IBM

LY27-9525-1

VS Pascal

Diagnosis Guide and Reference

Release 2

IBM

LY27-9525-1

VS Pascal

Diagnosis Guide and Reference

Release 2

Second Edition (December 1988)

This edition replaces and makes obsolete the previous edition, LY27-9525-0.

This edition applies to Release 2 of VS Pascal, Program Number 5668-767 (Compiler and Library) and 5668-717 (Library only) 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" in Appendix A. Because the technical changes in this edition are extensive and difficult to localize, they are not indicated by vertical bars in the left margin.

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, 4300, and 9370 Processors Bibliography*, GC20-0001, for the editions that are applicable and current.

Preface

This book is provided to help persons diagnose problems when using VS Pascal and presents a systematic way of selecting keywords to describe a suspected failure in the VS Pascal compiler or library.

This book contains no definition of the VS Pascal programming language and its syntax. For information about this subject matter, see *VS Pascal Language Reference.*

It assumes that you have:

1. Used all the debugging tools and options available to you, such as static debug statements or the VS Pascal Interactive Debugging Tool.
2. Examined all VS Pascal messages and error conditions produced by the program, and corrected them when possible.
3. Ensured that the error is occurring in the VS Pascal compiler or library and not in the application program.
4. Noted the specific sequence of events preceding the error condition.

If you have not completed all these steps, see *VS Pascal Application Programming Guide* for more information about how to do so.

How This Manual Is Organized

Part 1, Determining the Cause of the Problem, helps you locate and identify the source of the problem.

Chapter 1, "Isolating the Problem" on page 3, suggests ways in which you can determine whether you are experiencing a user error or a problem in the VS Pascal compiler or library.

Chapter 2, "Creating a Test Case" on page 11, explains how to create a test case that you can use to isolate the error.

Chapter 3, "Diagnosis Aids" on page 13, summarizes VS Pascal features that will help you identify the error.

Part 2, Identifying the Problem with Keywords, helps you locate any existing solutions to the problem once you have determined that VS Pascal is the cause.

Chapter 4, "Building a Keyword String" on page 21, explains how to build a keyword string with which you can search a data base for possible solutions to the problem.

Chapter 5, "Keywords for Options" on page 35, lists the compile-time, link-time, and run-time options you need to include in the keyword string.

Chapter 6, "Searching for a Solution with Your Keyword String" on page 39, describes how to search the data base.

Part 3, Reporting the Problem to IBM, explains what information you must submit to the IBM Support Center if the problem cannot be resolved by any other means.

Chapter 7, "Preparing an APAR" on page 43, explains how to submit an authorized program analysis report (APAR) if an APAR describing the problem does not already exist.

Part 4, Reference, provides additional information on VS Pascal that may help you isolate the source of the problem.

Chapter 8, "Further VS Pascal Diagnostic Information" on page 49, describes the major functions and routines of the VS Pascal compiler and library.

Appendix A, "Summary of Changes" on page 61, details additions and enhancements related to diagnosing problems that VS Pascal Release 2 makes to VS Pascal Release 1.

The **"Bibliography" on page 63,** lists VS Pascal publications and other publications that might help you to diagnose problems.

Contents

Figures

Part 1. Determining the Cause of the Problem

Chapter 1. Isolating the Problem

In order to correct a problem in a VS Pascal program, you must isolate and accurately define the problem. The diagnosis hints in this chapter, and in those parts of the *VS Pascal Application Programming Guide* that discuss debugging VS Pascal programs, can help you in this process.

General Questions to Consider

The following questions and suggested courses of action might help you determine whether you are dealing with an error in your application program or in VS Pascal.

As you work, keep a record of messages that you receive. You should also keep copies of all programs run and output generated. These records will help you in isolating and solving both user problems and VS Pascal problems.

Has the program changed since you last compiled or executed it successfully?
If so, examine the changes. If the error occurs in the changed code and cannot be corrected, note the change that caused the error. Keep copies of both the original and the changed programs.

Did you compile your program using the OPTIMIZE option?
If so, either recompile the program using the NOOPTIMIZE compile-time option, or use the VS Pascal Interactive Debugging Tool. If you can successfully compile the program using NOOPTIMIZE, you have bypassed the problem— not solved it — and you can run the program until a permanent solution is developed. Keep a record of the conditions and options in effect at the time the problem occurred.

Did you compile your program using the NOOPTIMIZE option?
If so, use either the VS Pascal Interactive Debugging Tool or Pascal debugging statements in the code to identify the problem.

Are you accessing routines provided by other products, such as IMS, from VS Pascal?
If so, consider whether new releases of these programs might be causing problems that appear related to VS Pascal.

Are you accessing other languages from VS Pascal?
If so, consider whether these languages might be causing problems that appear related to VS Pascal.

Does the problem look like a wait or loop?
If so, it might be a system problem. You should follow your site's procedures for resolving such problems. If no system problem exists, and the failure occurs while your program is running, the problem is probably a user error. Carefully check your VS Pascal source program to be sure it does not contain an endless loop. Adding a WRITELN statement and recompiling might help detect such an error.

Does the problem occur during run time?
If you are running in batch mode and the error results in a system abend, you might not have allotted enough time to compile or execute the program. Increasing the time allotment and recompiling or rerunning the program might solve this problem. It is also possible that

the region size specified in the execute step is too small. The problem may be resolved if you increase the region size and rerun the program.

If the problem persists at this point, read through the rest of this chapter until you find the section that applies to your problem, and follow the steps given there.

Compile-Time Problems

A problem at compile time will result in one or more of these responses:

- Error messages
- A wait, loop, or abend
- A trace-back report
- An incorrect compilation.

If you receive one of these responses while compiling a program, proceed to the appropriate section and follow the instructions given there. Remember to keep a record of any changes you make and their effects.

If You Receive Compile-Time Error Messages

1. For the full meaning of the message and recommended action to take, refer to the *VS Pascal Application Programming Guide*. You should also verify the correct syntax and usage of the code that produces the error. The *VS Pascal Language Reference* provides specific coding syntax and the *VS Pascal Application Programming Guide* provides usage guidelines.
2. If possible, correct the condition causing the message. Ensure that previous messages are not related to the current problem.
3. If the problem persists, create the smallest test case possible, as explained in Chapter 2, "Creating a Test Case" on page 11. Change the values of compile-time options specified in the test case to determine if one of the options is causing the problem. Sometimes, rearranging the order of the specified options can help isolate the problem.
4. If you cannot identify the cause of the problem, follow the instructions given in Chapter 3, "Diagnosis Aids" on page 13. If you are still unsuccessful, follow the instructions given in the section "VS Pascal Product Problems" on page 9.

If the Problem Is a Wait, Loop, or Abend

1. Create the smallest test case possible, as explained in Chapter 2, "Creating a Test Case" on page 11. Change the values of compile-time options specified in the test case to determine if one of the options is causing the problem. Sometimes, rearranging the order of the specified options can help isolate the problem.
2. Produce a dump and verify that the flow of control in the program is correct.
3. If you cannot locate the cause of the problem, follow the instructions given in Chapter 3, "Diagnosis Aids" on page 13. If you are still unsuccessful, follow the instructions given in the section "VS Pascal Product Problems" on page 9.

If You Receive a Trace-Back Report

1. Check the statement indicated, and the code immediately preceding the statement, for proper syntax and usage. The *VS Pascal Language Reference* provides specific coding syntax and the *VS Pascal Application Programming Guide* provides usage guidelines.
2. If the problem persists, create the smallest test case possible, as explained in Chapter 2, "Creating a Test Case" on page 11. Change the values of compile-time options specified in the test case to determine if one of the options is causing the problem. Sometimes, rearranging the order of the specified options can help isolate the problem.
3. If you cannot identify the cause of the problem, follow the instructions given in Chapter 3, "Diagnosis Aids" on page 13. If you are still unsuccessful, follow the instructions given in the section "VS Pascal Product Problems" on page 9.

If You Receive an Incorrect Compilation

1. Create the smallest test case possible, as explained in Chapter 2, "Creating a Test Case" on page 11. Change the values of compile-time options specified in the test case to determine if one of the options is causing the problem. Sometimes, rearranging the order of the specified options can help isolate the problem.
2. Produce a compiler listing and a console listing of the compilation.
3. If you cannot locate the cause of the problem, follow the instructions given in Chapter 3, "Diagnosis Aids" on page 13. If you are still unsuccessful, follow the instructions given in the section "VS Pascal Product Problems" on page 9.

Link-Time Problems

A problem at link time will result in one or more of these responses:

- EXEC or CLIST error messages
- Linkage editor error messages.

If you receive one of these responses at link time, first verify that you are not linking your program with an old release of the library. If not, proceed to the appropriate section and follow the instructions given there. Remember to keep a record of any changes you make and their effects.

