIO IAD is in effect, any continuation line that begins with leading blanks must be prefixed with a quotation mark (''). The quotation mark will not be passed to the program.*

Example 1

Specify that Interactive Debug routines are to be used for subsequent I/O requests of the VS FORTRAN program.

```
termio iad
```

Example 2

Display the current setting of the terminal I/O mode.

```
termio
```

Example 3

Specify that Interactive Debug input and output are to be echoed to user ID "SMITH" (in batch mode).

```
termio msg(smith)
```

For additional examples of the TERMIO command, see "Entering Terminal Input" on page 91.

TIMER Command

The TIMER command controls the timing of program units. If timing is activated for a program unit, that unit will be timed when called. The LISTTIME command displays the timing information.

Abbreviation: None

Syntax

```
TIMER { * | program-unit-name | (program-unit-name list) }
 [ON | OFF | RESET ]
```

*
: specifies that the command applies to all debuggable program units.

program-unit-name
: specifies an individual program unit.

program-unit-name list
: specifies a list of program unit names. Enclose the list in parentheses and separate entries with commas or blanks.

ON
: specifies that timing is to be activated for the indicated program units. The program units are timed when called. ON is the default.

OFF
: specifies that timing is to be deactivated for the indicated program units. This does not clear the timing values or activation counts. The program units are no longer timed. OFF is the initial setting for TIMER.

RESET
: specifies that timing information and activation counts are to be reset to zero for the indicated program units.

Notes:

1. *Timing is cumulative. Timing values are only reset by* `TIMER name RESET`.

2. *The time for a routine is measured beginning at the entry point and ending at the exit. If a call is made to another routine for which TIMER is on, the time spent in the second routine (and lower-level routines) is not included in the measurement for the first routine. However, time spent in called (and lower-level) nontimed routines* is *included in the measurement for the calling routine. For example, if program A calls program B and you do not want the time in B to be included in the timing of A, you must specify:* `TIMER (A,B) ON`. *Program B must be debuggable.*

3. *In MVS, timing measurements will be incorrect if your program uses the STIMER macro, or if it uses a system service that calls STIMER. This includes the BTAM OPEN and LINE OPEN operations, and Dynamic Allocation. Animated execution with the STEP command under ISPF Version 2 also interferes with timing on MVS.*

4. *Timing for very small routines may be erratic.*

5. *The timing information provided by Interactive Debug includes overhead caused by the debugging hooks in your program. Thus, the timing information is not an accurate representation of the time it takes to execute* **without** *Interactive Debug.*

   To get the most accurate timing information, there should be hooks only at entry and exit of the program unit. To set only entry and exit hooks, specify `name ENTRY` *in the AFFON file. For more information, see* *"Displaying Timing Information" on page 82.*

6. *The ENDDEBUG command turns timing off for all program units.*

7. *TIMER is not permitted after the VS FORTRAN program has terminated.*

8. *Timing is measured separately for each entry point in the program unit.*

Example 1

Turn timing on for program units MAIN and SUB2:

```
timer (main,sub2)
```

Example 2

Reset timing to zero for program unit MAIN:

```
timer main reset
```

Example 3

Turn timing off for all debuggable program units:

```
timer * off
```

TRACE Command

The TRACE command starts or stops tracing of the flow of the program as it executes. You can trace each transfer of control in the program, trace just entries to and exits from debuggable subroutines, or determine the current trace status.

Abbreviation: T

Syntax

```
TRACE [GOTO | ENTRY | OFF] [PRINT]
```

GOTO
specifies that a record of each apparent branch taken within the program is to be created. GOTO produces a listing showing a statement label or statement identifier for the origin and destination of each transfer made, including entries to and exits from debuggable subroutines.

ENTRY
specifies that only a record of entries to and exits from debuggable subroutines is to be produced.

OFF
turns off tracing that you previously initiated. This is the initial setting.

PRINT
specifies that output is to be written to the print data set instead of to the terminal.

Notes:

1. *Tracing continues until turned off, or altered by another TRACE command, or an ENDDEBUG command is issued.*
2. *TRACE operations apply to all debuggable program units.*
3. *TRACE GOTO issues a trace message if two consecutively executed debugging hooks are not on consecutively stored statements. This also occurs if statements have been collapsed or vectorized, or have no debugging hooks.*
4. *TRACE is not permitted after the VS FORTRAN program has terminated.*
5. *If no operand is specified, TRACE issues a message describing the current trace status.*

Example 1

Trace each program transfer, and have the trace output sent to the print data set instead of to the terminal.

```
trace goto print
```

Example 2

Discontinue tracing entirely.

```
trace off
```

Example 3

Trace only the calls to and returns from subroutines and functions.

```
trace entry
```

For additional examples of the TRACE command, see "Tracing Program Execution" on page 83.

WHEN Command

The WHEN command allows you to suspend execution every time a particular condition is met. You can define a condition and supply its name, or restart monitoring of a previously defined condition. The condition is tested at all statements with debugging hooks.

Abbreviation: WN

Syntax

```
WHEN condition-name [(condition) | variable]
```

condition-name
identifies the condition. The condition name must be 1 through 4 alphameric characters, with the first character alphabetic. The condition name is only a name; it is not to be confused with the definition of the condition.

(condition)
defines a condition to be monitored. The condition itself appears only in the initial WHEN command. The condition must be enclosed in parentheses.

Only scalars and single array elements are allowed in the condition expressions. They can be explicitly qualified; for example, `sub1.x=4.0`

variable
specifies the name of a variable or an array element to be monitored for any change in value. Only the name is specified; no parentheses to enclose the name are used. The name can be explicitly qualified.

The condition used in the WHEN command can be either relational or logical.

Relational condition: A relational condition is a signed or unsigned variable or array element or constant, followed by a *relational operator*, followed by another signed or unsigned variable or array element or constant.

There are six relational operators that can be used to define test conditions between scalar variables, array elements or constants.

= or .EQ.

¬= or .NE.

> or .GT.

< or .LT.

>= or .GE.

<= or .LE.

Logical condition: A logical condition is a logical variable or a logical array element, optionally preceded by the negation operator (¬ or .NOT.). No other operators are permitted. If the value being tested is a logical variable or a logical array element, the negation operator can be applied in the condition definition. For example:

```
WHEN NTON (¬ LOGVAR)
```

Notes:

1. *The condition itself is only defined once. At that time, a condition name must be supplied. Subsequent WHEN commands referring to that condition should contain only the condition name.*

2. *Monitoring remains in effect after a condition is satisfied. To turn the condition off, issue an OFFWN command, naming the condition.*

3. *After being turned off by an OFFWN, condition monitoring can be reactivated by reissuing WHEN and specifying only the condition name.*

4. *WHEN conditions are evaluated at each debugging hook in each debuggable program unit. If you use a variable subscript, it is reevaluated each time the condition is tested. The array element actually tested, therefore, depends on the subscript value at that moment.*

5. *You can redefine an existing WHEN condition by entering a new WHEN command with the same condition name and a new condition definition.*

6. *Variables can be prefixed by a program unit qualifier to override the current qualification (at the time the WHEN command is issued). For example,*

   ```
   WHEN COND (MAIN.A .LT. SUB1.B)
   ```

7. *When referring to array elements, the subscripts must use simple arithmetic expressions no more complex than the form "variable plus (or minus) a constant." For example,*

   ```
   ARY(I), ARY(3), ARY(I+3) or ARY(I-3)
   ```

   are valid.

8. *In the relational condition form of the WHEN command, for example, WHEN TEST (A .GT. B):*

   - *When either variable or constant is a logical, character, or complex type, both must be of that same type, and a sign preceding the variable or constant is not permitted.*

   - *When either variable or constant is a logical or complex type, only the relational operators .EQ. and .NE. (or = and ¬=) may be used.*

   - *When character variables or constants of unequal length are compared, the shorter is considered to be extended with blanks during the comparison.*

9. *When an unparenthesized variable name is specified as the condition (for example, WHEN TEST NAME) and the variable is a character variable:*

   - *The length of the character variable is determined at the time the WHEN command is issued.*
   - *If the character variable subsequently changes length (perhaps because it is a parameter to the subroutine being monitored), the old and new values are compared by effectively extending the shorter one with blanks.*
   - *If a change in value is found, the new value is preserved, either extended or truncated, using the size of the variable at the time the WHEN command was issued.*
   - *The current length and value of the variable can be captured at any time by issuing the WHEN command with just the condition name (for example, WHEN TEST). It is not necessary to issue an OFFWN command first or to state the variable name again. (The WHEN command can be embedded in an AT command list to automate this process.)*

10. *When a specified condition is met, messages will be displayed. These indicate the condition name that was satisfied, and where execution is currently suspended.*

11. *If you refer to undefined variables (such as dummy arguments in an inactive subprogram), you will receive an error message at each statement where the condition is tested. To avoid these messages, you can use OFFWN in an AT EXIT command list, and WHEN to reactivate it in an AT ENTRY command list.*

12. *WHEN is not permitted after the VS FORTRAN program has terminated.*

Example 1

Start monitoring variable MIKE to see if it changes. Call the condition (the change to variable MIKE) OLD.

```
when old mike
```

Example 2

Define a condition named OVFL as an array element A(1,5) greater than 3.5E10. Start monitoring it.

```
when ovfl (a(1,5) .gt. 3.5e+10)
```

Example 3

If the monitoring from Example 2 has been turned off with an OFFWN command, restart monitoring condition OVFL.

```
when ovfl
```

For additional examples of the WHEN command, see "Controlling Program Execution" on page 74.

WHERE Command

The WHERE command identifies the statement at which execution is suspended.

Abbreviation: W

Syntax

WHERE [TRBACK] [FLOW] [PRINT]

TRBACK
specifies a traceback showing the names of all program units that are currently active.

FLOW
specifies a trace of the last 10 program transfers that were executed.

PRINT
specifies that output is to be written to the print data set instead of to the terminal.

Notes:

1. *If WHERE is used following an attention interrupt, it can only be used to identify the current statement. Any parameters are ignored.*
2. *If there are no active program units, WHERE TRBACK indicates that no subroutines have been called.*
3. *The current statement identified by the WHERE command has not yet been executed.*
4. *An automatic WHERE command is forced at the first debugging hook found. This is usually the first executable statement in the main program.*

Example 1

Find out where you are after execution was suspended by an attention interrupt.

```
where
```

Example 2

Find out the sequence of control transfers that led to the current breakpoint.

```
where trback flow
```

Example 3

Conditionally indicate that a breakpoint has been reached, even though the AT command list does not cause execution to be suspended. The WHERE command will only be executed if A is less than or equal to 4.8.

```
at 120 (if (a .gt. 4.8) go%where%list a%go) nonotify
```

For additional examples of the WHERE command, see "Tracing Program Execution" on page 83.

WINDOW Command

The WINDOW command opens or closes the source listing display window. The size of the window can be defined in the PROFILE command panel, or by positioning the cursor. If the cursor is positioned in the log area or an existing window and WINDOW is entered with no operand, the current cursor position determines the lower left corner of the source display window. The upper right corner is fixed.

WINDOW is valid only under ISPF Version 2, and cannot be used in a command list, an IF command, an attention exit, or a restart file.

Abbreviation: None

Syntax

```
WINDOW [ OFF | ON ]
```

ON

specifies that the source listing window is to be open. This is the initial and default setting under ISPF Version 2.

If the window size has been defined and "YES" has been specified in the SOURCE column of the ISPF data set name panel for the currently qualified program unit, the source listing is displayed in the window.

If a window has previously been defined and the cursor is positioned on the command line, the previous window dimensions will be restored. The previous dimensions are also used if the cursor is at an invalid position. In all other cases, new dimensions will be used.

OFF

temporarily removes the source listing window from the screen. The window size, if defined, is saved. The window is not displayed again until you enter `WINDOW` or `WINDOW ON`, provided that the listing file is known for the current program unit.

Notes:

1. *If the cursor is positioned on the command line, WINDOW with no parameters has the same effect as WINDOW ON.*

2. *When defining the window size by cursor position, WINDOW is easiest to use if assigned to a PF key.*

3. *You can turn the window on or off, or define the window's size, using the PROFILE command panel. For details, see "PROFILE Command" on page 186.*

4. *If you have specified NO in the SOURCE column of the ISPF data set name panel for the currently qualified program unit, or if you have not yet specified the source listing file name, WINDOW has no immediate effect.*

To specify source listing file names, type a question mark (?) in the first character of the Q: field of the screen header line, and the data set name panel will be displayed.

5. *The size of the window is saved in your user profile from one session to the next. When you later begin a new debugging session, the saved window size will be used to automatically define the source listing display window at the start of the session.*

6. *If the window is too wide for any part of the log to be seen on the left, the first log line visible below the source window becomes the top line of the log. In this case, no information is hidden beneath the source. Scrolling to the top of the log file displays log line 1 as the first line below the source.*

7. *It is possible to cover the entire screen with the source display (except for the header and command lines).*

Appendix A. Using the Interactive Debug HELP Facility

Help at the Terminal

The HELP command offers interactive tutorial assistance in learning about or using VS FORTRAN Version 2 Interactive Debug. The HELP command displays information about any VS FORTRAN Version 2 Interactive Debug command or function.

HELP contains:

- A brief introduction to VS FORTRAN Version 2 Interactive Debug, composed of a general discussion of VS FORTRAN Version 2 Interactive Debug components
- Individual command descriptions, each of which contains a description of the command, the syntax, usage notes, and examples
- A task-oriented section offering an explanation of related commands required to perform various basic debugging tasks

You can invoke help by entering HELP (or using an equivalent PF key). The main help menu will appear, or, if the last command was in error, a screen will appear containing an explanation of the specific command.

Alternatively, you can enter HELP with an operand to go directly to the information for a specific command or task.

Help screens for each command consist of:

- The function of the command
- The syntax of the command
- A description of each component of the syntax
- Usage notes and examples

Figure 31 on page 220 shows the main menu screen for the help facility. This screen lists all the topics for which help is available. In ISPF or CMS line mode, you can select additional information from the main menu.