If You Receive EXEC or CLIST Error Messages

1. For the full meaning of the message and recommended action to take, refer to the *VS Pascal Application Programming Guide*.
2. If possible, correct the condition causing the message.
3. If the problem persists, change the values of link-time options specified to determine if one of the options is causing the problem. Sometimes, rearranging the order of the specified options can help isolate the problem.
4. If you cannot identify the cause of the problem, follow the instructions given in Chapter 3, "Diagnosis Aids" on page 13. If you are still unsuccessful, follow the instructions given in the section "VS Pascal Product Problems" on page 9.

If You Receive Linkage Editor Error Messages under MVS

1. Rerun the job using the linkage editor options LIST and MAP in order to have more diagnostic information.
2. If the problem persists at this point, verify that the data sets in the link-edit step are in the correct search order. See the catalogued procedures in the *VS Pascal Application Programming Guide* for more information.
3. If the problem continues, change the values of link-time options specified to determine if one of the options is causing the problem.
4. If you cannot identify the cause of the problem, follow the instructions given in Chapter 3, "Diagnosis Aids" on page 13. If you are still unsuccessful, follow the instructions given in the section "VS Pascal Product Problems" on page 9.

Run-Time Problems

A problem at run time will result in one or more of these responses:

- Run-time error messages
- A wait or loop
- An abend
- Incorrect output.

If you receive one of these responses at run time, you should first:

- Ensure that your code contains no user errors. In most cases, run-time errors are the result of errors in an application program and not in VS Pascal.
- Use the VS Pascal Interactive Debugging Tool to perform a run-time analysis of the program and its error. See "VS Pascal Interactive Debugging Tool" on page 17 for more information on the VS Pascal Interactive Debugging Tool.
- Recompile the program. Specify these compile-time options to produce the maximum diagnostic information, unless one of these options masks the problem, or the opposite option is required to cause the problem:

  ```
  CHECK          PXREF
  GOSTMT         SOURCE
  HEADER         UCODE
  LIST           XREF(LONG)
  OUCODE
  ```

  See "Compile-Time Options" on page 13 for information on the compile-time options.
- Run the program with different run-time options. Run the program with SETMEM and MAINT, and without NOCHECK and NOSPIE, unless the opposite action is required to cause the problem. Note any changes from the previous run. See "Run-Time Options" on page 16 for information on the run-time options.

If the problem persists, proceed to the appropriate section and follow the instructions given there. Remember to keep a record of any changes you make and their effects.

If You Receive Run-Time Error Messages

1. For the full meaning of the message and recommended action to take, refer to the *VS Pascal Application Programming Guide*. You should also verify the correct syntax and usage of the code that produces the error. The *VS Pascal Language Reference* provides specific coding syntax and the *VS Pascal Application Programming Guide* provides usage guidelines.
2. If possible, correct the condition causing the message. Ensure that previous messages are not related to the current problem.
3. Run the program using the MAINT option. Use the trace-back report to determine if the error occurred in a VS Pascal routine. See *VS Pascal Application Programming Guide* for more information on trace-back reports.
4. If the problem persists, create the smallest test case possible, as explained in Chapter 2, "Creating a Test Case" on page 11. Change the values of compile-time, link-time, and run-time options specified in the test case to determine if one of the options is causing the problem. Sometimes, rearranging the order of the specified options can help isolate the problem.
5. If the problem continues, either use the Interactive Debugging Tool or add WRITE statements to the code to isolate the problem. Tracing the program flow might help you locate where, and determine why, the problem is occurring.
6. If you cannot identify the cause of the problem, follow the instructions given in Chapter 3, "Diagnosis Aids" on page 13. If you are still unsuccessful, follow the instructions given in the section "VS Pascal Product Problems" on page 9.

If the Problem Is a Wait or Loop

1. Create the smallest test case possible, as explained in Chapter 2, "Creating a Test Case" on page 11. Change the values of compile-time, link-time, and run-time options specified in the test case to determine if one of the options is causing the problem. Sometimes, rearranging the order of the specified options can help isolate the problem.
2. If the problem continues, either use the Interactive Debugging Tool or add WRITE statements to the code to isolate the problem. Tracing the program flow might help you locate where, and determine why, the problem is occurring.
3. If you cannot identify the cause of the problem, follow the instructions given in Chapter 3, "Diagnosis Aids" on page 13. If you are still unsuccessful, follow the instructions given in the section "VS Pascal Product Problems" on page 9.

If the Problem Is an Abend

1. Create the smallest test case possible, as explained in Chapter 2, "Creating a Test Case" on page 11. Change the values of compile-time, link-time, and run-time options specified in the test case to determine if one of the options is causing the problem. Sometimes, rearranging the order of the specified options can help isolate the problem.
2. If the problem continues, either use the Interactive Debugging Tool or add WRITE statements to the code to isolate the problem. Tracing the program flow might help you locate where, and determine why, the problem is occurring.

3. If you originally compiled the program using the NOCHECK compile-time option, recompile the program using CHECK and try to identify the cause of the error. If you originally ran the program using the NOCHECK or NOSPIE run-time options, rerun the program using the MAINT option.
4. If you cannot identify the cause of the problem, follow the instructions given in Chapter 3, "Diagnosis Aids" on page 13. If you are still unsuccessful, follow the instructions given in the section "VS Pascal Product Problems" on page 9.

If You Receive Incorrect Output

1. Create the smallest test case possible, as explained in Chapter 2, "Creating a Test Case" on page 11. Change the values of compile-time, link-time, and run-time options specified in the test case to determine if one of the options is causing the problem. Sometimes, rearranging the order of the specified options can help isolate the problem.
2. If the problem continues, either use the Interactive Debugging Tool or add WRITE statements to the code to isolate the problem. Tracing the program flow might help you locate where, and determine why, the problem is occurring.
3. If you cannot identify the cause of the problem, follow the instructions given in Chapter 3, "Diagnosis Aids" on page 13. If you are still unsuccessful, follow the instructions given in the section "VS Pascal Product Problems" on page 9.

Installation Problems

If you experience problems during installation you should:

1. Check the PSP bucket.
2. Read the Program Directory accompanying the installation tape.
3. If you still cannot resolve the problem, develop a keyword string based on the symptoms of the problem, as described in Chapter 4, "Building a Keyword String" on page 21.
4. As a last resort, re-install VS Pascal following the steps outlined in the proper installation guide for your system (*VS Pascal Installation and Customization for VM* or *VS Pascal Installation and Customization for MVS*).

Errors in VS Pascal Publications

If you discover an error in a VS Pascal publication, you should take one of these two courses of action, based upon the severity of the error:

1. If the problem is not severe, fill out the Reader's Comment Form attached to the back of the manual, and provide the problem description you developed. Include your name and return address so that IBM can respond to your comments.
2. If the problem is so severe as to affect other users, follow the instructions given in Chapter 4, "Building a Keyword String" on page 21.

Degraded Performance

If the system suffers from degraded performance, and you cannot correct the problem by system tuning, follow the instructions given in "Degraded Performance" on page 33.

VS Pascal Product Problems

If you have followed every suggestion given in this chapter, and you have determined that VS Pascal and not the program is causing the problem, you should note the sequence of events that leads to the failure and follow the instructions given in "Component Identification Keyword" on page 23.

Chapter 2. Creating a Test Case

A *test case* is a simplified version of a failing program that can reproduce the original error. When faced with a problem, using a test case can help you:

- Distinguish between an error in the program and an error in VS Pascal
- Locate and identify the error
- Choose keywords that best describe the problem.

A test case is created largely through a process of elimination. Depending upon the type of failure, you change various options and code and recompile or rerun the program until you discover the likely cause of the error.

The following list of failures and suggested option and code changes will help you create a test case. Start with the most obvious change that applies to the type of failure you are experiencing. If the program fails in the same way after changing one option or section of code, try the next most obvious change. Through this process of elimination, the test case will be the shortest and simplest version of the original program and will reproduce the original error.

Unreferenced identifiers
Remove any unreferenced identifiers. You can locate them using the XREF(LONG) compile-time option.

Compile-time failure
Remove any code that was not processed at the time of the failure.

Compile-time failure in a procedure or function
Remove all code and declarations from any other routines and keep only the procedure header and BEGIN/END block.

Run-time failure
Remove any code that was not executed, as well as references to the code that could cause a syntactically or semantically invalid program. Use the COUNT run-time option to find these code references.

Failure in a structured statement
Put the failing statement in the mainline code. (Structured statements include IF, CASE, WITH, FOR, WHILE, and REPEAT.)

Failure in a procedure or function
Place the failing code in the main program.

Failure using structured variables
Use scalar variables.

%INCLUDE compiler directives
Put all the %INCLUDE member code in the main file.

Segment units
Put all the declarations that are not in the program unit into the program unit.

Unrelated code

Remove any code that appears unrelated to the failure (except for code necessary to ensure the syntactic and semantic validity of the program).