A tutorial is included in the help facility to familiarize you with VS FORTRAN Version 2 Interactive Debug by describing a basic debugging session. The tutorial can only be accessed from the main menu.

Figure 32 on page 221 shows the task menu screen for the help facility, illustrating which tasks have information available. After selecting a task, another screen (or set of screens) is presented, describing how the task is accomplished. The task menu may be accessed from the main menu.

Figure 33 on page 221 is a sample help screen for an individual command. (In this case, more information about the CLOSE command would appear on additional screens.)

```
SELECTION ===>                          VS FORTRAN VERSION 2 INTERACTIVE DEBUG

              VS FORTRAN VERSION 2 INTERACTIVE DEBUG
                          Main Menu

  These are the topics available for help on VS FORTRAN Version 2
  Interactive Debug.  Select a topic by number.  When finished
  reading a topic, enter TOP to redisplay this menu.  Note: The
  enter key alternates between this menu and the task menu.


   1  Task Menu     14  go           26  next         38  rewind
   2  tutorial      15  halt         27  off          39  search
   3  annotate      16  help         28  offwn        40  set
   4  at            17  if           29  position     41  step
   5  autolist      18  list         30  prevdisp     42  syscmd
   6  backspace     19  listbrks     31  profile      43  termio
   7  close         20  listfreq     32  purge        44  timer
   8  color         21  listings     33  qualify      45  trace
   9  describe      22  listsamp     34  quit         46  when
  10  enddebug      23  listsubs     35  reconnect    47  where
  11  endfile       24  listtime     36  refresh      48  window
  12  error         25  movecurs     37  restart      49  *,"
  13  fixup

TOP for main menu
```

Figure 31. Main Menu for the Help Facility (under ISPF)

```
SELECTION ===>                          VS FORTRAN VERSION 2 INTERACTIVE DEBUG

              VS FORTRAN VERSION 2 INTERACTIVE DEBUG
                          TASK MENU

  The topics below describe debugging tasks.  To request
  information for one of the tasks, enter the corresponding number
  in the command line.  Note: The enter key alternates between this
  menu and the main menu.

   1  (Return to main menu)            13  Display variable values
   2  Specify files to debug           14  Handle library errors
   3  Use full-screen animation        15  Enter program input
   4  Enter commands in attention      16  Debug optimized code
   5  Debug in batch mode              17  Use program sampling
   6  Use command continuation         18  Process external files
   7  Use command lists                19  Define LISTING files
   8  Control program execution        20  Set breakpoints
   9  Use cursor-oriented commands     21  Execute a system command
  10  Display statement frequencies    22  Display timing information
  11  Display program information      23  Trace program execution
  12  Display data types

TOP for main menu
```

Figure 32. Help Facility Task Menu (under ISPF)

```
SELECTION ===>                          VS FORTRAN VERSION 2 INTERACTIVE DEBUG
CLOSE COMMAND                                                    panel 1 of 4

  The CLOSE command disconnects a VS FORTRAN external file from an input or
  output unit.  Its usage is similar to that of the CLOSE statement in the VS
  FORTRAN Version 2 language.  This command allows you to close an external
  file, for example to assign another file to the input or output unit, or to
  examine the contents of the file.

          Abbreviation: None
          Syntax:
                  CLOSE <|number | <qual.>integer-variable |
                  <qual.>integer-array-element|>

  NUMBER
    is the number of the I/O unit associated with the file that is to be
    closed.

TOP for main menu                                    hit ENTER for next page
```

Figure 33. Sample Help Screen

ISPF Procedures

When using VS FORTRAN Version 2 Interactive Debug under ISPF (in either CMS or TSO), the HELP command invokes the ISPF HELP facility. If you prefer to invoke CMS or TSO HELP, precede the HELP command with "CMS" or "TSO."

When HELP is specified without an operand, you will be presented with either the main menu or a description of an unsuccessfully executed command.

When you are viewing the main menu, and want to transfer to another panel, you can:

- Press the Enter key, causing the task menu to be displayed.
- Enter a number corresponding to one of the items listed on the menu.

The task menu operates in the same manner as the main menu. Pressing the Enter key will return you to the main menu.

If you specify HELP with an operand, the panel for that particular command or task will be displayed.

When you are viewing HELP information for a particular command or task, use the following procedures:

- Use PF3 (or enter QUIT) to terminate viewing of HELP information and return to Interactive Debug.
- Press the Enter key to move to the next panel. If you press Enter on the final panel for a particular command or task, you will be returned to the first panel of that command or task.
- Enter TOP on the command line of any panel to return to the main menu.

CMS Procedures

When executing VS FORTRAN Version 2 Interactive Debug in line mode under CMS, CMS HELP procedures must be used to transfer between panels. When HELP is entered following an Interactive Debug prompt (FORTIAD), Interactive Debug invokes the CMS HELP facility, passing along any options that were specified on the Interactive Debug HELP command.

When HELP is specified without any options, you will be presented with either the main menu, or a description of an unsuccessfully executed command.

To transfer from the main menu to another panel, you can:

- Position the cursor under one of the names on the menu and press PF1 (or press the Enter key).
- Enter a complete HELP command of the form:

```
HELP AFF name (options
```

The task menu operates in the same manner as the main menu. To view information about one of the tasks, enter HELP TASK name.

TSO Procedures

When executing VS FORTRAN Version 2 Interactive Debug in line mode under TSO, TSO HELP procedures are employed. To display the main menu in line mode, enter:

```
HELP IADHELP
```

Information is written to the terminal one line at a time until the screen fills up. Use the Enter key to display more information. There is no way to go back to information presented earlier without reentering the command.

If you enter a HELP command without options, you will receive either the main menu or information about an unsuccessfully executed command.

You will need to issue a separate HELP command for each topic. If you have just seen the main menu and now want to see the task menu, you must enter HELP TASK.

Appendix B. Interactive Debug Command Summary

Symbols used to describe the syntax of VS FORTRAN Version 2 Interactive Debug commands are the same as are used in other VS FORTRAN Version 2 publications.

- All entries in uppercase letters indicate the minimum abbreviation allowed.
- Square brackets indicate optional entries.
- Braces contain required entries separated by one or more vertical lines (|); select one.
- Defaults are underlined.

Command and Syntax	Function
* or " [comment]	Insert comments in the debug log.
Format 1 ANnotate {unit | (unit list) | * } [**SAMPLING** [**DIRECT** | CALLED | ALL] | FREQUENCY] *Format 2* ANnotate [ON | OFF | TOGGLE] [SAMPLING [**DIRECT** | CALLED | ALL] | FREQUENCY]	*Format 1* Copy source listings to a print file with program sampling information. *Format 2* Display a program sampling bar chart on the source listing window.
AT [qual.] {number [:[qual.]number] | ENTRY | EXIT } | (number/ENTRY/EXIT list) [(command list)] [Count(n)] [**NOTify** | NONotify]	Set breakpoints. Use a percent sign (%) to separate individual commands in a command list.

Command and Syntax	Function
AutoList [{[qual.] name [:[qual.]name] | * | 'string' | number | (specification list)} [Format [(code)] | Dump [(code)]]]	Automatically display values of variables in full screen mode. Valid only under ISPF. *Note:* Check the format and dump codes table. See Figure 26 on page 127.
BACKSpace {number | [qual.]integer–variable | [qual.]integer–array–element}	Position a sequentially accessed external file at the beginning of the previous record.
CLOSE {number | [qual.]integer–variable | [qual.]integer–array–element}	Disconnect a sequential external file from an input or output unit.
COLOR	Allows you to select color, highlighting and intensity on the debug panel. Valid only under ISPF Version 2.
DEscribe {[qual.] name | * | (name list)} [Print]	List data types of variables and dimension information for arrays.
ENDDEBUG [SAMPLE[(msecs)] [MAXSAMP(n[,STOP])] [CALLED]]	1- Terminate debugging and continue program execution. 2- Initiate program sampling.
ENDFile {number | [qual.]integer–variable | [qual.]integer–array–element}	Write an end-of-file record on a sequentially accessed external file.
ERror {error | error:error | (error list)} [**Msg** | NOMsg] [**Exit** | NOExit]	Select diagnostic options for execution errors.
Fixup [ARG1(value)] [ARG2(value)]	Specify corrective action.
GO [[qual.] {number | EXIT}]	Resume execution.
HALT [Off | Stmt | Goto | Entry | **Immed**]	Continue execution until a specified condition is reached.
Help *(ISPF Version 2)* [command]	Request online information about a command.
Help *(CMS)* [command [(**ALL** | (DESC | (PARM | (FORM]]	Request online information about a command.

Command and Syntax	Function
Help *(TSO)* [command] [**ALL** | FUNCTION | SYNTAX] [OPERANDS [(keyword list)]]	Request online information about a command.
IF (condition) command	Test a condition. *Note:* Check the accompanying table for valid conditions.
List {[qual.]name [:[qual.]name] | * | 'string' | number | specification list)} [Print] [Format [(code)] | Dump [(code)]]	Display values of variables. *Note:* Check the format and dump codes table. See Figure 26 on page 127.
ListBrks [Print]	List all breakpoints and WHEN conditions currently set, and the current HALT status.
ListFreq [[qual.]{number[:[qual.]number] | ENTRY | EXIT} | (number/ENTRY/EXIT list)] [Zerofreq] [Print]	List the number of times statements have been executed.
LISTINGS	Display the listings data set specification panel under ISPF Version 2.
Format 1 LISTSAMP {[qual.]number[:[qual.]number] | [qual.]ENTRY | [qual.]* | (specification list) | *.*} [**DIRECT**][CALLED][ALL] [TOP[(n)]][PRINT] *Format 2* LISTSAMP {unit name | (unitname list) | *} SUMMARY [**DIRECT**][CALLED][ALL] [TOP[(n)]][PRINT]	*Format 1* List sampling counts by statement. *Format 2* List sampling counts by program unit.
ListSubs [Print]	List information about all debuggable program units in the executing load module.

Command and Syntax	Function
ListTime [Print]	Display timing information for all program units.
MoveCurs	Move the cursor between the primary command line and the source window. Valid only under ISPF Version 2.
Next	Set a temporary breakpoint at the next executable statement that has a debugging hook.
OFF [qual.] {number [:[qual.]number] | ENTRY | EXIT} | * | (number/ENTRY/EXIT list)	Turn off breakpoints.
OFFWN condition name | * | (condition name list)	Turn off WHEN monitoring.
POSition number	Position the cursor at a given ISN or sequence number. Valid only under ISPF Version 2.
PREVdisp	Redisplay the previous panel displayed by the application program. Valid only under ISPF Version 2.
PROFILE	Display a profile panel to change current conditions or profile settings. Valid only under ISPF Version 2.
PURGE	Purge output.
Qualify [program]	Change the current default program unit name.
QUIT	End the debugging session.
RECONNECT {number | [qual.]integer–variable | [qual.]integer–array–element}	Reconnect a closed external file.
REFRESH [ON | OFF]	Control whether the IAD panel is refreshed. Valid only under ISPF.
RESTART	Restarts the debugging session while maintaining the log file. Valid only under ISPF.
REWind {number | [qual.]integer–variable | [qual.]integer–array–element}	Position a sequentially accessed external file at the beginning of its first record.

Command and Syntax	Function
SEARCH [/string[/]]	Search the source listing window or the scrollable log for a given character string. Valid only under ISPF Version 2.
Set [qual.]name=value[,value...]	Assign values to variables.
STep [number]	Execute one or more statements, then suspend execution. Under ISPF Version 2, execution is animated.
SYScmd [system–command]	Execute system commands during debugger execution.
TERMIO [IAD | Library] [Msg [(userid)] | Nomsg]	Select I/O routines for terminal I/O from the VS FORTRAN program.
TIMER {* | program-unit-name | (program-unit-name list)} [ON | OFf | Reset]	Control program unit timing.
Trace [Goto | Entry | Off] [Print]	Trace statement branches and subprogram calls.
WheN condition-name [(condition) | variable]	Set up monitoring of a condition.
Where [Trback] [Flow] [Print]	Display statement at which execution is suspended.
WINDOW [ON | OFF]	Open or close the source listing window, or define the window so the lower left corner is at the cursor position. Valid only under ISPF Version 2.

Appendix C. Interactive Debug Messages

All the following VS FORTRAN Version 2 Interactive Debug messages consist of a message number and text. The message number is composed of three parts:

1. A prefix, AFF
2. A 3-digit number
3. A suffix, which indicates the level of the message:
   - I - informational
   - W - warning
   - E - error
   - A - action required

You can control whether or not the message numbers are displayed. Under CMS, use the SET EMSG command. Under TSO, use the PROFILE MSGID command.

An Explanation is provided for each message, and, in many cases, a User Response and System Action are also presented. Generally, the User Response for many of the messages is to reenter the command, correcting the problem that was identified in the error message.

Similarly, the System Action for many of the messages is to issue the VS FORTRAN Version 2 Interactive Debug prompt (FORTIAD) in line mode, or redisplay the screen in full screen mode, and await entry of another command.

In this appendix, a User Action and/or System Action is shown only if it differs from the above general actions.

AFF000E THIS MESSAGE IS RESERVED FOR FUTURE USE; INFORM IBM

Explanation: This message should never occur. If it does, it is the result of a programming error within VS FORTRAN Version 2 Interactive Debug.

User Response: Contact your IBM representative.

AFF001A FORTIAD

Explanation: This is the standard prompt displayed when executing VS FORTRAN Version 2 Interactive Debug in line mode (that is, not under ISPF).

User Response: Enter any VS FORTRAN Version 2 Interactive Debug command.

AFF002A IAD/E

Explanation: This is the prompt displayed when execution is suspended during an error exit for an error detected by the VS FORTRAN Version 1 or VS FORTRAN Version 2 Library.

User Response: When appropriate, use the FIXUP command to supply corrected values for invalid arguments. Use the GO command to cause standard corrective action to be taken.

AFF003A IAD/A

Explanation: This is the prompt displayed when execution is suspended because of an attention interrupt. When this prompt is displayed, processing is within the VS FORTRAN Version 2 Interactive Debug attention exit.

User Response: Enter an Interactive Debug command or a null line.

AFF010I VS FORTRAN VERSION 2.2 INTERACTIVE DEBUG

Explanation: This message is issued at the first hook in the program to identify the Interactive Debug product and the release level.

User Response: None required.

AFF011I (C) COPYRIGHT IBM CORP. 1985, 1987

Explanation: This message is issued at the first hook in the program.

User Response: None required.

AFF012I ALL RIGHTS RESERVED

Explanation: This message is issued at the first hook in the program.

User Response: None required.

AFF013I LICENSED MATERIALS - PROPERTY OF IBM

Explanation: This message is issued at the first hook in the program.

User Response: None required.

AFF020E INTERNAL IAD ERROR *number*. FORTRAN PROGRAM MAY HAVE MODIFIED IAD STORAGE

Explanation: This message is issued when an internal error occurs.

User Response: Make sure the application program has not modified Interactive Debug storage. If it has not, contact your IBM representative.

AFF100E "PRINT" OPTION NOT PERMITTED

Explanation: The AUTOLIST command was entered with the PRINT option.

AFF101E "*name*" IS ONLY VALID IN FULLSCREEN MODE UNDER ISPF VERSION 2 OR HIGHER; COMMAND IGNORED

Explanation: The COLOR, LISTINGS, MOVECURS, POSITION, PREVDISP, PROFILE, SEARCH, or WINDOW command was entered using line mode or batch mode.

User Response: Be sure you are running under ISPF Version 2 before issuing this command.

AFF102E "*name*" MUST SPECIFY A COMMAND IN BATCH MODE

Explanation: A SYSCMD, CMS, or TSO command was entered with no operand while operating in batch mode.

User Response: Correct the problem and resubmit the job. If you want to issue a sequence of system commands, you must enter separate SYSCMDs.

AFF103E "*name*" IS SUPPORTED ONLY IN FULLSCREEN MODE; COMMAND IGNORED

Explanation: AUTOLIST, REFRESH, or RESTART was entered in line mode or batch mode.

User Response: Be sure you are running under ISPF before entering this command.

AFF111I PROGRAM TERMINATED EARLY BECAUSE MAXIMUM COUNT WAS REACHED

Explanation: The maximum count value specified in the sublist for the MAXSAMP keyword of the ENDDEBUG command was reached. STOP was also specified which caused the program to be terminated.

AFF112E INTERVAL TIMER WAS RESET BY USER PROGRAM, THUS CANCELLING SAMPLING

Explanation: A non-VS FORTRAN routine called by a VS FORTRAN program performed an STIMER macro, resetting the STIMER set by IAD program sampling.

System Action: Program sampling was discontinued.

AFF121E THE AFFON STATEMENT RESTRICTION LIST WILL BE IGNORED FOR "*name*" BECAUSE IT WAS COMPILED WITH THE "TEST" OPTION

Explanation: You cannot restrict statement hooks if the program unit is compiled with "TEST" because the compiler inserts the hooks.

User Response: Remove the statement restriction list, or recompile the program unit.

System Action: The restriction list is ignored.

AFF122E THE AFFON STATEMENT RESTRICTION LIST FOR "*name*" CONTAINS AN INVALID RANGE, WHICH WILL BE TREATED AS A SINGLE ISN

Explanation: If an invalid range syntax is specified in the AFFON file, only the first ISN will be considered.

User Response: Correct the restriction list, and rerun the job.

System Action: The second ISN is ignored.

AFF123E THE AFFON STATEMENT RESTRICTION LIST FOR "*name*" CONTAINS INVALID SYNTAX AND WILL BE IGNORED

Explanation: If invalid syntax such as alphabetic characters are specified in the AFFON file, the entire list for the associated program unit will be ignored.

User Response: Correct the restriction list, and rerun the job.

System Action: The restriction list is ignored.

AFF124E THE AFFON STATEMENT RESTRICTION LIST FOR "*name*" CONTAINS AN ENTRY THAT EXCEEDS THE MAXIMUM POSSIBLE ISN; THE MAXIMUM IS ASSUMED

Explanation: An ISN greater than 16777215 was specified in the AFFON restriction list.

User Response: Correct the restriction list, and rerun the job.

System Action: The entry is treated as 16777215.

AFF190E ATTEMPT TO REFERENCE INACCESSIBLE STORAGE

Explanation: Interactive Debug tried to examine storage in an area where it was not allowed to look. This was probably caused by either an invalid address entered as part of a LIST command, or an invalid address within a VS FORTRAN module.

User Response: If you specified an invalid address on a LIST command, reissue the command with valid addresses. If not, try to determine the cause of the invalid address in the program.

AFF191E END STATEMENT SAME AS EPILOG IN PROGRAM *program*; INTERNAL ERROR

Explanation: This indicates an internal error and should not occur. The statement table entry in a VS FORTRAN Version 1 program for an END statement points to the epilog processing routine.

User Response: Contact your IBM representative. Debugging may be continued, however.

System Action: The statement is treated as a collapsed statement.

AFF192I CURRENT HALT STATUS: *status*

Explanation: This message tells you whether HALT has been issued to indicate when execution is to be suspended. Possible status is STMT, GOTO, or ENTRY.

User Response: None required. This is an informational message.

AFF194E I/O IS ALREADY ACTIVE; COMMAND IGNORED

Explanation: An attempt to issue BACKSPACE, CLOSE, ENDFILE, RECONNECT or REWIND, has been detected while I/O was already active.

User Response: If you need to issue one of these commands, issue a NEXT command so that execution will be suspended after the I/O event is completed.

AFF195E NO DEBUGGABLE FORTRAN PROGRAMS WERE FOUND

Explanation: Interactive Debug has not found any programs within the module to be executed that are debuggable program units. In general, for a program unit to be considered debuggable, it must have been compiled with the SDUMP option.

User Response: The problem may be because of an incorrectly specified AFFON file. If so, you should have received other messages detailing other errors.

AFF196E THERE IS NO PRINT DATA SET DEFINED

Explanation: An error has occurred while attempting to open the AFFPRINT data set.

User Response: Correct the cause of the error.

System Action: Processing continues. You will not be able to use the PRINT keyword on any Interactive Debug command.

AFF198E RELEASE 4.0 LIBRARY TRANSFERRED TO IAD IN 24-BIT ADDRESSING MODE, CONTACT IBM

Explanation: This indicates an internal library error, or an installation or system error that should not occur.

User Response: Contact your IBM representative.

AFF199E RELEASE 3.1 LIBRARY TRANSFERRED TO IAD IN 31-BIT ADDRESSING MODE

Explanation: This indicates a link-edit error, an internal library error, or an installation or system error.

User Response: Relink-edit your program with AMODE(24)/RMODE(24). If this does not solve the problem, contact your IBM representative.

AFF200E STORAGE EXHAUSTED; SIMPLIFY THE COMMAND OR REMOVE SOME BREAKPOINTS

Explanation: While attempting to build internal control blocks to represent a command, Interactive Debug used up all available storage.

User Response: If possible, issue a less complicated command, or reinvoke Interactive Debug with more virtual storage (VM) or a larger user region (MVS).

AFF210E STORAGE EXHAUSTED DURING SAMPLING, ENTRY POINT SAMPLING INFORMATION WILL BE INCOMPLETE

Explanation: IAD was not able to obtain storage for recording the sampling counts for some entry points of nondebuggable VS FORTRAN routines, VS FORTRAN math. library routines, or non-VS FORTRAN routines. Counts that would have normally been categorized by entry point are grouped together in the *UNKNOWN count.

User Response: Reinvoke Interactive Debug with more virtual storage (VM) or a larger user region (MVS).

AFF220I *synad message*

Explanation: This is the error message returned by the operating system when an attempt was made to write to the AFFPRINT data set.

User Response: Correct the error that caused the message. If printed output is required, after correcting the problem, reinvoke Interactive Debug.

AFF224I "*count*" LINES OF OUTPUT WRITTEN TO AFFPRINT BY "*command*"

Explanation: Confirms that information was written to the print file, following a command that was specified with the print option.

AFF225E ERROR WRITING THE PRINT DATA SET; SUBSEQUENT OUTPUT WILL BE WRITTEN TO THE TERMINAL

Explanation: An error was detected when attempting to send "printed" output to the AFFPRINT data set. From this point on, output that would be destined for the print data set is redirected to the terminal.

User Response: If you need to have the print output in a print data set, terminate your debugging session, correct the error, and reinvoke the program.

System Action: Further print output is sent to the terminal.

AFF226E ERROR WRITING THE PRINT DATA SET; SUBSEQUENT OUTPUT WILL BE DISCARDED.

Explanation: An error was detected when attempting to send "printed" output to the AFFPRINT data set in batch mode under TSO. From this point on, output that would be destined for the print data set is discarded.

User Response: If you must have the print output in a print data set, terminate your debugging session, correct the error, and reinvoke the program.

System Action: Further print output is discarded.

AFF229E INVALID COUNT VALUE SPECIFIED IN "*number*"

Explanation: A count value larger than 65535 was specified on the AT command.

User Response: Specify a smaller count value, and reissue the AT command.

AFF230E NO BREAKPOINTS CAN BE SET AT STATEMENT "*number*" BECAUSE IT IS COLLAPSED

Explanation: The indicated statement occupies no storage so a breakpoint cannot be set.

User Response: Set your breakpoint at a statement before or after the indicated statement. You can use LISTFREQ to see which statements have hooks.

AFF231E NO BREAKPOINT CAN BE SET AT STATEMENT "*number*" BECAUSE THERE IS NO HOOK THERE

Explanation: The specified statement was not included in the AFFON file restriction list, or is an ENTRY or EXIT of a main program unit.

User Response: None required. You can use LISTFREQ to see which statements have hooks.

AFF240W A SUBSCRIPT IS OUT OF RANGE IN "*array element*"

Explanation: A subscript has been specified for the indicated array element that exceeds the dimension specified when the array element was defined.

AFF241W WARNING: A SUBSTRING BOUNDARY IS OUT OF RANGE IN "*string*"

Explanation: A substring value has been specified which is outside the defined variable length.

System Action: The command is executed as normal.

AFF242E "*name*" CANNOT BE ACCESSED; IT COULD BE IN AN UNINITIALIZED DYNAMIC COMMON

Explanation: Variables in a dynamic common cannot be accessed until the common has been initialized and the address has been obtained for the qualifying VS FORTRAN program unit. This occurs the first time the program unit is entered.

User Response: Set a breakpoint at some point that will be reached after the program unit is entered, and access the variables when you get there. If the dynamic common has been initialized, you may be

able to access it using a different program unit that has already been entered at least once.

AFF245E "WHERE" INFORMATION IS NOT AVAILABLE AFTER "ENDDEBUG" IS ISSUED

Explanation: ENDDEBUG has been issued and WHERE information cannot be determined.

AFF292E LISTING FILE "*dsname*" CANNOT BE READ

Explanation: Issued when a read error occurs while attempting to annotate a listing. Can occur due to an actual read error or unexpected file format.

User Response: Insure that the file or data set is sequential or is a PDS member and that the LRECL is not greater than 151.

AFF293E AN ARRAY WAS USED WHERE A SCALAR IS REQUIRED IN "*variable*"

Explanation: While scanning the syntax of the previous command, an array variable was found when a scalar variable was required.

AFF294E A SIGN WAS SPECIFIED IN "*text*," BUT THE VARIABLE IS NOT A NUMERIC SCALAR

Explanation: While scanning the previous command, a sign (+ or -) was specified for a nonnumeric variable. Only numeric variables may have signs.

AFF295E THERE IS NO ROOM TO INSERT A HOOK IN STATEMENT "*name.number*"; STATEMENT TREATED AS COLLAPSED

Explanation: This is an internal error and should not occur. It indicates that the VS FORTRAN Version 1 or VS FORTRAN Version 2 compiler only allocated two bytes for a VS FORTRAN statement. The compiler should allocate at least four bytes so that an Interactive Debug hook can be inserted.

User Response: Contact your IBM representative. Debugging may be continued, however.

System Action: The statement is treated as a collapsed statement, and you will not be allowed to set a breakpoint at the statement.

AFF296E THE AFFON FILE CANNOT BE READ; FILE IGNORED

Explanation: An I/O error occurred trying to access the AFFON file.

User Response: Correct the cause of the I/O error if you want the AFFON file to be read.

System Action: The AFFON file is ignored and processing continues.

AFF297E AN ATTEMPTED BRANCH TO LOW STORAGE OCCURRED AT "*name.number*"; IT MAY BE A CALL TO A NONEXISTENT PROGRAM

Explanation: An attempted branch to low storage has occurred. This is frequently caused by a call to a missing program unit.

User Response: Determine the cause of the error. Perhaps the required program unit was not found at link-edit or load time.

System Action: Control is returned to the calling program as if the called program only contained a RETURN statement.

AFF298I FORTRAN DEBUG PACKET STATEMENTS IN PROGRAM "*program*" WILL BE TREATED AS COLLAPSED

Explanation: For programs compiled prior to VS FORTRAN Version 1 Release 4, Interactive Debug has detected static debug packet statements within the program. Interactive Debug will operate with programs containing debug packets, but debugging of the packets themselves is not supported.

User Response: None required.

System Action: The statements within the debug packet are treated as collapsed statements, and debugging will not be possible at those statements.

AFF299E ERROR WRITING AFFOUT FILE; FILE IGNORED

Explanation: An I/O error has occurred while attempting to write to the AFFOUT data set. No further attempts will be made to access the file.

User Response: Correct the problem that caused the message. If the log is required, reinvoke Interactive Debug after correcting the problem.

AFF300I AT: *name.number*

Explanation: Execution has been suspended at the identified statement in the identified program unit. It is suspended because a prior AT command requested a breakpoint at this statement.

User Response: Enter debugging commands, or GO to resume execution.

AFF301I NEXT: *name.number*

Explanation: Execution has been suspended at the identified statement in the identified program unit. It is suspended because a NEXT or STEP command was issued.

User Response: Enter debugging commands, or GO to resume execution.