Chapter 3. Diagnosis Aids

This chapter summarizes the features of VS Pascal that can help you diagnose a problem and locate the source of an error:

- Compile-time options
- Compiler directives
- Routines
- Link-time options
- Run-time options
- Interactive Debugging Tool.

For more information on these features, refer to the *VS Pascal Language Reference* and the *VS Pascal Application Programming Guide*.

Note: Any of the following features not listed in the above manuals are intended for diagnostic purposes only, and are marked "FOR DIAGNOSTIC PURPOSES ONLY". These features should be used only when IBM Service directs you to. IBM will not accept APARs related to these features.

Compile-Time Options

Some compile-time options can help you isolate problems in the compiler or in a user program. Figure 1 summarizes these options. For more information on compile-time options, refer to the *VS Pascal Application Programming Guide*.

Option	Description	Used to
CHECK	Enables run-time checking for a user program.	Catch run-time errors (should be used at all times).
COMPILER	Allows the compiler to be invoked with certain run-time options. FOR DIAGNOSTIC PURPOSES ONLY.	Invoke the compiler to check for an uninitialized variable in the compiler (**VM only**).
CONDPARM	Allows only selected sections of source code to be compiled.	Selectively include debugging statements (WRITE) in compilation.
DEBUG	Allows the VS Pascal Interactive Debugging Tool to be used with a program.	Provide information needed by the Interactive Debugging Tool.
HEADER	Places a header in the generated code of each user routine.	Identify routines in a program dump.

Figure 1 (Part 1 of 2). Compile-Time Options

Option	Description	Used to
LANGUAGE	Displays textual information in a language different from that selected at installation.	Generate text in the language requested by IBM service personnel.
LIST	Generates a pseudoassembler listing for the user program being compiled.	Determine the instruction causing a run-time error or abend and determine if the compiler is generating incorrect code.
LOG	Generates a processing log in the third compiler pass (code generation). FOR DIAGNOSTIC PURPOSES ONLY.	Determine if an error occurred when OPTIMIZE was specified.
OLOG	Generates a processing log in the second compiler pass (intermediate code optimization). FOR DIAGNOSTIC PURPOSES ONLY.	Diagnose errors occurring in the second compiler pass.
OUCODE	Generates a listing of the optimized intermediate code. FOR DIAGNOSTIC PURPOSES ONLY.	Diagnose errors occurring in the second compiler pass.
UCODE	Generates a listing of the intermediate code produced by the compiler. FOR DIAGNOSTIC PURPOSES ONLY.	Diagnose errors occurring in the first compiler pass.
WRITE	Allows %WRITE statements to output messages while a program is being compiled.	Diagnose errors occurring in the first compiler pass.

Figure 1 (Part 2 of 2). Compile-Time Options

Compiler Directives

Some compiler directives can help you isolate problems in the compiler or a user program. Figure 2 on page 15 summarizes these compiler directives. For more information on compiler directives, refer to the *VS Pascal Language Reference*.

Compiler Directive	Description	Used to
%CHECK	Enables run-time checking for a portion of a user program.	Turn checking on and off.
%LIST	Generates a pseudoassembler listing for a portion of the user program being compiled.	Check the compiler-generated code, isolate the address of an abend, and make the pseudoassembler code listing smaller.
%SELECT %WHEN %ENDSELECT	Marks a section of code that is selectively compiled.	Selectively include debugging statements (WRITE) in compilation.
%UHEADER	Allows a customized header to be placed after the compiler-generated header.	Put specific information in the header.
%WRITE	Issues a message during compilation of a program.	Isolate where the first pass of the VS Pascal compiler is failing.

Figure 2. Compiler Directives

Routines

Some compiler routines can help you isolate problems in a user program. Figure 3 summarizes these compiler routines. For more information on compiler routines, refer to the *VS Pascal Language Reference*.

Routine	Description	Used to
ADDR	Returns the address of a given variable.	Determine if a variable is where you expected it to be in storage.
AMPXMDMP	Dumps the storage pools currently in use. This is declared as: `PROCEDURE AMPXMDMP; EXTERNAL;` and can be called from a user program. FOR DIAGNOSTIC PURPOSES ONLY.	Check if the error occurred in the heap manager.
HBOUND	Returns the maximum subscript of an array variable or type.	Check if an array subscript was valid.

Figure 3 (Part 1 of 2). Compiler Routines

Routine	Description	Used to
HIGHEST	Returns the maximum value of a variable or type.	Check if an ordinal value was valid.
LBOUND	Returns the minimum subscript of an array variable or type.	Check if an array subscript was valid.
LOWEST	Returns the minimum value of a variable or type.	Check if an ordinal value was valid.
ONERROR	Allows run-time errors to be trapped and handled specially.	Specify a user-customized exit routine for run-time errors.
SIZEOF	Returns the size of a given variable or type.	Check storage allocations and out-of-storage conditions.
TRACE	Produces a trace-back report of the program executing.	Provide a "map" showing the path taken to a specific instruction.

Figure 3 (Part 2 of 2). Compiler Routines

Link-Time Option

Figure 4 summarizes a link-time option that can help you isolate problems in a user program. For more information on this option, refer to the *VS Pascal Application Programming Guide*.

Link-Time Option	Description	Used to
DEBUG	Activates the Interactive Debugging Tool.	Prepare to use the Interactive Debugging Tool or to generate a symbolic dump of the variables in the failing routine.

Figure 4. Link-Time Option

Run-Time Options

Some run-time options can help you isolate problems in a user program. Figure 5 on page 17 summarizes these run-time options. For more information on run-time options, refer to the *VS Pascal Application Programming Guide*.

Run-Time Option	Description	Used to
COUNT	Provides a statement execution count for a program.	Tune performance and show loops in code.
DEBUG	Activates the VS Pascal Interactive Debugging Tool.	Isolate failures in a program.
ERRCOUNT	Terminates program execution after a specified number of errors have occurred.	Reduce amount of information for service.
ERRFILE	Directs run-time error messages to a specified output device.	Trap error output for service uses.
LANGUAGE	Displays textual information in a language different from that selected at installation.	Generate text in the language requested by IBM service personnel.
MAINT	Includes all run-time library routines called in any trace backs.	Check if an error occurred in the VS Pascal run-time environment.
SETMEM	Initializes local variable storage to a common value at each routine invocation.	Detect uninitialized variables in a user program.

Figure 5. Run-Time Options

VS Pascal Interactive Debugging Tool

You can use the VS Pascal Interactive Debugging Tool to help identify the statement causing an error. With the VS Pascal Interactive Debugging Tool, you can:

- Suspend program execution
- Continue execution
- Examine variable values
- Display storage
- Trace program execution
- View a program trace-back report
- Count statement execution
- Issue system commands (CMS only).

You can also use the VS Pascal Interactive Debugging Tool to debug optimized code, with some restrictions. For more information on the debugging tool, see the *VS Pascal Application Programming Guide*.

There is one diagnostic command for the VS Pascal Interactive Debugging Tool — the DG command. The DG command displays the values contained in general registers 12 and 13. This command is intended FOR DIAGNOSTIC PURPOSES ONLY.

Part 2. Identifying the Problem with Keywords

Chapter 4. Building a Keyword String

VS Pascal Compiler and Library product failures can be described using keywords. A *keyword* is a word or abbreviation used to describe a single aspect of a product failure. A set of keywords for a given problem is called a *keyword string*. Keywords ensure that any two people report the same type of problem caused by the same program error in identical terms.

This chapter explains how to use keywords describing various types of problems to develop a full keyword string that includes:

- A component identification keyword that identifies the VS Pascal product involved
- A release level keyword that identifies the release and modification level under which you are operating
- Additional keywords that identify the specific type of problem you are experiencing.

Before Continuing

Ensure that you have followed all instructions outlined in Chapter 1, Isolating the Problem that relate to the problem you are experiencing.

Using Keywords

A keyword string is used as a search argument in an IBM software support data base, such as the Software Support Facility (SSF) or the Early Warning System (EWS). In order to search the data base, you must develop a keyword string that accurately describes the problem.

If the problem has already been entered in the software support data base using the same keyword string you developed, your search will turn up the matching entry and, usually, a solution to the problem.

If the problem has not been entered in the software support data base, you can use the keyword string you developed to prepare an APAR. For additional information on keywords and APAR preparation, see *Field Engineering Programming System General Information.*

Figure 6 on page 22 shows a flowchart of a VS Pascal keyword string.