AFF303I TRACE STATUS: status

Explanation: Provides the current trace status in response to the TRACE command with no operands.

AFF304I TRACE: FROM *name.number* TO *name.number*

Explanation: Execution has passed from the first identified statement to the second identified statement. The second statement did not immediately follow the first statement in the VS FORTRAN source. This message is received because of an earlier TRACE command that was issued.

User Response: None required.

AFF305W "ERROR" COMMAND TERMINATED AFTER PROCESSING ERROR NUMBER *number*

Explanation: The PURGE command was used to terminate excessive output from an ERROR command. The last error number processed was "number."

User Response: None required.

AFF306I PROGRAM HAS TERMINATED; RC=(*code*)

Explanation: The application program being executed has completed. If a return code was coded on the STOP statement, it is provided.

User Response: None required.

System Action: You will be allowed to continue entering commands until a QUIT command is entered, at which time the debugging session will be terminated.

AFF307W COMMAND OUTPUT REFLECTS THE STATE OF EXECUTION PRIOR TO ENTERING "ENDDEBUG"

Explanation: The information presented as output for the command which was just issued, is not necessarily current information. It was correct when you issued an ENDDEBUG command earlier in the debugging session, and has not been updated since ENDDEBUG was issued.

User Response: None required.

AFF400I THE COMMAND LIST FOR THE BREAKPOINT AT "*name.number*" HAS BEEN TERMINATED BY AN ATTENTION

Explanation: Because of entering an attention exit, the indicated command list cannot be completed.

User Response: If necessary, enter the commands that were not completed.

AFF405E THE NONIMMEDIATE COMMAND "*command*" WAS IGNORED DUE TO A PENDING ERROR EXIT

Explanation: The indicated command could not be executed now because it is not a command that can be immediately processed (that is, processed easily by issuing a message, like WHERE, or by setting a flag, like NEXT), and cannot be deferred to later processing because execution is currently within an error exit.

User Response: Issue a NEXT, and then issue the command after execution has left the error exit.

AFF410E UNKNOWN COMMAND

Explanation: The syntax of the previous "command" name was not a valid Interactive Debug command.

User Response: Check your spelling of the command name for accuracy or insure that you have included the SYSCMD command for a system command.

AFF450E "*name*" IS AN ASSUMED SIZE ARRAY; SUBSCRIPTS MUST BE SPECIFIED FOR "LIST"

Explanation: A LIST or SET command was issued for the specified array. However, the final dimension of the array is unknown because it was not defined at compile time. For example, B(*) may have been specified. Interactive Debug will not LIST or SET values in an assumed size array unless a specified element (for example, B(3)) is specified.

AFF454E "GOTO" OR "ENTRY" MUST BE SPECIFIED WITH "PRINT"

Explanation: GOTO or ENTRY was not specified with the PRINT option on the TRACE command.

User Response: Reissue the TRACE command with either the GOTO or ENTRY options.

AFF455E INVALID OPERAND SYNTAX SPECIFIED IN "*text*"

Explanation: Invalid syntax has been specified for the indicated command operand.

AFF456E "*word*" IS NOT A VALID KEYWORD FOR THE "*cmnd*" COMMAND

Explanation: Issued when an invalid keyword option is detected in an IAD command.

AFF457E *"abbr"* **IS AN AMBIGUOUS KEYWORD ABBREVIATION FOR THE** *"cmnd"* **COMMAND**

Explanation: Issued when an ambiguous abbreviation is detected in an IAD command.

User Response: Use a longer abbreviation for the keyword.

AFF458E **KEYWORD** *"word"* **WAS SPECIFIED MORE THAN ONCE FOR THE** *"cmnd"* **COMMAND**

Explanation: Issued when an option keyword is specified more than once in an IAD command.

User Response: Specify the option keyword only once.

AFF459E *"word1"* **AND** *"word2"* **CANNOT BOTH BE SPECIFIED ON THE** *"cmnd"* **COMMAND**

Explanation: Conflicting keywords are specified.

User Response: Reissue the command with only one of the keywords.

AFF460E **UNBALANCED DELIMITERS SPECIFIED IN** *"text"*

Explanation: While scanning the syntax of the previous command, an invalid use of delimiters was detected. Usually, a right or left parenthesis is missing.

AFF461E **INVALID SUBLIST SYNTAX IN** *"text"*

Explanation: Issued when incorrect syntax is detected in a parenthesized sublist following a keyword.

AFF462E **KEYWORD** *"word"* **OF THE** *"cmnd"* **COMMAND REQUIRES A SUBLIST**

Explanation: Issued when a parenthesized sublist was not specified for a keyword that requires one.

User Response: Reissue the command with the required sublist.

AFF463E **KEYWORD** *"word"* **OF THE** *"cmnd"* **COMMAND DOES NOT ALLOW A SUBLIST**

Explanation: Issued when a parenthesized sublist was specified for a keyword that does not permit one.

AFF464E **SUBLIST VALUE** *"number"* **IS TOO LARGE FOR KEYWORD** *"word"*

Explanation: The value specified in the sublist is larger than the maximum allowed.

AFF470E UNKNOWN COMMAND "*text*"

Explanation: The indicated string was found in a command list, but is not recognized as a valid Interactive Debug command.

AFF481E THE DESTINATION CANNOT BE BRANCHED TO IN "*text*"

Explanation: A GO command was entered that references a VS FORTRAN statement that has no hook. This includes a GO EXIT for a VS FORTRAN Version 1 MAIN program.

User Response: You can use LISTFREQ to see which statements have hooks.

AFF483E "GO" WITH A STATEMENT IDENTIFIER CANNOT BE ISSUED FROM AN ENTRY

Explanation: A GO command with a statement identifier cannot be issued from the entry point of any VS FORTRAN program unit. At entry, the unit is not yet active.

User Response: Issue `STEP 1` to get to the first statement with a hook. You should be able to issue the GO command from there.

AFF484E CONTINUATION IS NOT PERMITTED WITH THE "*command*" COMMAND

Explanation: A command valid only under ISPF Version 2 was entered under ISPF Version 2, but with continuation. Continuation is not supported for these commands.

AFF485E "*cmnd*" COMMAND CANNOT BE ISSUED FROM AN ATTENTION EXIT

Explanation: A full screen display command is not allowed from an attention exit.

User Response: Exit the attention mode and reissue the command.

AFF486E "*cmnd*" COMMAND CANNOT BE ISSUED FROM A RESTART FILE

Explanation: A full screen display command is not allowed in a RESTART file.

System Action: The command is ignored.

AFF500E STATEMENT *number* IS NOT EXECUTABLE

Explanation: The statement reference does not identify an executable statement. Possibly the current qualification identifies a program unit that does not contain a statement with the specified number.

User Response: Use the QUALIFY command to set the proper qualification, or select a different statement.

AFF510E **INVALID RANGE; THE RIGHT SIDE (*number*) IS LESS THAN THE LEFT SIDE (*number*)**

Explanation: If a command has been entered with a range of statements specified, the first statement must appear prior to the second in the program.

AFF511E **INVALID SUBSTRING RANGE; THE RIGHT SIDE IS LESS THAN THE LEFT SIDE IN "*string*"**

Explanation: The value specified on the right side of the substring notation is larger than the value specified on the left side.

AFF520E ***program*** **IS NOT A DEBUGGABLE FORTRAN PROGRAM UNIT**

Explanation: The indicated program unit cannot be debugged.

1. If it is a VS FORTRAN Version 1 or VS FORTRAN Version 2 program, it probably was compiled with the NOSDUMP option. Either SDUMP or TEST must be specified (or defaulted) for the program unit to be eligible for debugging.

2. A valid AFFON file was found but did not contain this program unit.

User Response: If you want to debug the indicated program unit, terminate the current debugging session, using the QUIT command, correct the AFFON file or recompile the program unit with the appropriate compiler options, and reexecute the program.

AFF530E **"*text*" IS INVALID "FIXUP" SYNTAX**

Explanation: Invalid keyword syntax has been detected for a FIXUP command. The only valid keywords are ARG1 and ARG2 and both must contain a value within parentheses following the keyword.

AFF535E **NONNUMERIC VALUE "*text*" IS NOT ALLOWED IN FIXUP**

Explanation: A logical or character value has been specified in a FIXUP command as value for either ARG1 or ARG2. Only numeric values are valid.

AFF540E **FORTRAN TERM "*text*" IS NOT ALLOWED IN FIXUP**

Explanation: Usually, valid syntax has been detected where it is not allowed in a FIXUP command (for example, a duplication factor).

AFF545E **A NULL FORTRAN TERM IS INVALID**

Explanation: One of the sides of a range specification is missing (for example, "LIST A(1):").

AFF549E PROGRAM SAMPLING REQUIRES VS FORTRAN VERSION 2 RELEASE 2.0 LIBRARY OR LATER

Explanation: The module being debugged was link-edited with an older release of the VS FORTRAN library. Interactive Debug needs Version 2 Release 2.0 in order to perform program sampling.

User Response: Relink-edit the program with the latest release of the library.

AFF550I PROGRAM SAMPLING INTERVAL WAS *m* MS; TOTAL NUMBER OF SAMPLES WAS *n*

Explanation: This is the header message for a LISTSAMP display, where "m" is the sampling time interval used and "n" is the total number of sampling interruptions that occurred.

AFF551I "*type*" SAMPLES:

Explanation: The sampling counts that follow are for sampling interruptions of the type indicated.

AFF552I SUM OF DIRECT AND CALLED SAMPLES:

Explanation: The counts that follow are the sum of both DIRECT and CALLED counts.

AFF553E "*unitname*" CANNOT BE ANNOTATED BECAUSE ITS LISTING FILE IS NOT KNOWN

Explanation: Issued when the listing file has not been identified for a program unit that has had annotation requested for it.

User Response: Specify the file or data set name in the AFFON file, or use the LISTINGS panel under ISPF Version 2.

AFF554E "*unitname*" CANNOT BE ANNOTATED BECAUSE IT WAS NOT FOUND IN "*dsname*"

Explanation: Issued when the listing for a specified program unit cannot be found in the specified listing data set.

User Response: Be sure the correct file or data set name is specified in the AFFON file.

AFF555I STATEMENT SAMPLES %UNIT %TOTAL

Explanation: This is the title line for a LISTSAMP output when statement information is listed.

AFF556I PROGRAM UNIT SAMPLES %TOTAL

Explanation: This is the title line for LISTSAMP output when summary information is listed.

AFF557I **statement samples %unit %total histogram**

Explanation: This is the program sampling information for a statement, showing statement number, number of samples, percentage of total samples for the program unit, percentage of total samples for the entire program, and a histogram (bar chart) that graphically displays the size of this value relative to the other sampling values listed.

AFF558I **program samples %total histogram**

Explanation: This is the program sampling information for a program unit, showing program unit name, number of samples, percent of total samples, and a histogram (bar chart).

AFF559E **"*unitname*" CANNOT BE ANNOTATED BECAUSE IT WAS NOT COMPILED WITH VS FORTRAN V2**

Explanation: Issued when annotation is requested for a program unit that was not compiled with VS FORTRAN VERSION 2. Annotation is not supported for listings produced by previous versions of the compiler.

AFF560E **PROGRAM SAMPLING HAS NOT BEEN DONE; ISSUE "ENDDEBUG" WITH THE "SAMPLE" OPTION TO INITIATE PROGRAM SAMPLING**

Explanation: Issued when LISTSAMP or ANNOTATE with the SAMPLING option is entered but program sampling has not been initiated.

AFF561I **ANNOTATING LISTING FOR PROGRAM UNIT "*unitname*"**

Explanation: An informational message indicating the progress of the ANNOTATE command.

AFF562I **samples %unit %total histogram**

Explanation: This is program sampling information for a statement, showing number of samples, percentage of total samples for the program unit, percentage of total samples for the entire program, and a histogram (bar chart) that graphically displays the size of this value relative to the other sampling values listed. This information will appear on the line below the statement number, in those cases where these fields and the statement number are too long to display together on a single line.

AFF563I **VS FORTRAN INTERACTIVE DEBUG V2 R2.0 ANNOTATED LISTINGS:**

Explanation: This is the heading line for annotated listings.

AFF564I PROGRAM UNIT PAGE %TOTAL DISTRIBUTION

Explanation: This is the heading line for the summary page of annoted listings.

AFF565I unitname page percent histogram

Explanation: This is the format of the output of the summary lines shown on the summary page of annotated listings.

AFF566I ANNOTATE: *status*

Explanation: Indicates the current status of the ANNOTATE controls that are used for displaying bar charts on the source listing window.

AFF567E "CALLED" IS NOT VALID UNLESS SAMPLING WAS INITIATED WITH THE "CALLED" OPTION

Explanation: The CALLED keyword must be specified in the ENDDEBUG command in order for CALLED to be valid in the ANNOTATE and LISTSAMP commands.

User Response: Reissue the command without the CALLED option, or restart the debugging and specify the CALLED option, in ENDDEBUG, when initiating sampling.

AFF568I FREQUENCIES

Explanation: Heading for annotated listing by program unit when frequency values are displayed.

AFF569E "*word***" KEYWORD IS NOT PERMITTED WITHOUT THE "SAMPLE" KEYWORD**

Explanation: The MAXSAMP or CALLED keywords are only valid if SAMPLE has been specified for the ENDDEBUG command.

AFF570E TIMING INTERVAL MUST BE AN INTEGER LARGER THAN ZERO

Explanation: The sample time specified in the SAMPLE sublist of the ENDDEBUG command must be an integer value larger than zero.

AFF571E INVALID STOP PARAMETER SPECIFIED IN "MAXSAMP" SUBLIST

Explanation: STOP, or an abbreviated form of STOP, was incorrectly specified in the MAXSAMP sublist of the ENDDEBUG command.

AFF572E NO SUBLIST SPECIFIED FOR THE "*word***" KEYWORD**

Explanation: A sublist is required with the indicated keyword.

AFF573E **"*number*" IS AN INVALID "*word*" VALUE**

Explanation: The numeric value specified in the keyword sublist is too large.