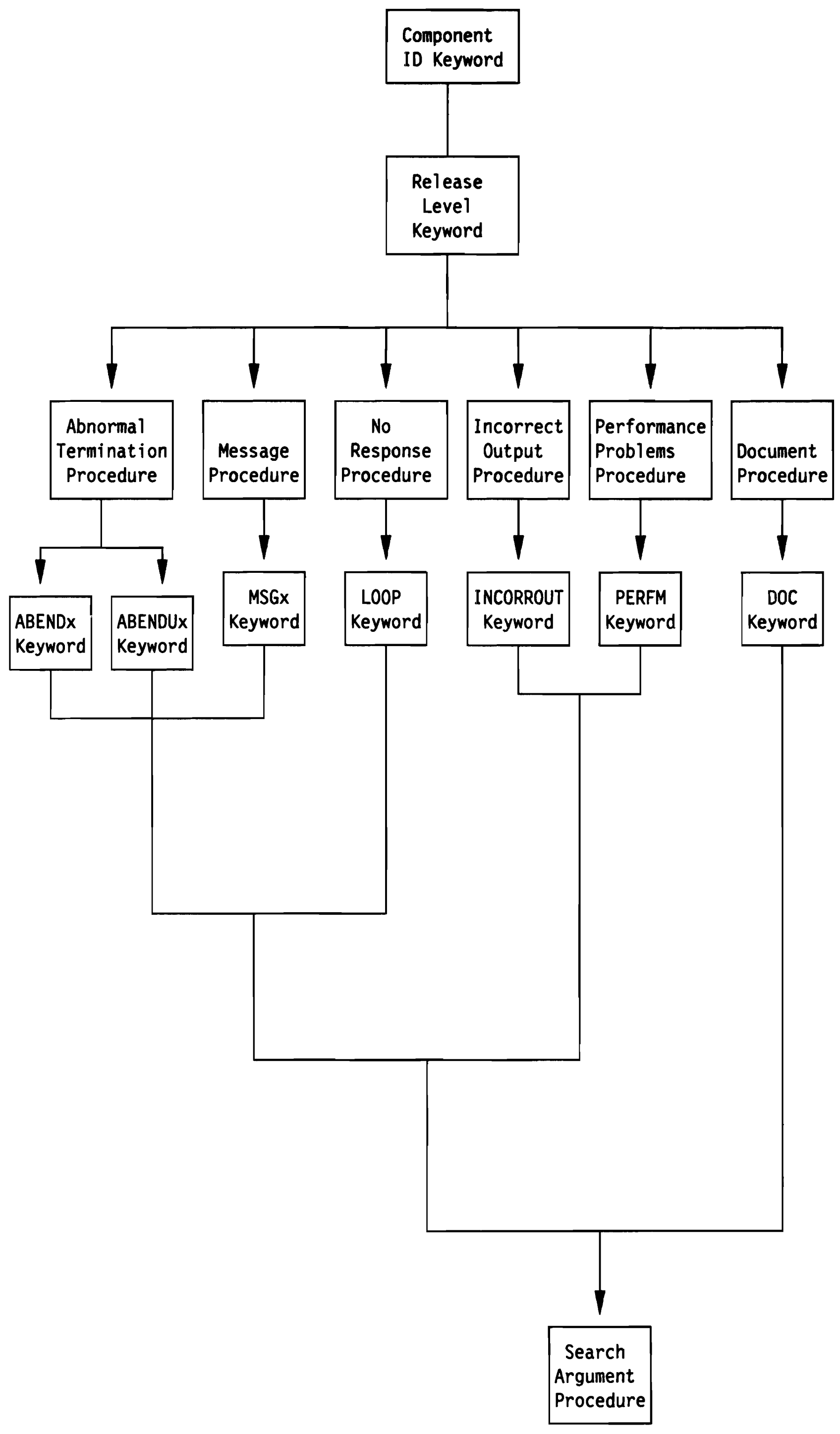

Figure 6. Flowchart of a VS Pascal Keyword String

Component Identification Keyword

The component identification number is the first keyword in a keyword string. It identifies the library within the software support data base that contains known problem descriptions for the product. A search of the software support data base with this keyword alone will turn up all reported problems for the entire licensed program.

566876701 is the component identification keyword for the VS Pascal Compiler and Library. **566871701** is the component identification keyword for the VS Pascal Library.

Continue the diagnosis with "Release Level Keyword."

Release Level Keyword

The release level is the second keyword in a keyword string. It identifies the specific release level of VS Pascal under which you were operating at the time the problem occurred. A search of the software support data base with this keyword, and the component identification keyword, will turn up all reported problems for a specific release and modification level of the licensed program.

The release and modification level (denoted by VS PASCAL RELEASE r.m) is located at the top of the first page of the most recent compiler listing for your test case. Specify the release level keyword, using the format Rrm, where "r" is the release level and "m" is the modification level.

Continue the diagnosis with "Keywords to Use for Various Types of Problems."

Keywords to Use for Various Types of Problems

Select the symptom that best describes the problem from the list of symptoms in Figure 7. If more than one keyword describes the problem, use the keyword that appears first in the list.

Symptom	Description	Keyword	See
Abend (Abnormal Termination)	The compiler or user program has terminated abnormally without a message, and you have performed the steps in "If the Problem Is a Wait, Loop, or Abend" on page 4 and "If the Problem Is an Abend" on page 7.	ABEND*x* ABENDU*x*	"Abnormal Terminations (Abends)" on page 25

Figure 7 (Part 1 of 2). VS Pascal Symptom Table

Symptom	**Description**	**Keyword**	**See**
Error Messages	A message indicates a compiler or user program error, or seems itself to be an error, and you have performed the steps in "If You Receive Compile-Time Error Messages" on page 4.	MSG*x*	"Error Messages" on page 26
Nothing Is Happening	The program seems to be doing nothing, or is doing something repetitively, and you have performed the steps in "If the Problem Is a Wait, Loop, or Abend" on page 4.	LOOP	"Nothing Is Happening" on page 31
Errors in Publications	Information in one of the VS Pascal publications is incorrect or missing, and you have read the information in "Errors in VS Pascal Publications" on page 8.	DOC	"Errors in VS Pascal Publications" on page 31
Incorrect Output	Output from the program is missing or invalid, and you have performed the steps in "If You Receive Incorrect Output" on page 8.	INCORROUT	"Incorrect Output" on page 32
Degraded Performance	The performance of the program is degraded, and you have performed the steps in "Degraded Performance" on page 9.	PERFM	"Degraded Performance" on page 33

Figure 7 (Part 2 of 2). VS Pascal Symptom Table

Abnormal Terminations (Abends)

Before proceeding, read "If the Problem Is a Wait, Loop, or Abend" on page 4 if the problem occurs during compilation, or "If the Problem Is an Abend" on page 7 if the problem occurs during run time.

Use the ABEND*x* keyword when a program exception occurs:

- Within VS Pascal
- Within a VS Pascal-compiled program
- Whenever the VS Pascal compiler or a VS Pascal-compiled program terminates without a message.

Do not use this keyword if termination is accompanied by a message with the prefix "AMP". For those situations, refer to "Error Messages" on page 26.

Do not use this keyword if termination was forced because too much time was spent in a wait state or in an endless loop. For those situations, refer to "Nothing Is Happening" on page 31.

Procedure

If the problem occurs during installation, use the modifier INSTALL.

If the problem occurs during compile time or run time:

1. Use the CMPL modifier if the problem occurs during the compilation of your program. If the problem occurs during run time, use the EXEC modifier.
2. Determine with which compile-time, link-time, and run-time options the failure occurs. If the failure occurs only when using certain options, indicate those options in the keyword string. Select the appropriate modifier keyword from the list shown in Chapter 5, "Keywords for Options" on page 35.
3. Determine which statement causes the problem.

   Use the following keywords as modifier keywords to describe the statement you think is causing the error.

AND	IF	RANGE[1]
ARRAY	IN	RECORD
ASSERT[1]		REF[1]
	LABEL	REPEAT
BEGIN	LEAVE[1]	RETURN[1]
CASE	MOD	SET
CONST		SPACE[1]
CONTINUE[1]	NIL	STATIC[1]
	NOT	
DEF[1]		THEN[4]
DIV	OF[6]	TO[3]
DO[2]	OR	TYPE
DOWNTO[3]	OTHERWISE[1 7]	
		UNTIL[8]
ELSE[4]	PACKED	
END[5]	PROCEDURE	VALUE[1]
	PROGRAM	VAR
FILE		
FOR		WHILE
FUNCTION		WITH
GOTO		XOR[1]

Figure 8. Statement Names as Modifier Keywords

Notes to Figure 8:

1 LANGLVL(EXTENDED) only.
2 Use FOR, WHILE, or WITH as keyword as appropriate.
3 Use FOR as keyword.
4 Use IF as keyword.
5 Use BEGIN, CASE, or RECORD as keyword as appropriate.
6 Use ARRAY, CASE, FILE, SET, or SPACE as keyword as appropriate.
7 Use CASE as keyword.
8 Use REPEAT as keyword.

Example

A keyword string showing an 0C1 abend in the library has this format:

```
Component Identification:  566871701
Release Level:             R20
Type of Failure:           ABEND0C1
Modifiers:                 EXEC
                           GOTO
```

Error Messages

Before proceeding, read "If You Receive Compile-Time Error Messages" on page 4 if the problem occurs during compilation, or "If You Receive Run-Time Error Messages" on page 7 if the problem occurs during run time.

Use the MSG*x* keyword for any of these conditions:

- A message indicates either a compiler or user program error.
- A message is issued under conditions that should not cause it to be issued.
- A message contains invalid data or is missing data.

Chapter 5. Keywords for Options

This chapter lists the keywords used to describe VS Pascal compile-time, link-time, and run-time options in keyword strings.

Compile-Time Options

Use the modifier keywords shown below to identify compile-time options and VS Pascal EXEC and CLIST options in the keyword string.

Option	Modifier Keywords
CHECK or NOCHECK	CHECK, NOCHECK
CONDPARM	CONDPARM
CONSOLE	CONSOLE
DEBUG or NODEBUG	DEBUG, NODEBUG
DDNAME	DDNAME
DISK	DISK
FLAG	FLAG
GOSTMT or NOGOSTMT	GOSTMT, NOGOSTMT
GRAPHIC or NOGRAPHIC	GRAPHIC, NOGRAPHIC
HEADER or NOHEADER	HEADER, NOHEADER
LANGLVL (EXTENDED) LANGLVL (ANSI83)	LANGLVL EXTENDED LANGLVL ANSI83
LANGUAGE(*ccc*)	LANGUAGE *ccc* (see note)
LIB or NOLIB	LIB or NOLIB
LINECOUNT	LINECOUNT
LIST or NOLIST	LIST, NOLIST
MARGINS	MARGINS
OBJECT or NOOBJECT	OBJECT or NOOBJECT
OPTIMIZE or NOOPTIMIZE	OPT, NOOPT
PAGEWIDTH	PW
PRINT or NOPRINT	PRINT or NOPRINT
PXREF or NOPXREF	PXREF, NOPXREF
SEQUENCE or NOSEQUENCE	SEQ, NOSEQ
SOURCE or NOSOURCE	SOURCE, NOSOURCE
STDFLAG	STDFLAG

Figure 13 (Part 1 of 2). VS Pascal Compile-Time Options

Option	Modifier Keywords
SYSPRINT	SYSPRINT
TRANLIB or NOTRANLIB	TRANLIB, NOTRANLIB
WRITE or NOWRITE	WRITE, NOWRITE
XREF (LONG|SHORT) or NOXREF	LONGXREF, SHRTXREF, NOXREF

Figure 13 (Part 2 of 2). VS Pascal Compile-Time Options

Note to Figure 13 on page 35: In LANGUAGE(*ccc*), the identifier *ccc* represents one of the three languages that VS Pascal supports:

UEN	Uppercase English
ENG	Mixed-case English
JPN	Japanese

Link-Time Options

Use the modifier keywords shown below to identify link-time options in the keyword string.

Option	Modifier Keywords
DEBUG or NODEBUG	LDEBUG or LNODEBUG
LIB	LLIB
NAME	LNAME
OBJECT	LOBJECT
TRANLIB or NOTRANLIB	LTRANLIB or LNOTRANLIB
XA or NOXA	LXA or LNOXA

Figure 14. VS Pascal Link-Time Options

Run-Time Options

Use the modifier keywords shown below to identify run-time options in the keyword string.

Option	Modifier Keywords
COUNT	RCOUNT
DEBUG	RDEBUG
ERRCOUNT	RERRCNT
ERRFILE	RERRFILE
HEAP	RHEAP

Figure 15 (Part 1 of 2). VS Pascal Run-Time Options

Option	Modifier Keywords
LANGUAGE(*ccc*)	RLANGUAGE *ccc* (see note)
MA!NT	RMAINT
NOCHECK	RNOCHECK
NOSPIE	RNOSPIE
STACK	RSTACK
SETMEM	RSETMEM

Figure 15 (Part 2 of 2). VS Pascal Run-Time Options

Note to Figure 15 on page 36: In LANGUAGE(*ccc*), the identifier *ccc* represents one of the three languages that VS Pascal supports:

UEN	Uppercase English
ENG	Mixed-case English
JPN	Japanese

Do not use this keyword if you receive a message as the result of a run-time abnormal termination. In that case, see "If the Problem Is an Abend" on page 7. If you receive a message as the result of a compile-time abnormal termination, read "If the Problem Is a Wait, Loop, or Abend" on page 4.

Messages

... or semantic error message is identified by a string ... message number. For the purpose of ... a message identifier of

[illegible]

Do not use this keyword if you receive a message as the result of a run-time abnormal termination. In that case, see "If the Problem Is an Abend" on page 7. If you receive a message as the result of a compile-time abnormal termination, read "If the Problem Is a Wait, Loop, or Abend" on page 4.

Syntax and Semantic Messages
Each VS Pascal compiler syntax or semantic error message is identified by a string of three characters: *nnn*, where *nnn* is the message number. For the purpose of identification, consider each of these messages to have a message identifier of AMPL*nnn*.

Compiler Messages
Every VS Pascal compiler message other than syntax and semantic error messages is identified by a string of eight characters, AMP*pnnnc*, where:

AMP	is the message prefix identifying all VS Pascal compiler messages.
p	is a letter representing the compiler phase issuing the message. *p* will be either "L", "O", or "T".
nnn	is the message number.
c	is either an "I" for informational messages, a "W" for warning messages, an "E" for normal error conditions, or an "S" for severe error conditions.

Library Messages
Every VS Pascal library message is identified by a string of eight characters, AMPX*nnnc*, where:

AMPX	is the message prefix identifying all VS Pascal library messages.
nnn	is the message number.
c	is either an "I" for informational messages, an "E" for normal error conditions, or an "S" for severe error conditions.

Debugging Messages
Every VS Pascal debugging message is identified by a string of eight characters, AMPD*nnnc*, where:

AMPD	is the message prefix identifying all VS Pascal debugging messages.
nnn	is the message number.
c	is either an "I" for informational messages, a "W" for warning messages, an "E" for normal error conditions, or an "S" for severe error conditions.

CMS EXEC Messages
Every CMS EXEC message is identified by a string of eight characters, AMPE*nnnc*, where:

AMPE	is the message prefix identifying all CMS EXEC messages.
nnn	is the message number.
c	is either an "I" for informational messages, a "W" for warning messages, an "E" for normal error conditions, or an "S" for severe error conditions.

TSO CLIST Messages

Every TSO CLIST message is identified by a string of eight characters, AMPC*nnnc*, where:

AMPC is the message prefix identifying all TSO CLIST messages.

nnn is the message number.

c is either an "I" for informational messages, a "W" for warning messages, an "E" for normal error conditions, or an "S" for severe error conditions.

Procedure

1. For each VS Pascal message issued, replace the *x* of MSG*x* keyword with the complete message identifier but do not include the severity character (if any). Remember that syntax and semantic errors have an implied prefix of AMPL.
2. For the module name keyword, use the name of the module, routine, and statement number that caused the message to be issued.

   To determine which module, routine, and statement number caused the error, look at the top-most routine and module shown in the trace-back report (as shown in Figure 9). The top-most module, routine, and statement number are usually the ones you need to report. If no module or statement number is shown in the trace-back report, use the routine name and address as modifiers.

```
 AMPX036S Assertion failure checking error
 AMPL999S COMPILER ERROR: NOTIFY VS PASCAL SUPPORT
      TRACE BACK OF CALLED ROUTINES
 ROUTINE                          STMT AT ADDRESS IN MODULE
 MAXLENFUNC                         25    0006119C   AMPLSTRG
 CALL                               56    00060BD6   AMPLCALL
 FACTOR                             16    0006B482   AMPLFACT
 TERM                                3    00055F30   AMPLEXPR
 SIMPLEEXPRESSION                    7    00056EA6   AMPLEXPR
 EXPRESSION                          3    0005767E   AMPLEXPR
 WRITE                              52    0005A3F4   AMPLREAD
 CALL                               13    000608CA   AMPLCALL
 STATEMENT                          84    0003CBBE   AMPLSTMT
 BODY                               92    0002180C   AMPLMAIN
 BLOCK                              51    0002202E   AMPLMAIN
 PROGRAMME                          36    000225CE   AMPLMAIN
 <MAIN-PROGRAM>                     42    000248A4   AMPLMAIN
 VSPASCAL                                 0002020A

AMPX900S EXECUTION NOT ALLOWED TO CONTINUE
```

Figure 9. A Trace-Back Report Identifying a Failing Module and Routine

The type of failure shown in Figure 9 is "MSGAMPX036 MSGAMPL999 MSGAMPX900" and the modifiers are "AMPLSTRG MAXLENFUNC 25".

There are several cases in which the top-most module and routine are not reported. Figure 10, Figure 11, and Figure 12 show examples of these types of trace-back reports.