AFF574E **MAXIMUM NUMBER OF SAMPLING INTERRUPTS MUST BE AN INTEGER GREATER THAN ZERO**

Explanation: The maximum number of sampling interrupts specified in the MAXSAMP sublist of the ENDDEBUG command must be an integer value greater than zero.

AFF575E **"*text*" IS NOT A VALID SUBLIST FOR "*word*"**

Explanation: A sublist after the keyword on a command contains invalid syntax.

AFF576E **"DUMP" IS NOT PERMITTED WITH A CONSTANT OPERAND**

Explanation: LIST or AUTOLIST has been specified with both a constant operand and the dump option. This combination is not permitted.

AFF577E **THE MINIMUM SAMPLING INTERVAL ON CMS IS 4 MILLISECONDS.**

Explanation: When using CP timer assist, the minimum accuracy of the interval timer on CMS is about 3.3 milliseconds. To prevent sampling interruptions from occurring in the operating system code servicing of the interruption, the interval time is restricted to 4 milliseconds or greater.

AFF605E **ONLY REALS ALLOWED IN COMPLEX CONSTANT "*text*"**

Explanation: Integers have been used as part of a complex constant. Only real numbers are allowed.

AFF610E **REAL AND IMAGINARY PARTS OF "*variable*" DIFFER IN LENGTH**

Explanation: One part of a complex constant has been entered as a REAL*4 number and the other part has been entered as a REAL*8 number. Both parts must be of equal length.

AFF615E **"*text*" IS INVALID FORTRAN TERM SYNTAX**

Explanation: While scanning the previous command, an operand that must be a VS FORTRAN term had invalid syntax.

AFF620E **SUBSCRIPTS ARE NOT PERMITTED ON "*variable*"; THE VARIABLE IS NOT AN ARRAY**

Explanation: Subscripts have been specified for a variable that is not an array.

AFF625E THE NUMBER OF SUBSCRIPTS ON "*array*" DOES NOT MATCH THE DECLARED NUMBER OF DIMENSIONS

Explanation: There are too many or too few subscripts specified for the indicated array.

AFF630E "*variable*" IS AN INVALID FORTRAN VARIABLE NAME

Explanation: The name of a VS FORTRAN variable is invalid. For example, the name contains more than six characters, or does not begin with an alphabetic character.

AFF631E PROGRAM UNIT "*name*" IS NOT ACTIVE; "*varname*" IS A DUMMY ARGUMENT AND CANNOT BE ACCESSED

Explanation: An AUTOLIST, LIST, SET, IF, or WHEN command attempted to access variables that have no storage because the program unit is not active.

User Response: Set a breakpoint within the program unit and access the variables when you get there. If the dynamic common has been initialized, you may be able to access it using a different program unit.

AFF632E "*varname*" IS A DUMMY ARGUMENT THAT IS NOT DEFINED AT THE ENTRY POINT BY WHICH "*program*" WAS ENTERED, AND CANNOT BE ACCESSED

Explanation: An attempt was made to reference a dummy argument that is defined only in an alternate entry point.

User Response: Wait until the program unit is entered by an entry point that defines the dummy variable you want to access.

AFF633E "QUIT" HAS BEEN ISSUED. ENTER "QUIT" AGAIN TO FORCE AN ABNORMAL TERMINATION

Explanation: This message is issued by the attention interrupt handler.

User Response: Enter QUIT again if you want to terminate debugging; otherwise, enter any command that is appropriate in an attention exit.

AFF634E "H" IS NOT A VALID ABBREVIATION UNDER ISPF; USE "HELP"

Explanation: The only way to request HELP information when executing VS FORTRAN Version 2 Interactive Debug under ISPF is to type "HELP" and press the Enter key, or to press a PF key that has been assigned to HELP (normally PF1).

User Response: If HELP information is desired, request it, using one of the two techniques described in the Explanation.

AFF635E VARIABLE "*variable*" IS NOT IN PROGRAM UNIT *program*

Explanation: The specified variable was not found in the specified program unit. The variable may be misspelled, the program qualification may be missing or incorrect, or the variable may have been removed by the optimizer.

User Response: Make sure that the variable is defined in the currently qualified program unit. If not, then either use the QUALIFY command, or explicitly qualify the variable name in the Interactive Debug command.

AFF636E "COMMAND NOT FOUND"

Explanation: The TSO command requested using the SYSCMD command was not found by TSO.

AFF640E "*text*" IS INVALID CONSTANT SYNTAX

Explanation: There is a syntax error in the indicated constant.

AFF645E CONSTANT "*number*" EXCEEDS THE MACHINE CAPACITY

Explanation: The indicated constant is too large. This may occur if leading zeros are specified with the constant.

AFF650E INVALID SUBSCRIPT IN "*text*"; A SUBSCRIPT CANNOT BE AN ARRAY OR ARRAY ELEMENT

Explanation: The subscripts of the indicated array must be scalar constants or variables.

AFF655E ONE OR MORE PARTS OF COMPLEX CONSTANT "*text*" ARE MISSING

Explanation: Either the real or the imaginary portion of the indicated complex constant is missing. Both portions are required.

AFF660E "*text*" HAS INVALID SUBSCRIPT SYNTAX

Explanation: While scanning what appears to be a subscript, invalid syntax was discovered. Possibly the right parenthesis was missing.

AFF665E INVALID SUBSCRIPT IN "*array*"; A NON-INTEGER VARIABLE WAS SPECIFIED

Explanation: A logical, real, character, or complex variable has been used as a subscript for the indicated array. Only integer numbers may be used as subscripts.

AFF670E "*text*" IS INVALID ON THE LEFT SIDE OF A SET COMMAND

Explanation: A duplication factor, a minus sign, or a constant appears on the left side of a SET command. None of these are valid on the left side.

AFF675E "*text*" IS INVALID "SET" SYNTAX

Explanation: Either the equal sign (=) or the right side of the SET assignment has been omitted.

AFF680E "*text*" IS INVALID IN THE "LIST" RANGE

Explanation: An item is specified in a range that is not allowed in a LIST command.

AFF690E VARIABLE "*variable*" IS INVALID IN "*command*" COMMAND

Explanation: A duplication factor, a minus sign (-), or a constant appears with a variable in the indicated command.

AFF691E LITERAL OR NUMERIC CONSTANTS ARE NOT PERMITTED IN THE "DESCRIBE" COMMAND

Explanation: Only variables and array names can be specified on the DESCRIBE command.

AFF700E NULL VARIABLE LIST/RANGE SPECIFIED IN "*text*"

Explanation: While scanning the previous command, the end of the command was found before a list or range specification was completed.

AFF715E "*variable*" IS NOT A LOGICAL VARIABLE

Explanation: While scanning the previous command, a logical variable was expected, but the indicated variable, which is not a logical variable, was found instead.

AFF720E "*condition*" HAS AN INVALID CONDITION

Explanation: The indicated condition is not syntactically correct. For example, ".EE." may have been used instead of ".EQ.."

AFF725E INVALID COMBINATION OF DATA TYPES IN CONDITION "*condition*"

Explanation: While scanning the syntax of the previous command, an invalid combination of data type was found within the condition specification. The data types of the variables must be the same or compatible.

AFF730E **"*condition*" IS INVALID CONDITION SYNTAX**

Explanation: In the specification of an arithmetic condition, either the right side of the condition has been omitted, or some extraneous data follows what appears to be a complete condition.

AFF735E **CONDITION "*condition*" HAS AN INVALID FORTRAN TERM**

Explanation: A duplication factor is specified within a condition specification. This is not valid.

AFF740E **"*text*" IS INVALID "IF" SYNTAX**

Explanation: Invalid syntax has been detected within an IF command. For example, parentheses may be missing around the condition definition.

AFF741E **A USERID MUST BE SPECIFIED WITH THE MSG OPERAND; NO DEFAULT IS AVAILABLE**

Explanation: TERMIO MSG was entered and the default user ID cannot be determined.

User Response: If the MSG operand is desired, specify the desired user ID.

AFF742W **"MSG" OPERAND IGNORED; "MSG" IS NOT VALID OUTSIDE BATCH MODE**

Explanation: A TERMIO command containing a MSG operand was entered while Interactive Debug was being used interactively. The MSG operand is only for batch mode.

AFF743E **"*command*" COMMAND IS NOT PERMITTED IN BATCH MODE**

Explanation: The indicated command was entered while in batch mode, but is not permitted there.

AFF745E **"*text*" IS INVALID "WHEN" SYNTAX**

Explanation: In the specification of an arithmetic condition, either the right side of the condition has been omitted, or some extraneous data follows what appears to be a complete condition.

AFF750E **A COMMAND MUST BE SPECIFIED AFTER "*text*"**

Explanation: No command was specified after the condition on an IF statement.

AFF755E **"QUALIFY" IS NOT PERMITTED IN AN "IF" COMMAND**

Explanation: QUALIFY cannot be the command specified as the action to be taken if the condition specified in an IF command is true.

AFF760E "*text*" REQUIRES AN OPERAND; THE COMMAND IS IGNORED

Explanation: An operand must be specified on the indicated command.

AFF765E "*condition*" HAS AN INVALID "WHEN" CONDITION NAME

Explanation: The name of the indicated condition has a syntactically invalid name. Valid names must begin with an alphabetic character and contain no more than four alphameric characters.

AFF768W "*condition*" CONDITION IS NOT ON

Explanation: An attempt to turn off the indicated condition has been detected, but the condition is already off.

AFF775E "*command*" COMMAND IGNORED IN AN IF OR COMMAND LIST

Explanation: The indicated command was found in an IF command or a command list specified with an AT command. The command is not valid in this context and is ignored.

AFF780E "*command*" IS NOT PERMITTED TO HAVE OPERANDS; THE COMMAND IS IGNORED

Explanation: A keyword has been specified for a command that has no keywords. The extra keyword is ignored.

User Response: Determine why the extra keyword was entered. Possibly the wrong command was issued. If so, issue the right command.

AFF795E INVALID COMBINATION OF DATA TYPES IN "*text*"

Explanation: In a SET command, the data type of the variables and constants on the right side of the equal sign is different from the data type of the variable on the left side of the equal sign.

- A character variable may be assigned only character data items.
- A logical variable may be assigned only logical data items.
- An arithmetic variable may be assigned only arithmetic data items.

AFF800E INVALID "GO"; THE STATEMENT IDENTIFIER IS NOT IN THE CURRENT PROGRAM UNIT

Explanation: An attempt has been detected to go to a statement outside the program unit that was being executed when processing was suspended.

User Response: Reenter the GO command either without an ISN or sequence number, or with an ISN or sequence number that is within the correct program unit.

AFF801I QUALIFICATION IS *program*

Explanation: This message is issued in response to a QUALIFY command, and identifies the currently qualified program unit. Unless you have issued a QUALIFY command to set a different program unit, the currently qualified program unit will be the program unit which is currently being executed.

AFF802E COMMAND IGNORED; THE PROGRAM HAS FINISHED EXECUTION

Explanation: The VS FORTRAN program has finished execution. Interactive Debug will allow you to enter most commands before you enter the QUIT command, but the last command entered is not one of those that may be issued at this time.

User Response: Enter a valid command. Valid commands are:

ANNOTATE	LISTFREQ	QUIT
AUTOLIST	LISTINGS	RECONNECT
BACKSPACE	LISTSAMP	REFRESH
CLOSE	LISTSUBS	RESTART
COLOR	LISTTIME	REWIND
comment	MOVECURS	SEARCH
DESCRIBE	POSITION	SET
ENDFILE	PROFILE	SYSCMD
HELP	PREVDISP	TERMIO
LIST	PURGE	WHERE
LISTBRKS	QUALIFY	WINDOW

For a list of commands that are **not** valid, see "Issuing Commands after Termination of a VS FORTRAN Program" on page 96.

AFF805W *program* IS OPTIMIZED; "GO" WITH STATEMENT ID MAY CAUSE UNPREDICTABLE RESULTS

Explanation: The indicated program unit was compiled with OPT(n) with n>0. Because the program is optimized, it may depend on values being kept in registers between some statements. A GO command to a specific statement may bypass code that is needed to set registers, and may cause unpredictable results.

System Action: Message AFF806 is issued to confirm whether the command should be executed.

AFF806A DO YOU WISH TO EXECUTE THIS COMMAND? (YES OR NO)

Explanation: This message was preceded by message AFF805, which warned of the possible consequences of executing the GO command. You must now confirm your desire to issue the command.

User Response: Reply YES or NO.

System Action: If YES is specified, the command is issued. If NO is specified, the command is not issued. Following either action, processing continues.

AFF820E OPEN ERROR ON AFFOUT FILE

Explanation: This message is issued to your SYSMSG file or spooled console when the OPEN of the AFFOUT file fails while running batch mode.

User Response: Correct the cause of the I/O error.

AFF821E ERROR PROCESSING AFFOUT FILE

Explanation: This message is issued to your SYSMSG file or spooled console when a PUT to the AFFOUT file fails while running batch mode.

User Response: Correct the cause of the I/O error.

AFF822E OPEN ERROR ON AFFIN FILE

Explanation: This message is issued when the OPEN of the AFFIN file fails while running batch mode.

User Response: Correct the cause of the I/O error.

System Action: Debugging is terminated.

AFF823E ERROR PROCESSING AFFIN FILE

Explanation: This message is issued if a GET to the AFFIN file fails while running batch mode.

User Response: Correct the cause of the I/O error.

System Action: Debugging is terminated.

AFF824E END-OF-FILE ON AFFIN FILE

Explanation: This message is issued when end-of-file is reached on the AFFIN file for batch mode.

User Response: Correct the input file and rerun the job if more commands are desired.

System Action: A QUIT command is forced.

AFF825E UNABLE TO WRITE DIAGNOSTIC MESSAGE CONCERNING AFFIN FILE

Explanation: This message is issued to your SYSMSG file or spooled console when an AFFIN diagnostic cannot be written to the AFFOUT file in batch mode.

User Response: Correct the cause of the I/O error.

AFF831E "ENTRY" AND "EXIT" ARE NOT PERMITTED IN A RANGE

Explanation: An ENTRY or EXIT keyword was issued in a statement ID range.

User Response: Use ISNs or sequence numbers for ranges. ENTRY and EXIT are usable only as individual elements.

AFF832E THE QUALIFIER ON THE RIGHT SIDE OF A RANGE MUST MATCH THE QUALIFIER ON THE LEFT SIDE

Explanation: A statement ID range was entered where a qualifier was specified for the right statement ID that does not match the one on the left side.

User Response: Use a matching qualifier or allow it to default.

AFF840E LIST RANGE IGNORED; THE SECOND VARIABLE PRECEDES THE FIRST IN STORAGE

Explanation: A LIST command has been entered with a range of variables specified. In a range, the first variable must appear in storage prior to the second variable so that the area between the two variables may be displayed.

AFF841I RANK = *number*; DUMMY ARRAY ARGUMENT OF INACTIVE SUBPROGRAM OR ALTERNATE ENTRY POINT

Explanation: This message is produced by the DESCRIBE command for a dummy array argument of an inactive subprogram or alternate entry point.

AFF850I *variable value*

Explanation: This is the output of a LIST command that did not include the DUMP keyword. The name of the requested variable is shown along with its current value.

AFF851I PROGRAM UNIT COMPILER OPT HOOKED TIMING

Explanation: This is the title line for LISTSUBS output.

AFF852I ENTRY POINT TASK TIME (MIC) PERCENT INVOCATIONS

Explanation: This is the title line for LISTTIME output.

AFF853I *timing information*

Explanation: This is the output line for timing information for program units that are being timed.

AFF854I **NO TIMING INFORMATION IS AVAILABLE**

Explanation: This message is issued if no program units are timing or have accumulated time.

AFF855I *name datatype*

Explanation: This is the DESCRIBE output for a scalar variable.

AFF856I **RANK =** *number*, **SIZE =** *number* **ELEMENTS**

Explanation: This message provides DESCRIBE information about the size of an array variable.

AFF857I **DIM** *number*, **LBOUND =** (*number*), **UBOUND =** (*number*)

Explanation: This message provides DESCRIBE information about the dimensions of an array variable.

AFF859I **DIMENSION INFORMATION NOT AVAILABLE**

Explanation: Dimension information is not available for dummy arguments of an inactive subprogram or for an alternate entry point.

AFF860E **FIXUP IGNORED; SUBSCRIPT ERROR**

Explanation: A subscript specified within a FIXUP command was out of range for the variable it was subscripting.

AFF861E **FIXUP IGNORED; NOT IN ERROR EXIT**

Explanation: You can issue a FIXUP command only if an execution error has occurred in the VS FORTRAN program.

AFF862E **FIXUP IGNORED; NO ARGUMENTS MAY BE MODIFIED**

Explanation: A FIXUP command with either ARG1 or ARG2 (or both) specified was entered for an error that has no modifiable arguments.

User Response: Enter a corrected FIXUP command or a GO command.

AFF863E **FIXUP IGNORED; ARG2 MAY NOT BE MODIFIED**

Explanation: A FIXUP command has been entered with a value specified for ARG2, but the VS FORTRAN library subroutine that detected the error only has one modifiable argument.

AFF865E "GO" WITH A STATEMENT ID IS NOT ALLOWED IN AN ERROR EXIT

Explanation: This type of GO command is not allowed within an error exit.

User Response: If you want to continue processing at some other statement, you may issue a NEXT command followed by a GO command. When execution is suspended because of the NEXT command, you may issue the GO command with a specific statement identification.

AFF866E LAST COMMAND IGNORED DUE TO AN ERROR EXIT

Explanation: The last command entered within an attention exit was not executed because an error exit occurred.

AFF867E ERROR EXIT: ERROR *number* **AT** *name.number*

Explanation: This is the notification message that occurs when an error exit is taken. It is also the response to a WHERE command in an error exit. The indicated statement and program unit name identify the last statement that was executing prior to the indicated error.

User Response: None required. Use the FIXUP or GO command to terminate the error exit processing.

AFF868E "*text***" IS INVALID I/O COMMAND SYNTAX**

Explanation: The syntax of the previous BACKSPACE, CLOSE, ENDFILE, or REWIND command is invalid. An integer variable or constant must be specified.

AFF871I WHEN: *condition* **SATISFIED**

Explanation: The indicated WHEN condition has been satisfied.

System Action: This message will be followed by message AFF872I.

AFF872I CURRENTLY AT *name.number*

Explanation: This message is issued after message AFF871I to identify the current location within the program unit.

AFF873I PROGRAM TERMINATED BY USER REQUEST

Explanation: This message is issued when you enter the "QUIT" command in an attention exit to terminate execution of a VS FORTRAN program.

System Action: Control is returned to Interactive Debug.

AFF874E ERROR ON AFFIN FILE; FILE IGNORED

Explanation: An error has occurred while attempting to open the AFFIN file. No further attempts to access this file are made.

User Response: Correct the problem that caused the message. If the file is required, correct the problem and reinvoke Interactive Debug.

AFF875E WHEN IGNORED; CONDITION "*condition*" NOT DEFINED

Explanation: A WHEN command has been detected that contains only a condition name. This is interpreted as a request to "turn the condition on." However, no condition with the specified name has been defined.

AFF876E WHEN IGNORED; SUBSCRIPT ERROR

Explanation: A subscript error was encountered while processing the WHEN command.

AFF880E OFFWN IGNORED; NO WHEN CONDITIONS ARE DEFINED

Explanation: The previous OFFWN command was ignored because there are no WHEN conditions defined that could be turned off.

AFF881E OFFWN IGNORED FOR UNDEFINED CONDITION "*condition*"

Explanation: The previous OFFWN command specified a WHEN condition that does not exist. The command is ignored.

AFF900E SYSTEM COMMAND RETURN CODE: *code*

Explanation: The indicated code was received from processing the previous system command using Interactive Debug's SYSCMD command.

AFF910I CURRENT TERMIO STATUS: *status*

Explanation: This message indicates the status of settings that are controlled by the TERMIO command.

AFF920E SET COMMAND NOT COMPLETED NORMALLY

Explanation: Because of an error described in the preceding message, the SET command was unable to be completed; not all values were assigned.

AFF925E TOO MANY VALUES; EXCESS IGNORED; SET COMPLETED

Explanation: Too many values for an array have been specified with a SET command. The extraneous values are ignored.

AFF929E MULTIPLE FORTRAN MAIN PROGRAMS HAVE BEEN FOUND; PROGRAM "program-unit-name" IS ALSO A MAIN PROGRAM

Explanation: Multiple main FORTRAN program units were found in the program being debugged. Only programs with a single main program unit can be debugged.

System Action: Processing continues. Interactive Debug assumes that the last main program unit in the load module is the main program that was invoked. If this is not the case, message AFF937E will be issued.

User Response: Relink-edit the program to contain only one main program unit.

AFF930E NAME "name" GREATER THAN 31 CHARACTERS; TRUNCATED

Explanation: A program unit name in the AFFON file is greater than 31 characters. The name is truncated to 31 characters.

User Response: If the name is misspelled, correct the spelling in the AFFON file and reinvoke the debugger.

AFF931E THE AFFON SOURCE LISTING DATA SET NAME FOR "name" CONTAINS INVALID SYNTAX AND WILL BE IGNORED.

Explanation: The syntax of the data set name specified in the AFFON file for the named program unit is invalid.

System Action: The listing data set name specification and the ISN restriction list (if any) are ignored.

User Response: Correct the file name. Especially check for a missing closing quote.

AFF932E DUPLICATE NAME "*name*"

Explanation: The indicated program unit name has been found more than once in the AFFON file.

System Action: The duplicate entry is ignored, and processing continues.

AFF933I AFFON FILE PROCESSED

Explanation: This is an informational message informing you that the AFFON file was processed.

AFF934E PROGRAM NAME "*program*" NOT FOUND

Explanation: The indicated program unit name was specified in the AFFON file, but could not be found in the program to be debugged.

System Action: The program unit name is ignored, and processing continues.

AFF935E INTERNAL ERROR IN GENERATED CODE FOR "*program*"

Explanation: This is an internal error and should never occur. It indicates that, within the first four bytes generated by the VS FORTRAN Version 1 or VS FORTRAN Version 2 compiler for a VS FORTRAN statement, an instruction of the form "BALR x,y" was detected. Such an instruction is only valid if y=0.

User Response: Contact your IBM representative.

System Action: Debugging is discontinued.

AFF936E CALL STACK OVERFLOW; TOO MANY LEVELS OF CALL IN "*program*"

Explanation: Interactive Debug maintains information about each debuggable program unit involved in execution. It has the ability to keep information for 512 different program units. This message indicates that more than 512 debuggable program units are currently active.

User Response: Use the AFFON file to select fewer program units for debugging so no more than 512 units will be active at one time.

System Action: An abend is forced.

AFF937E HOOK FOUND AT UNKNOWN LOCATION

Explanation: This is probably an internal error. It indicates that what appears to be an Interactive Debug hook was found at a location that Interactive Debug could not recognize.

User Response: Contact your IBM representative.

System Action: Interactive Debug returns control to the VS FORTRAN Version 2 library for its normal program check handling.

AFF938E ADDRESSING MODE CHANGED IN *program unit name*

Explanation: An external routine called from a VS FORTRAN program unit has changed the MVS/XA addressing mode and not restored it. Unless this is an unusual or intentional situation, an abend will probably occur. Be aware that there are some situations in which an abend will occur before Interactive Debug can get control and issue this message. For example, a program running above the 16-megabyte line that switches to 24-bit addressing mode will abend immediately.

There are situations, however, where this message will be issued as informational only. For example, the Release 4 VS FORTRAN Version 1 library always enters user error exits in 31-bit addressing mode. If an MVS/XA program is linked to run 24/24, this message will be issued the first time such a routine is entered. In this case, it should be treated as informational only, and not regarded as an error condition.

User Response: Unless the change of addressing mode was intentional, or unless a user exit was taken by a 24/24 program, you should check and correct the addressing mode logic in any external routines called by the specified program unit.

AFF939I RELEASE 3.0 COMPILED PROGRAM UNIT *name* CANNOT BE DEBUGGED ON MVS/XA

Explanation: A program unit compiled with the Release 3 VS FORTRAN Version 1 compiler has been included in the AFFON file. Such program units are not debuggable in MVS/XA, unless they were compiled with the TEST option, or are recompiled with Release 3.1 or later.

System Action: The program unit name is ignored, and processing continues.

AFF940I ZERO-FREQUENCY STATEMENTS

Explanation: This is the heading of a listing of statements within the indicated program unit that have not been executed, in response to LISTFREQ.

System Action: The ISNs or sequence numbers are listed.

AFF942I *name.statement*

Explanation: This is the output of the LISTFREQ command when the ZEROFREQ keyword has been specified.

AFF943I NONE

Explanation: One of the following is true:

- A LISTFREQ command with the ZEROFREQ keyword was entered, and all statements in the currently qualified program unit have been executed at least once.
- A LISTBRKS command was entered, and no breakpoints have been established using the AT command.
- A LISTBRKS command was entered, and no WHEN conditions have been established.

AFF945I STATEMENT FREQUENCY

Explanation: This is the heading for a listing of execution counts, in response to LISTFREQ.

System Action: The ISNs or sequence numbers are listed.

AFF947I *name.number frequency*

Explanation: This is the output of the LISTFREQ command.

AFF950I CURRENT BREAKPOINTS:

Explanation: This is the heading for the output of the LISTBRKS command for breakpoints set by the AT command.

System Action: All active breakpoints are listed.

AFF952I *name.statement*

Explanation: This is the output from the LISTBRKS command for AT commands with a COUNT of 1.

AFF953I *name.statement* **COUNT(***count***)**

Explanation: This is the output from the LISTBRKS command for AT commands with a COUNT greater than 1.

AFF955I CURRENT WHEN CONDITIONS:

Explanation: This is the heading for the output of the LISTBRKS command for WHEN conditions.

AFF957I *condition name setting condition*

Explanation: This is the output from the LISTBRKS command for WHEN conditions. Each defined condition is shown, along with an indication of whether the condition is currently being monitored (ON) or not (OFF).

AFF960W COMMAND WILL BE ATTEMPTED, BUT PROGRAM HAS FINISHED EXECUTION

Explanation: The previous command has been accepted and will be executed normally, but execution of the program has completed and may not be restarted.

AFF970W *program* **IS OPTIMIZED; SETS MAY BE INEFFECTIVE AND REFERENCES MAY GET INVALID VALUES**

Explanation: The indicated program unit has been compiled with the OPT(n) or VECTOR(n) compiler option with n>0. Because the program unit is optimized, Interactive Debug commands that reference storage locations may produce unexpected results. For example, LIST A will report the value in the storage location the compiler established for variable A, but the compiled code may not be using the storage location to keep the value of the variable; it may be kept in a register instead. Interactive Debug cannot detect this situation, but it is likely to occur in optimized program units.

User Response: Use affected commands with care. See "Debugging Optimized and Vectorized Code" on page 98, which discusses the effect of optimized code on VS FORTRAN Version 2 Interactive Debug.

System Action: Processing continues. This message should only appear the first time you issue one of the affected commands in a program unit.

AFF971E STATEMENT *number* IS COLLAPSED AND MAY NOT BE USED FOR DEBUGGING

Explanation: The indicated statement cannot have a breakpoint associated with it because it occupies no storage (because of optimization or vectorization).

User Response: If necessary, set a breakpoint at a statement immediately before or after the collapsed statement. You can use LISTFREQ to show which statements are collapsed.

AFF972I STATEMENT *number* DOES NOT HAVE AN ESTABLISHED BREAKPOINT

Explanation: An OFF command specifies that a breakpoint at the indicated statement is to be removed. There is no breakpoint at this statement.

AFF973I "OFF" IGNORED; NO BREAKPOINTS ARE DEFINED

Explanation: An OFF command has been issued, but there are no breakpoints defined in the currently qualified program unit.