For example, when compiling, ignore the top-most occurrences of the following modules and routines:

Module	**Routine**
AMPLINSY	ERROR
AMPLINSY	WARNING
AMPLINSY	INFORMATION
AMPLINSY	NORMAL_MSG
AMPLINSY	IMMEDIATE_MSG
AMPOMISC	ERROR
AMPOMISC	OVERFLOW
AMPTXMSC	ERROR
AMPTXMSC	FATAL

An example of the trace-back report when compiling is shown in Figure 10.

```
AMPT998S *** TRANSLATOR ERROR:  NOTIFY VS PASCAL SUPPORT ***
FAILED AT HALFMULT/<MAIN-PROGRAM>/1
     TRACE BACK OF CALLED ROUTINES
 ROUTINE                          STMT AT ADDRESS IN MODULE
 FATAL                               9    00026244   AMPTXMSC
 BD                                  3    0003D264   AMPTXA
 BXD                                 7    0003D4C2   AMPTXA
 XMPYI                              21    0005823C   AMPTXE
 GENBINARY                          24    0003EE36   AMPTGEN
 EXPRESSION                         37    0002DA50   AMPTTRAN
 EXPRESSION                          9    0002D85C   AMPTTRAN
 EXPRESSION                          9    0002D85C   AMPTTRAN
 EXPRESSION                          6    0002D818   AMPTTRAN
 DOBB                               13    0002DFD2   AMPTTRAN
 TRANSLATE                          54    0002E47E   AMPTTRAN
 <MAIN-PROGRAM>                      7    00021EEE   AMPTMAIN
 VSPASCAL                                 0002020A
```

Figure 10. A Trace-Back Report Showing an Error-Handling Module as the Last Action Module

The failure shown in Figure 10 is "MSGAMPT998". The modifiers are "AMPTXA BD 3" in this case because AMPTXMSC/FATAL is ignored.

When using the MAINT run-time option, ignore the top-most occurrences of the following modules and routines:

Module	**Routine**
AMPXTRAC	AMPXTRAC
AMPXCHKR	AMPXERR
AMPXCHKR	AMPXDIAG
AMPXCHKR	AMPXCHKR
AMPXIO	AMPXIOER

In addition, ignore all occurrences of routine AMPXMEMF. Figure 11 shows an example of a trace-back report issued when the MAINT run-time option is in effect.

```
AMPX023E Exponent underflow exception
     TRACE BACK OF CALLED ROUTINES
ROUTINE                          STMT AT ADDRESS IN MODULE
AMPXTRAC                            9    0002E756   AMPXTRAC
AMPXERR                            81    000262F6   AMPXCHKR
AMPXDIAG                           37    00026B82   AMPXCHKR
CHARX                              17    00022B00   AMPXPICT
PICTURE                             7    000242B8   AMPXPICT
<MAIN-PROGRAM>                      1    000200F0   PICTBUG
AMPXMEMF                                 00025808
VSPASCAL                                 000203FA
```

Figure 11. A Trace-Back Report Issued When the MAINT Run-Time Option Is in Effect

The type of failure shown in Figure 11 is "MSGAMPX023". The modifiers are "AMPXPICT CHARX 17" because AMPXTRAC/AMPXTRAC, AMPXCHKR/AMPXERR, and AMPXCHKR/AMPXDIAG should not be reported.

If, after ignoring the specified routines, the top-most routine is in your program (the module generally does not start with AMPX), then you should not report the failing module and routine.

An example of a trace-back report for a probable user error is shown in Figure 12.

```
AMPX014S Protection exception
     TRACE BACK OF CALLED ROUTINES
ROUTINE                          STMT AT ADDRESS IN MODULE
AMPXTRAC                            9    00028C46   AMPXTRAC
AMPXERR                            81    00021C3E   AMPXCHKR
AMPXDIAG                           37    000224CA   AMPXCHKR
<MAIN-PROGRAM>                      1    000200CA   TEMP
AMPXMEMF                                 00021150
VSPASCAL                                 000202F2

AMPX900S EXECUTION NOT ALLOWED TO CONTINUE
```

Figure 12. A Trace-Back Report for a Probable User Error

The type of failure shown in Figure 12 is "MSGAMPX014 MSGAMPX900". There are no modifiers because the error appears to have occurred in the main program of TEMP.

3. Determine with which compile-time, link-time, and run-time options the failure occurs. If the failure occurs only when using certain options, indicate those options in the keyword string. Select the appropriate modifier keyword from the list shown in Chapter 5, "Keywords for Options" on page 35.

Examples

If the message 9 occurs during compilation, use the following keywords:

```
Component Identification:  566871701
Release Level:             R20
Type of Failure:           MSGAMPL009
Modifier:                  CMPL
```

If the message AMPX067E occurs during execution, use the following keywords:

```
Component Identification:  566871701
Release Level:             R20
Type of Failure:           MSGAMPX067
Modifier:                  EXEC
```

Nothing Is Happening

Before proceeding, read "If the Problem Is a Wait, Loop, or Abend" on page 4.

Use the LOOP keyword when a program seems to be doing nothing or is doing something repetitively.

Procedure

If the problem occurs during installation, use the modifier INSTALL.

If the problem occurs during compile time or run time:

1. Use the modifier CMPL if the loop occurs during compilation. If the loop occurs during execution, use the modifier EXEC.
2. Determine with which compile-time, link-time, and run-time options the failure occurs. If the failure occurs only when using certain options, indicate those options in the keyword string. Select the appropriate modifier keyword from the list shown in Chapter 5, "Keywords for Options" on page 35.

Example

If the compiler appears to loop, the set of keywords describing the problem has this format:

```
Component Identification:  566876701
Release Level:             R20
Type of Failure:           LOOP
Modifier:                  CMPL
```

Errors in VS Pascal Publications

Before proceeding, read "Errors in VS Pascal Publications" on page 8.

Use the DOC keyword when a program problem appears to be caused by incorrect or missing information in a VS Pascal publication.

Procedure

1. Locate the page in the document where the information is incorrect or missing and prepare a description of the error and the problem it caused.

   If the problem might affect other users, use a keyword string to search the software support data base to determine if IBM has a record of the problem. Use the DOC keyword and the order number on the cover of the document (omitting the hyphens) as the failure keyword. For example, rather than using

LY27-9525-1 (for this book), use LY2795251. You should have the following set of keywords:

```
Component Identification:  566876701
Release Level:             R20
Type of Failure:           DOC LY2795251
```

2. If your search argument is not in the software support data base, try another search using this format:

   ```
   Component Identification:  566876701
   Release Level:             R20
   Type of Failure:           DOC LY279525*
   ```

   This format searches for all entries for the specified document number.

3. If your search does not turn up an identical entry, the problem has not been entered in the software support data base, and you will be asked to submit a severity 4 (DOC) APAR.

Incorrect Output

Before proceeding, read "If You Receive Incorrect Output" on page 8.

Use the INCORROUT keyword when output appears to be incorrect or missing, and the program terminates normally otherwise.

Procedure

If the problem occurs during installation, use the modifier INSTALL.

If the problem occurs during compile time or run time:

1. Use the modifier CMPL if the incorrect output occurs during compilation. If the incorrect output occurs during execution, use the modifier EXEC.
2. If you suspect incorrect or missing output from a compilation or execution that otherwise compiled or executed successfully, select a modifier keyword from the following list to describe the type of error in the output. You can also use these modifier keywords for an installation problem.

Modifier	Type of Incorrect Output
MISSING	Some expected output was missing.
DUPLICATE	Some records or data were duplicated, but not repeated endlessly (in that case, see "If the Problem Is a Wait, Loop, or Abend" on page 4).
INVALID	The proper amount of output appeared, but it was not what was expected.

3. If the failure occurred during compilation, select another modifier keyword from the following list to describe the portion of the output in which the error occurred.

Modifier	Portion of Output in Error
ERROR	Error summary of listing
EXTSYM	External symbol listing
OBJECT	Machine-language object program

OPTS	Compiler parameters and options summary
PXREF	Page cross-reference listing
SOURCE	Source listing
STAT	Compilation statistics summary
TERMERR	Console error listing
TRACE	Trace-back report
XREF	Cross-reference listing

4. Determine with which compile-time, link-time, and run-time options the failure occurs (this step is not appropriate for installation problems). If the failure occurs only when using certain options, indicate those options in the keyword string. Select the appropriate modifier keyword from the list shown in Chapter 5, "Keywords for Options" on page 35.

Example

If you believe the compiler produced an incorrect cross-reference listing only when compiling with the XREF(LONG) option, the keyword string has this format:

```
Component Identification:  566876701
Release Level:             R20
Type of Failure:           INCORROUT
Modifiers:                 CMPL
                           INVALID XREF
                           LONGXREF
```

Degraded Performance

Use the PERFM keyword when a performance problem cannot be corrected by system tuning, and performance is below expectations as documented in an IBM product publication.