AFF974E NO FORTRAN MODULE WAS FOUND IN STORAGE

Explanation: Interactive Debug could not determine the storage limits for searching for debuggable programs. This happens if the application program was invoked using a load (for example, using the TSO LOADGO command).

User Response: Be sure your application program is invoked using an **attach** or **link** command.

System Action: Debugging is terminated with ABEND 111.

AFF975E *command* COMMAND IGNORED; NO CURRENT QUALIFICATION

Explanation: There is no program unit serving as the current qualification. Perhaps execution was suspended before any debuggable program was executed.

User Response: Set a qualification using the QUALIFY command, and reenter the command causing the message.

AFF976E SYNTAX ERROR IN *command* COMMAND

Explanation: A syntax error was detected in scanning the indicated command.

User Response: Determine the cause of the error, and reenter the command with the correct syntax.

AFF977E I/O UNIT NUMBER IN "*command*" MUST BE AN INTEGER

Explanation: While scanning the previous command, where an integer value or integer variable was expected, a duplication factor or some other type of data was found.

User Response: Reenter the command with an integer value or the name of an INTEGER variable.

AFF978E COMMAND IGNORED; "ENDDEBUG" HAS BEEN ISSUED

Explanation: The last command entered is being ignored because of a previously issued ENDDEBUG command. After ENDDEBUG is issued, you no longer have the ability to debug.

User Response: You may issue QUIT if you no longer want to execute the program, or may enter a null line to continue execution.

AFF979E "*command*" IGNORED; TERMINAL INPUT PENDING

Explanation: A GO, ENDDEBUG, or STEP command was entered, but is not permitted when a program is waiting for input.

User Response: If you want to enter a GO, ENDDEBUG, or STEP command, enter a NEXT command now, and, when execution stops for the NEXT, enter the GO or ENDDEBUG command.

AFF980E THE "STEP" OPERAND MUST BE AN INTEGER GREATER THAN ZERO

Explanation: STEP was entered with an invalid operand.

AFF981I ALL BREAKPOINTS IGNORED FOR PROGRAM UNIT "*name*"

Explanation: A dynamically loaded program unit with pending breakpoints was found to be loaded in read-only storage.

AFF982I NO BREAKPOINTS ARE DEFINED WITHIN THE SPECIFIED RANGE

Explanation: The OFF command specified a range where no breakpoints were defined.

AFF990I START OF RUN OR NO DEBUGGABLE PROGRAM EXECUTING

Explanation: A WHERE command has been detected, but either only the first prompt has been received and execution has not really started, or execution was interrupted while a nondebuggable program was executing.

AFF991I *program* **CALLED AT** *name.number*

Explanation: This message is the output of the WHERE command with the TRBACK keyword. The message indicates the logic flow through the program units.

AFF992I NO SUBROUTINES CALLED

Explanation: This message is issued following a WHERE TRBACK command when no subroutines have been called. This implies that execution is currently within the topmost debuggable program unit (usually the "main" program).

AFF995I WHERE: *name.number*

Explanation: This is the output of the WHERE command. It shows that execution is currently at the indicated statement in the indicated program unit. If execution is currently within a program unit that is not debuggable (and execution was suspended because of an attention interrupt), then the specified statement will be the last statement executed in a debuggable program unit.

AFF996I TO: *name.number* **FROM:** *name.number*

Explanation: This message is the output of the WHERE command with the FLOW keyword. The message indicates the logic flow through the program units.

AFF998E INTERNAL ERROR *number* **AT LOCATION** *number*; **IAD STORAGE MAY HAVE BEEN OVERLAID BY PROGRAM**

Explanation: This message is issued when an internal Interactive Debug abend occurs.

User Response: Contact your IBM representative.

System Action: Abend 101 is issued.

Glossary

This glossary includes definitions developed by the American National Standards Institute (ANSI), and the International Organization for Standardization (ISO).

This material is reproduced from the American National Dictionary for Information Processing, copyright 1977 by the Computer and Business Equipment Manufacturers Association, copies of which may be purchased from the American National Standards Institute, 1430 Broadway, New York, New York 10018.

An asterisk (*) to the right of an item number indicates an ANSI definition in an entry that also includes other definitions.

The symbol "(ISO)" at the beginning of a definition indicates that it has been discussed and agreed upon at meetings of the International Organization for Standardization Technical Committee 97/Subcommittee 1 (Data Processing Vocabulary), and has also been approved by ANSI and included in the *American National Dictionary for Information Processing.*

addressing mode. The length of an address, either 24 bits or 31 bits, used by the processor. Indicated by the high-order bit of the PSW in an MVS/XA environment.

alphabetic character. A character of the set A, B, C, ... Z. See also "letter." In VS FORTRAN Version 1 and VS FORTRAN Version 2, the currency symbol ($) is considered an alphabetic character. In VS FORTRAN Version 2, lowercase letters (a through z) are also valid.

alphameric. Pertaining to a character set that contains letters (A through Z) and digits (0 through 9) only. In VS FORTRAN Version 2, the character set may also contain lowercase letters (a through z).

alphameric character set. A character set that contains both letters and digits.

alternate entry. (1) In VS FORTRAN, an entry provided by means of the ENTRY statement. (2) As used by Interactive Debug, an entry other than the one by which the subprogram was actually entered.

animation. Under ISPF with Interactive Debug, the ability to highlight the command currently executing and control the pace of execution when using the STEP command. This creates an "animated" picture of your program's execution.

argument. A parameter passed between a calling program and a SUBROUTINE subprogram, a FUNCTION subprogram, or a statement function.

arithmetic constant. A constant of type integer, real, double-precision, or complex.

arithmetic expression. One or more arithmetic operators and/or arithmetic primaries, the evaluation of which produces a numeric value. An arithmetic expression can be an unsigned arithmetic constant, the name of an arithmetic constant, or a reference to an arithmetic variable, array element, or function reference, or a combination of such primaries formed by using arithmetic operators and parentheses.

arithmetic operator. A symbol that directs VS FORTRAN to perform an arithmetic operation. The arithmetic operators are:

+	addition
-	subtraction
*	multiplication
/	division
**	exponentiation

array. An ordered set of data items identified by a single name.

array declarator. The part of a statement that describes an array used in a program unit. It indicates the name of the array, the number of dimensions it contains, and the size of each dimension. An array declarator may appear in a DIMENSION, COMMON, or explicit type statement.

array element. A data item in an array, identified by the array name followed by a subscript indicating its position in the array.

array name. The name of an ordered set of data items that make up an array.

assignment statement. A statement that assigns a value to a variable or array element. It is made up of a variable or array element, followed by an equal sign (=), followed by an expression. The variable, array element, or expression can be of type character, logical, or arithmetic. When the assignment statement is executed, the expression to the right of the equal sign replaces the value of the variable or array element to the left.

basic real constant. A string of decimal digits containing a decimal point, and expressing a real value.

blank common. An unnamed common block.

breakpoint. (1) (ISO) A place in a computer program, usually specified by an instruction, where its execution may be interrupted by external intervention or by a monitor program. (2) As used by IAD, a VS FORTRAN statement where the user has specified that execution is to be suspended, or that some action is to be taken.

character constant. A string of one or more characters enclosed in apostrophes. The delimiting apostrophes are not part of the constant.

character expression. An expression in the form of a single character constant, variable, array element, substring, function reference, or another expression enclosed in parentheses. A character expression is always of type character.

character type. A data type that can consist of any characters; in storage, one byte is used for each character.

collapsed statement. A statement for which no machine code has been generated because of the nature of the statement, or as a result of optimization or vectorization.

common block. A storage area that may be referred to by a calling program and one or more subprograms.

complex constant. An ordered pair of real or integer constants separated by a comma and enclosed in parentheses. The first real constant of the pair is the real part of the complex number; the second is the imaginary part.

complex type. An approximation of the value of a complex number, consisting of an ordered pair of real data items separated by a comma and enclosed in parentheses. The first item represents the real part of the complex number; the second represents the imaginary part.

connected file. A file that has been connected to a unit and defined by a FILEDEF command or by job control statements.

constant. An unvarying quantity. The four classes of constants specify numbers (arithmetic), truth values (logical), character data (character), and hexadecimal data.

control statement. Any of the statements used to alter the normal sequential execution of FORTRAN statements, or to terminate the execution of a VS FORTRAN program. FORTRAN control statements are any of the forms of the GO TO, IF, and DO statements, or the PAUSE, CONTINUE, and STOP statements.

data. (1) * (ISO) A representation of facts or instructions in a form suitable for communication, interpretation, or processing by human or automatic means. (2) In FORTRAN, data includes constants, variables, arrays, and character substrings.

data item. A constant, variable, array element, or character substring.

data set. The major unit of data storage and retrieval consisting of data collected in one of several prescribed arrangements and described by control information to which the system has access.

data set reference number. A constant or variable in an input or output statement that identifies a data set to be processed.

data type. The properties and internal representation that characterize data and functions. The basic types are integer, real, complex, logical, double precision, and character.

debugging hook. See "hook."

*** digit.** (ISO) A graphic character that represents an integer. For example, one of the characters 0 through 9.

DO-loop. A range of statements executed repetitively by a DO statement.

double precision. The standard name for real data of storage length 8.

DO variable. A variable, specified in a DO statement, that is initialized or incremented prior to each execution of the statement or statements within a DO range. It is used to control the number of times the statements within the range are executed.

dummy argument. A variable within a subprogram or statement function definition with which actual arguments from the calling program or function reference are positionally associated. Dummy arguments are defined in a SUBROUTINE or FUNCTION statement, or in a statement function definition.

dynamic common. A VS FORTRAN named common which the DC compiler option specifies should be allocated at execution time.

executable program. A program that can be executed as a self-contained procedure. It consists of a main program and, optionally, one or more subprograms or non-FORTRAN-defined external procedures, or both.

executable statement. A statement that causes an action to be taken by the program; for example, to calculate, to test conditions, or is a control statement.

existing file. A file that has been defined by a FILEDEF command or by job control statements. A valid unit number in FORTRAN's internal unit assignment table, as specified at installation time.

The INQUIRE statement considers a file to exist on the basis of FORTRAN I/O statements that have been processed.

existing unit. A valid unit number in FORTRAN's internal unit assignment table, as specified at installation.

expression. A notation that represents a value: a constant or a reference appearing alone, or combinations of constants and/or references with operators. An expression can be arithmetic, character, logical, or relational.

external file. A set of related external records treated as a unit; for example, in stock control, an external file would consist of a set of invoices.

external function. A function defined outside the program unit that refers to it.

external procedure. A SUBROUTINE OR FUNCTION subprogram written in FORTRAN.

file. A set of records. If the file is located in internal storage, it is an internal file; if it is on an input/output device, it is an external file.

file definition statement. A statement that describes the characteristics of a file to a user program. For example, the OS/VS DD statement or the FILEDEF command for CMS processing.

file reference. A reference within a program to a file. It is specified by a unit identifier.

formatted record. (1) A record, described in a FORMAT statement, that is transmitted, when necessary with data conversion, between internal storage and internal storage or to an external record. (2) A record transmitted with list-directed READ or WRITE statements and an EXTERNAL statement.

FORTRAN-supplied procedure. See "intrinsic function."

function reference. A source program reference to an intrinsic function, to an external function, or to a statement function.

function subprogram. A subprogram invoked through a function reference, and headed by a FUNCTION statement. It returns a value to the calling program unit at the point of reference.

hexadecimal constant. A constant that is made up of the character Z followed by two or more hexadecimal digits.

hierarchy of operations. The relative priority order used to evaluate expressions containing arithmetic, logical, or character operations.

hook. Also, debugging hook. An internal segment of code that is able to give temporary control to a debugging program such as Interactive Debug. The code is independent of the algorithm implemented in the application program containing the hook. In VS FORTRAN Version 2 Interactive Debug, breakpoints can only be set at statements that have hooks.

IAD. An abbreviation for Interactive Debug, used with VS FORTRAN Version 1 and VS FORTRAN Version 2.

implied DO. An indexing specification, similar to a DO statement, causing repetition over a range of data elements. (The word DO is omitted, hence the term "implied.")

integer constant. A string of decimal digits containing no decimal point and expressing a whole number.

integer expression. An arithmetic expression whose values are of integer type.

integer type. An arithmetic data type, capable of expressing the value of an integer. It can have a positive, negative, or zero value; it must not include a decimal point.

internal file. A set of related internal records treated as a unit.

internal statement number (ISN). A number assigned to each statement in a VS FORTRAN program by the VS FORTRAN compiler. ISNs are assigned sequentially beginning with 1, and are visible on the listing produced by the compiler.

ISN is sometimes referred to as "internal sequence number." See also "statement identifier" and "sequence number."

interruption localizing. A function that occurs at optimization level 2. It restricts certain optimizations so that no code is moved out of a loop if it would cause an interruption that would not occur without optimization.

intrinsic function. A function, supplied by VS FORTRAN, that performs mathematical or character operations.

*** I/O.** Pertaining to either input or output, or both.

I/O list. A list of variables in an input or output statement specifying which data is to be read or which data is to be written. An output list may also contain a constant, an expression involving operators or function references, or an expression enclosed in parentheses.

labeled common. See "named common."

length specification. A source language specification of the number of bytes to be occupied by a variable or an array element.

letter. A symbol representing a unit of the English alphabet.

list-directed. An input/output specification that uses a data list instead of a FORMAT specification.

logical constant. A constant that can have one of two values: "true" or "false."

logical expression. A combination of logical primaries and logical operators. A logical operator can have one of two values: true or false.

logical operator. Any of the set of operators .NOT. (negation), .AND. (connection: both), or .OR. (inclusion: either or both), .EQV. (equal), .NEQV.(not equal).

logical primary. A primary that can have the value "true" or "false." See also "primary."

logical type. A data type that can have the value "true" or "false" for VS FORTRAN Version 1 or VS FORTRAN Version 2. See also "data type."

looping. Repetitive execution of the same statement or statements. Usually controlled by a DO statement.

main program. A program unit, required for execution, that can call other program units but cannot be called by them.

name. A string of from one through thirty-one alphameric characters, the first of which must be alphabetic. The underscore (_) is a valid character. Used to identify a constant, a variable, an array, a function, a subroutine, or a common block.

named common. A separate common block consisting of variables, arrays, and array declarators, and given a name.

nested DO. A DO statement whose range of statements is entirely contained within the range of another DO statement.

nonexecutable statement. A statement that describes the characteristics of the program unit, of data, of editing information, or of statement functions, but does not cause an action to be taken by the program.

nonexisting file. A file that has not been defined by a FILEDEF command or by job control statements.

*** numeric character.** (ISO) Synonym for digit.

numeric constant. A constant that expresses an integer, real, or complex number.

preconnected file. A unit that was defined at installation time. However, a preconnected unit does not exist for a program if the unit is not defined by a FILEDEF command or by job control statements.

predefined specification. The implied type and length specification of a data item, based on the initial character of its name in the absence of any specification to the contrary. The initial characters I-N type data items as integer; the initial characters A-H, O-Z, and $ type data items as real. No other data types are predefined. For VS FORTRAN Version 1 and VS FORTRAN Version 2, the length of both types is 4 bytes.

primary. An irreducible unit of data; a single constant, variable, array element, function reference, or expression enclosed in parentheses.

procedure. A sequenced set of statements that may be used at one or more points in one or more computer programs, and that usually is given one or more input parameters and returns one or more output parameters. A procedure consists of subroutines, function subprograms, and intrinsic functions.

procedure subprogram. A function or subroutine subprogram.

program return code. When a program is terminated with a nonzero return code, the code is available for interrogation by means of job control language for the appropriate operating system.

program unit. A sequence of statements constituting a main program or subprogram.

real constant. A string of decimal digits that expresses a real number. A real constant must contain either a decimal point or a decimal exponent and may contain both.

real type. An arithmetic data type, capable of approximating the value of a real number. It can have a positive, negative, or zero value.

record. A collection of related items of data treated as a unit.

relational expression. An expression that consists of an arithmetic expression followed by a relational operator, followed by another arithmetic expression or a character expression followed by a relational operator, followed by another character expression. The result is a value that is true or false. In Interactive Debug the only arithmetic expression permitted in a relational expression is a variable, an array element, or a constant.

relational operator. Any of the set of operators that can express a comparison between arithmetic expressions, and that can be either true or false:

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

residence mode. Where a program resides in virtual storage in an MVS/XA environment: above or below 16 megabytes.

scale factor. A specification in a FORMAT statement that changes the location of the decimal point in a real number (and, if there is no exponent, the magnitude of the number).

sequence number. A number found in positions 73 through 80 of records containing source statements for the VS FORTRAN compiler. Sequence numbers are not necessarily unique, in sequence, or present in every record. See "internal statement number" and "statement identifier."

specification statement. One of the set of statements that provides the compiler with information about the data used in the source program. In addition, the statement supplies the information required to allocate data storage.

specification subprogram. A subprogram headed by a BLOCK DATA statement and used to initialize variables in named common blocks.

statement. The basic unit of a VS FORTRAN program, that specifies an action to be performed, or the nature and characteristics of the data to be processed, or information about the program itself. Statements fall into two broad classes: executable and nonexecutable.

statement function. A name, followed by a list of dummy arguments, that is equated to an arithmetic, logical, or character expression. In the remainder of the program the name can be used as a substitute for the expression.

statement function definition. A statement that defines a statement function. Its form is a name, followed by a list of dummy arguments, followed by an equal sign (=), followed by an arithmetic, logical, or character expression.

statement function reference. A reference in an arithmetic, logical, or character expression to the name of a previously defined statement function.

statement identifier. The statement label, internal statement number (ISN), or sequence number used by VS FORTRAN Version 2 Interactive Debug to identify a statement in a VS FORTRAN program. The options that existed when the program was compiled determine whether the ISN or the sequence number is valid. In many cases, you can also use the statement label, preceded by a slash, as a valid identifier for a statement.

See also "internal statement number" and "sequence number."

statement label. A number of from one through five decimal digits that is used to identify a statement. Statement labels can be used to transfer control, to define the range of a DO statement, or to refer to a FORMAT statement.

subprogram. A program unit that is invoked by another program unit in the same program. In FORTRAN, a subprogram has a FUNCTION, SUBROUTINE, or BLOCK DATA statement as its first statement.

subroutine subprogram. A subprogram whose first statement is a SUBROUTINE statement. It optionally returns one or more parameters to the calling program unit.

*** subscript.** A subscript quantity or set of subscript quantities, enclosed in parentheses and used with an array name to identify a particular array element.

subscript quantity. A component of a subscript: an integer constant, an integer variable, or an expression evaluated as an integer constant.

type specification. The explicit specification of the type of a constant, variable, array, or function by use of an explicit type specification statement.

unformatted record. A record that is transmitted unchanged between internal storage and an external record.

unit. A means of referring to a file in order to use input/output statements. A unit can be connected or not connected to a file. If connected, it refers to a file. The connection is symmetric: that is, if a unit is connected to a file, the file is connected to the unit.

unit identifier. The number that specifies an external unit.

1. An integer expression whose value must be zero or positive. For VS FORTRAN Version 1 and VS FORTRAN Version 2, this integer value of length 4 must correspond to a DD name, a FILEDEF name, or an ASSGN name.

2. An asterisk (*) that corresponds on input to FT05F001 and on output to FT06F001.

3. The name of a character array, character array element, or character substring for an internal file.

variable. A data item, identified by a name, that is not a named constant, array, or array element, and that can assume different values at different times during program execution.

vectorization. The process of creating machine instructions that will execute on the special vector processing facility of the IBM 3090 Vector Facility.

vectorize. To compile a source program so its eligible DO loop statements are transformed into vector object code.

Index

Special Characters

A

B

C

D

E

F

K

L

M

N

O

P

W

Z

Numerics

Reader's Comment Form

VS FORTRAN Version 2
Interactive Debug
Guide and Reference
SC26-4223-1

This manual is part of a library that serves as a reference source for system analysts, programmers, and operators of IBM systems. You may use this form to communicate your comments about this publication, its organization, or subject matter, with the understanding that IBM may use or distribute whatever information you supply in any way it believes appropriate without incurring any obligation to you.

Your comments will be sent to the author's department for whatever review and action, if any, are deemed appropriate.

Note: Do not use this form to request IBM publications. If you do, your order will be delayed because publications are not stocked at the address printed on the reverse side. Instead, you should direct any requests for copies of publications, or for assistance in using your IBM system, to your IBM representative or to the IBM branch office serving your locality.

If you have applied any technical newsletters (TNLs) to this book, please list them here: _____

Chapter/Section _____

_____ Page No. _____

Comments:

Note: Staples can cause problems with automated mail sorting equipment. Please use pressure sensitive or other gummed tape to seal this form.

If you want a reply, please complete the following information.

Name _____ Phone No. (____)_____

Company _____

Address _____

_____

Thank you for your cooperation. No postage stamp necessary if mailed in the U.S.A. (Elsewhere, an IBM office or representative will be happy to forward your comments or you may mail directly to the address in the Edition Notice on the back of the title page.)

SC26-4223-1

Reader's Comment Form

Fold and tape | Please do not staple | Fold and tape

NO POSTAGE
NECESSARY
IF MAILED
IN THE
UNITED STATES

BUSINESS REPLY MAIL
FIRST CLASS PERMIT NO. 40 ARMONK, N.Y.

POSTAGE WILL BE PAID BY ADDRESSEE

IBM Corporation
P.O. Box 50020
Programming Publishing
San Jose, California 95150

Fold and tape | Please do not staple | Fold and tape

IBM

VS FORTRAN Version 2: Interactive Debug Guide and Reference (File No. S370-40) Printed in U.S.A. SC26-4223-1

VS FORTRAN Version 2
Interactive Debug
Guide and Reference
SC26-4223-1

Reader's Comment Form

This manual is part of a library that serves as a reference source for system analysts, programmers, and operators of IBM systems. You may use this form to communicate your comments about this publication, its organization, or subject matter, with the understanding that IBM may use or distribute whatever information you supply in any way it believes appropriate without incurring any obligation to you.

Your comments will be sent to the author's department for whatever review and action, if any, are deemed appropriate.

Note: Do not use this form to request IBM publications. If you do, your order will be delayed because publications are not stocked at the address printed on the reverse side. Instead, you should direct any requests for copies of publications, or for assistance in using your IBM system, to your IBM representative or to the IBM branch office serving your locality.

If you have applied any technical newsletters (TNLs) to this book, please list them here: _____

Chapter/Section _____

_____ Page No. _____

Comments:

Note: Staples can cause problems with automated mail sorting equipment. Please use pressure sensitive or other gummed tape to seal this form.

If you want a reply, please complete the following information.

Name _____ Phone No. (____)_____

Company _____

Address _____

_____

Thank you for your cooperation. No postage stamp necessary if mailed in the U.S.A. (Elsewhere, an IBM office or representative will be happy to forward your comments or you may mail directly to the address in the Edition Notice on the back of the title page.)

SC26-4223-1

Reader's Comment Form

Fold and tape | Please do not staple | Fold and tape

NO POSTAGE
NECESSARY
IF MAILED
IN THE
UNITED STATES

BUSINESS REPLY MAIL
FIRST CLASS PERMIT NO. 40 ARMONK, N.Y.

POSTAGE WILL BE PAID BY ADDRESSEE

IBM Corporation
P.O. Box 50020
Programming Publishing
San Jose, California 95150

Fold and tape | Please do not staple | Fold and tape

IBM®

VS FORTRAN Version 2: Interactive Debug Guide and Reference (File No. S370-40) Printed in U.S.A. SC26-4223-1

Reader's Comment Form

VS FORTRAN Version 2
Interactive Debug
Guide and Reference
SC26-4223-1

This manual is part of a library that serves as a reference source for system analysts, programmers, and operators of IBM systems. You may use this form to communicate your comments about this publication, its organization, or subject matter, with the understanding that IBM may use or distribute whatever information you supply in any way it believes appropriate without incurring any obligation to you.

Your comments will be sent to the author's department for whatever review and action, if any, are deemed appropriate.

Note: Do not use this form to request IBM publications. If you do, your order will be delayed because publications are not stocked at the address printed on the reverse side. Instead, you should direct any requests for copies of publications, or for assistance in using your IBM system, to your IBM representative or to the IBM branch office serving your locality.

If you have applied any technical newsletters (TNLs) to this book, please list them here: _____

Chapter/Section _____

_____ Page No. _____

Comments:

Note: Staples can cause problems with automated mail sorting equipment. Please use pressure sensitive or other gummed tape to seal this form.

If you want a reply, please complete the following information.

Name _____ Phone No. (____)_____

Company _____

Address _____

_____

Thank you for your cooperation. No postage stamp necessary if mailed in the U.S.A. (Elsewhere, an IBM office or representative will be happy to forward your comments or you may mail directly to the address in the Edition Notice on the back of the title page.)

SC26-4223-1

Reader's Comment Form

Fold and tape | Please do not staple | Fold and tape

NO POSTAGE
NECESSARY
IF MAILED
IN THE
UNITED STATES

BUSINESS REPLY MAIL
FIRST CLASS PERMIT NO. 40 ARMONK, N.Y.

POSTAGE WILL BE PAID BY ADDRESSEE

IBM Corporation
P.O. Box 50020
Programming Publishing
San Jose, California 95150

Fold and tape | Please do not staple | Fold and tape

IBM

VS FORTRAN Version 2: Interactive Debug Guide and Reference (File No. S370-40) Printed in U.S.A. SC26-4223-1

VS FORTRAN Version 2
Interactive Debug
Guide and Reference
SC26-4223-1

Reader's Comment Form

This manual is part of a library that serves as a reference source for system analysts, programmers, and operators of IBM systems. You may use this form to communicate your comments about this publication, its organization, or subject matter, with the understanding that IBM may use or distribute whatever information you supply in any way it believes appropriate without incurring any obligation to you.

Your comments will be sent to the author's department for whatever review and action, if any, are deemed appropriate.

Note: Do not use this form to request IBM publications. If you do, your order will be delayed because publications are not stocked at the address printed on the reverse side. Instead, you should direct any requests for copies of publications, or for assistance in using your IBM system, to your IBM representative or to the IBM branch office serving your locality.

If you have applied any technical newsletters (TNLs) to this book, please list them here: _____

Chapter/Section _____

_____ Page No. _____

Comments:

Note: Staples can cause problems with automated mail sorting equipment. Please use pressure sensitive or other gummed tape to seal this form.

If you want a reply, please complete the following information.

Name _____ Phone No. (____)_____

Company _____

Address _____

_____

Thank you for your cooperation. No postage stamp necessary if mailed in the U.S.A. (Elsewhere, an IBM office or representative will be happy to forward your comments or you may mail directly to the address in the Edition Notice on the back of the title page.)

SC26-4223-1

Reader's Comment Form

Fold and tape | Please do not staple | Fold and tape

NO POSTAGE
NECESSARY
IF MAILED
IN THE
UNITED STATES

BUSINESS REPLY MAIL
FIRST CLASS PERMIT NO. 40 ARMONK, N.Y.

POSTAGE WILL BE PAID BY ADDRESSEE

IBM Corporation
P.O. Box 50020
Programming Publishing
San Jose, California 95150

Fold and tape | Please do not staple | Fold and tape

IBM

VS FORTRAN Version 2: Interactive Debug Guide and Reference (File No. S370-40) Printed in U.S.A. SC26-4223-1

VS FORTRAN Version 2
Interactive Debug
Guide and Reference

File Number S370-40

The VS FORTRAN Version 2 Library

LY27-9516	Diagnosis Guide
GC26-4219	General Information
SC26-4340	Installation and Customization for MVS
SC26-4339	Installation and Customization for VM
SC26-4223	Interactive Debug Guide and Reference
SC26-4221	Language and Library Reference
GC26-4225	Licensed Program Specifications
SC26-4222	Programming Guide
SX26-3751	Reference Summary

SC26-4223-01

Printed in U.S.A.