Procedure

1. Record the actual performance and the expected performance measurements for your system configuration. Note the order number and page of the IBM document that is the source of your performance expectations. You will be asked for this information if you contact the IBM Support Center. If you prepare materials for an APAR, you should also include this information in the error description.
2. Use the modifier CMPL if the performance problem occurs during compilation. If the performance problem occurs during execution, use the modifier EXEC.
3. Determine with which compile-time, link-time, and run-time options the failure occurs. If the failure occurs only when using certain options, indicate those options in the keyword string. Select the appropriate modifier keyword from the list shown in Chapter 5, "Keywords for Options" on page 35.

Example

If a performance problem occurs during compilation, the keyword string has this format:

```
Component Identification:  566876701
Release Level:             R20
Type of Failure:           PERFM
Modifiers:                 CMPL
```

Chapter 5. Keywords for Options

This chapter lists the keywords used to describe VS Pascal compile-time, link-time, and run-time options in keyword strings.

Compile-Time Options

Use the modifier keywords shown below to identify compile-time options and VS Pascal EXEC and CLIST options in the keyword string.

Option	**Modifier Keywords**
CHECK or NOCHECK	CHECK, NOCHECK
CONDPARM	CONDPARM
CONSOLE	CONSOLE
DEBUG or NODEBUG	DEBUG, NODEBUG
DDNAME	DDNAME
DISK	DISK
FLAG	FLAG
GOSTMT or NOGOSTMT	GOSTMT, NOGOSTMT
GRAPHIC or NOGRAPHIC	GRAPHIC, NOGRAPHIC
HEADER or NOHEADER	HEADER, NOHEADER
LANGLVL (EXTENDED) LANGLVL (ANSI83)	LANGLVL EXTENDED LANGLVL ANSI83
LANGUAGE(*ccc*)	LANGUAGE *ccc* (see note)
LIB or NOLIB	LIB or NOLIB
LINECOUNT	LINECOUNT
LIST or NOLIST	LIST, NOLIST
MARGINS	MARGINS
OBJECT or NOOBJECT	OBJECT or NOOBJECT
OPTIMIZE or NOOPTIMIZE	OPT, NOOPT
PAGEWIDTH	PW
PRINT or NOPRINT	PRINT or NOPRINT
PXREF or NOPXREF	PXREF, NOPXREF
SEQUENCE or NOSEQUENCE	SEQ, NOSEQ
SOURCE or NOSOURCE	SOURCE, NOSOURCE
STDFLAG	STDFLAG

Figure 13 (Part 1 of 2). VS Pascal Compile-Time Options

Option	Modifier Keywords
SYSPRINT	SYSPRINT
TRANLIB or NOTRANLIB	TRANLIB, NOTRANLIB
WRITE or NOWRITE	WRITE, NOWRITE
XREF (LONG|SHORT) or NOXREF	LONGXREF, SHRTXREF, NOXREF

Figure 13 (Part 2 of 2). VS Pascal Compile-Time Options

Note to Figure 13 on page 35: In LANGUAGE(*ccc*), the identifier *ccc* represents one of the three languages that VS Pascal supports:

UEN	Uppercase English
ENG	Mixed-case English
JPN	Japanese

Link-Time Options

Use the modifier keywords shown below to identify link-time options in the keyword string.

Option	Modifier Keywords
DEBUG or NODEBUG	LDEBUG or LNODEBUG
LIB	LLIB
NAME	LNAME
OBJECT	LOBJECT
TRANLIB or NOTRANLIB	LTRANLIB or LNOTRANLIB
XA or NOXA	LXA or LNOXA

Figure 14. VS Pascal Link-Time Options

Run-Time Options

Use the modifier keywords shown below to identify run-time options in the keyword string.

Option	Modifier Keywords
COUNT	RCOUNT
DEBUG	RDEBUG
ERRCOUNT	RERRCNT
ERRFILE	RERRFILE
HEAP	RHEAP

Figure 15 (Part 1 of 2). VS Pascal Run-Time Options

Option	Modifier Keywords
LANGUAGE(*ccc*)	RLANGUAGE *ccc* (see note)
MA!NT	RMAINT
NOCHECK	RNOCHECK
NOSPIE	RNOSPIE
STACK	RSTACK
SETMEM	RSETMEM

Figure 15 (Part 2 of 2). VS Pascal Run-Time Options

Note to Figure 15 on page 36: In LANGUAGE(*ccc*), the identifier *ccc* represents one of the three languages that VS Pascal supports:

UEN	Uppercase English
ENG	Mixed-case English
JPN	Japanese

Chapter 6. Searching for a Solution with Your Keyword String

This chapter explains how to use the keyword string you have developed to search the software support data base. You can conduct the search yourself if you have access to the correct data base, or you can request that IBM conduct the search.

If You Conduct the Search

If you conduct the search yourself, you should follow these rules:

- Use only the keywords provided in this manual.
- Spell keywords exactly as they are presented in this manual.
- Include all appropriate keywords in any discussion with IBM support personnel or in any written description of your problem.

The following steps will help you conduct your search of the software support data base.

1. Search the software support data base using the full set of keywords you developed. Given this format:

   ```
   Component Identification:  566876701
   Release Level:             R20
   Type of Failure:           MSGAMPL999
                              MSGAMPX059
   Module Name:               AMPLINSY
   Routine Name:              ENDOFLINE
   Statement Number:          2
   ```

   your keyword string will be:

   566876701 R20 MSGAMPL999 MSGAMPX059 AMPLINSY ENDOFLINE 2

2. Eliminate from the list of possible matches those APAR solutions which already apply to your system.

3. Compare the closing description of each remaining APAR with the symptoms of your current problem.

4. If you find an APAR that describes your problem, find out if there is a corresponding PTF (program temporary fix). You can order the application PTF from the IBM Support Center. You might already have the PTF at your site, and only need to install it from the correct program update tape (PUT).

   Note: Information on applying a specific PTF is provided in the cover letter sent with the PTF.

5. If you do not find an APAR that describes your problem, broaden your search using these techniques:

   a. One by one, eliminate keywords from the right of your keyword string. (The keyword string was developed in a specific sequence to make this technique possible.)

   b. Consider using a synonym as a replacement for a modifier keyword. The problem might have been entered into the data base using a slightly different expression than the now-recommended format.

   c. Consider using the other Component Identification keyword. It is possible that a library problem might have been reported as a compiler program, or vice-versa.

6. If you still cannot find an appropriate APAR, go to Chapter 7, "Preparing an APAR" on page 43.

If You Request that IBM Conduct the Search

You must contact the IBM Support Center if you want IBM to conduct the search. When you contact the IBM Support Center you will be asked to provide:

- Your customer number
- The full set(s) of keywords you developed.

Part 3. Reporting the Problem to IBM

Chapter 7. Preparing an APAR

This chapter explains what an authorized program analysis report (APAR) is and how to submit an APAR to the IBM Support Center.

When to Submit an APAR

An *authorized program analysis report*, known as an APAR, identifies a problem caused by a suspected defect in a current unaltered release of a program.

You should prepare an APAR only after you have exhausted all of the preceding diagnostic procedures and you cannot find a solution in an IBM data base.

If you need to submit an APAR, contact the IBM Support Center and be prepared to supply:

- Your customer number
- The keyword string(s) you used to search the software support data base.

The support personnel will review the problem with you, and give you an APAR number when you and they have agreed that an APAR is necessary.

Materials to Submit with Your APAR

It is essential to supply all of the required documentation relating to your problem when you submit an APAR. Otherwise, IBM will return the APAR.

If you are resubmitting an APAR that has been returned to you for further information, you must supply the additional requested documentation. Remember to indicate the number of the original APAR.

The following checklist summarizes the materials you must submit with an APAR. A complete description of each type of material follows the checklist.

Materials	**When Required**
Original source or failing test case	Always
Load library information	Run-time problems only
Input data set information	Run-time problems only
Compiler listing	Always
JCL listing	MVS only
CMS terminal session log	CMS only
Error trace-backs	Always
Debugging output	Always
CMS EXECs	CMS only

Figure 16 (Part 1 of 2). Summary of Materials Required for APAR Submission

Materials	When Required
A description of the application program and the organization of its data sets	Always
Linkage editor or loader map listing	Link-time and run-time problems only
Applied PTFs and solutions	Always, or specify no solutions applied

Figure 16 (Part 2 of 2). Summary of Materials Required for APAR Submission

Note: If you supply machine-readable material on a tape reel, describe how the tape was created.

Original Source Information

You must supply source information in one of three forms:

- A small test case that reproduces the problem
- Your original source
- The machine-readable source (large programs only).

Note: If you do not supply one of these three, IBM Programming Service will probably return your APAR.

If you send machine-readable source, submit the information on an unlabeled tape. Along with the tape, send a hard copy listing that shows how the tape was created. Carefully pack and clearly identify machine-readable information. Make sure the APAR number is on the tape, so it can be identified if it is separated from the rest of the material submitted with the APAR.

Load Library Information

If the failure occurs at run time, any routines not contained in your program must be provided as either source or object code in machine-readable form.

Input Data Set Information

If the failure occurs at run time, you must provide enough input data with your APAR so the failure can be reproduced.

Compiler Listing

If you think you have a compiler failure, all listings that you supply must relate to a specific run of the compiler. Do not send information derived from separate compilations or runs. This might mislead the programming support personnel.

Always supply with your APAR these listings that result from the compilation of the original source:

- Source listing
- Intermediate code listings
- Object listing
- Cross-reference listing.

JCL Listing

In MVS, you must provide listings of job control statements used to run the program. If you are having problems with a batch job, show any cataloged procedures you are using in expanded form by specifying MSGLEVEL=(1,1) in the JOB statement.

CMS Terminal Session Log

If the failure occurs while compiling or running a program under CMS, supply the full details of the VM (virtual machine) environment:

1. Immediately before you invoke the compiler to reproduce the problem, issue the following commands:

   ```
   QUERY SET
   QUERY TERMINAL
   QUERY VIRTUAL
   QUERY SEARCH
   QUERY DISK *
   QUERY FILEDEF
   QUERY LIBRARY
   QUERY INPUT
   QUERY OUTPUT
   ```

2. Invoke the compiler using the VSPASCAL command, specifying the compile-time options required to produce the relevant output, preferably on a line printer, or, alternatively, at a typewriter terminal.

Submit the entire terminal listing, from LOGON to LOGOFF. If a display terminal is used, spool console input and output using the following commands:

```
CP SPOOL CONSOLE START
```

to start spooling and:

```
CP SPOOL CONSOLE CLOSE
```

to stop spooling just before you log off.

If running under FULLSCREEN CMS, you can enter:

```
PUT VSCREEN CMS fn ft
```

to get this information.

The output from these commands provides full details of all input entered and all responses received.

Error Trace-Backs

Always supply with your APAR a hard copy listing of error traces relating to each specific run of the compiler.

Debugging Output

Always supply with your APAR a hard copy listing of output from the debugging program relating to each specific run of the compiler.

CMS EXECs

Always supply with your APAR a hard copy listing of any CMS EXECs used.

Application Program Description

Always supply with your APAR a hard copy listing describing the application program, the data set organization, and the operating instructions or console log. These might be helpful in reproducing the error.

Linkage Editor or Loader Map Listing

Always supply with your APAR a hard copy listing of the linkage editors and loaders output at the time of the run.

Applied Solutions

Always supply with your APAR a list of any program temporary fixes (PTFs) and local solutions applied to either the compiler or to the library. If no solutions have been applied, specifically indicate this with your APAR.

Part 4. Reference

Chapter 8. Further VS Pascal Diagnostic Information

This section describes the major functions of the VS Pascal compiler and library. It is intended to give you a basis for communicating with an IBM program specialist about possible program failures, and is intended for no other purpose.

The Compiler

The VS Pascal compiler consists of four logical phases. The first compiler phase is the L phase, which transforms VS Pascal code into intermediate code. The second compiler phase is the O phase, which optimizes the intermediate code. The third compiler phase is the T phase, which transforms intermediate code into 370 machine code. Under MVS, there is an I phase that calls the L, O, and T phases.

U-code is the intermediate language that VS Pascal generates from the source code. VS Pascal uses U-code to communicate between each phase of the compiler.

The functions performed by each phase are:

Phase L

- Parses the VS Pascal source code
- Checks for syntax and semantic errors
- Translates the source code into U-code
- Prints the source listings
- Records and prints errors on source listings.

Phase O

- Optimizes U-code
- Processes Boolean expressions
- Performs range checking.

Phase T

- Performs common subexpression elimination
- Translates U-code into machine code
- Prints pseudoassembler listing
- Prints external symbol dictionary (ESD)
- Prints compiler statistics.

Phase I

- Invokes the VS Pascal compiler under MVS.

The Library

The VS Pascal library is a collection of subprograms that are combined, as needed, with object modules produced by the VS Pascal compiler to form an executable load module.

As the compiler examines Pascal source statements and translates them into an object module, it identifies the need for certain library operations. At the corresponding points in the object module, the compiler inserts calls to the appropriate library routines. Copies of these library routines can be link-edited and made part of the load module, or they can be linked dynamically at run time.

The VS Pascal library contains seven types of subprograms:

- General run-time routines
- Input/output routines
- Error-handling routines
- Conversion routines
- Mathematical routines
- String routines
- Storage management routines.

When you invoke a VS Pascal program, the routine responsible for establishing the VS Pascal run-time environment gains control and performs the following functions:

1. Obtains storage where dynamic storage areas (DSAs) are allocated and deallocated (the stack)
2. Creates and initializes the VS Pascal communication work area (PCWA)
3. Initializes the begin-clock function (MVS only)
4. Processes any run-time options
5. Sets up an environment to intercept program interrupts (fixed-point overflow, divide by zero, and so forth)
6. Calls the main program
7. Cancels program interrupt interception
8. Closes any open files upon return from the main program
9. Frees acquired storage
10. Calls the end-clock function (MVS only)
11. Returns control to the caller.

Figure 17 summarizes general run-time routines.

Procedure	Description
AMPXBCLK	Initializes the execution clock
AMPXCHKS	Checks a set for membership
AMPXCLCK	CLOCK function
AMPXCMS	Formats the user parameter string into CMS parameter list format and then calls AMPXSVC2 or AMPXSVC4
AMPXCPMT	Calls the Interactive Debugging Tool when a run-time error occurs

Figure 17 (Part 1 of 3). General Run-Time Routines

Procedure	Description
AMPXCRTE	Initializes the PCWA
AMPXDATE	DATETIME procedure
AMPXDATI	Gets the system date and time
AMPXDBCB	Obtains the DBCB pointer of a procedure
AMPXDLNK	Frees vector table
AMPXECLK	Ends the execution clock
AMPXGOTO	Handles a GOTO out of block
AMPXGTOK	Obtains a token from the user's execution parameters
AMPXG12	Returns the contents of register 12
AMPXG13	Returns the contents of register 13
AMPXHALT	HALT procedure
AMPXINIT	Processes the run-time options and user parameters
AMPXLTOK	LTOKEN procedure
AMPXLTVT	Loads the transient library vector table
AMPXMAIN	Interface for calling VS Pascal from other languages
AMPXMLNK	Allocates vector table
AMPXMOVE	Storage-to-storage move
AMPXMUS	Adds elements to a set
AMPXNAME	Obtains a procedure's name
AMPXOTOS	Converts an offset in a procedure to a statement number
AMPXPACK	PACK procedure
AMPXPAD	Fills storage with blanks
AMPXPARM	PARMS function
AMPXRETC	RETCODE procedure
AMPXSEGE	Intercepts calls to a SEGMENT name and issues an error
AMPXSETV	Fills storage with a value (SETMEM)
AMPXSPAR	Initializes for PARMS function
AMPXSVC2	Issues an SVC 202
AMPXSVC4	Issues an SVC 204
AMPXTERM	Prints execution counting summary, closes any open files, and frees acquired storage
AMPXTOK	TOKEN procedure

Figure 17 (Part 2 of 3). General Run-Time Routines

Procedure	Description
AMPXTRAC	TRACE procedure
AMPXUNPK	UNPACK procedure
AMPZABND	Abnormal termination routine
AMPZCMSC AMPZMVSC	Links with VS Pascal run-time routines that are to reside below the 16-megabyte line in an XA environment
AMPZCVD	Converts to decimal
AMPZMLD	Loads a message module
AMPZMSGC	Loads the address of TSO CLIST messages
AMPZMSGD	Loads the address of Interactive Debugging Tool messages
AMPZMSGE	Loads the address of CMS EXEC messages
AMPZMSGL	Loads the address of first compiler pass messages
AMPZMSGO	Loads the address of second compiler pass messages
AMPZMSGT	Loads the address of third compiler pass messages
AMPZMSGX	Loads the address of library messages
CLOCKH	Returns program execution time in 100ths of a second
CMS	CMS procedure
ITOHS	Integer-to-hexadecimal string conversion
PSCLHX	Terminates execution for interlanguage calls
VSPASCAL	Main entry point for a VS Pascal main program

Figure 17 (Part 3 of 3). General Run-Time Routines

Input/Output Routines

VS Pascal uses OS/VS access methods to implement its input/output facilities. VS Pascal file variables are associated with a data set by means of a ddname. Queued sequential access method (QSAM) is used for sequential data sets. The basic partitioned access method (BPAM) is used for partitioned data sets or MACLIBs (for VM/SP). The basic direct access method (BDAM) is used for random record access.

Each VS Pascal file is associated with a specific ddname. See the DDNAME open option in the *VS Pascal Application Programming Guide* for information on ddnames.

At run time, a data control block (DCB) is associated with every VS Pascal file variable. The DCB contains information describing specific attributes of the associated data set. For information on the DCB attributes, see "Data Set DCB Attributes" in the *VS Pascal Application Programming Guide.*

Figure 18 on page 53 summarizes the input/output routines.

Procedure	Description
AMPXCLOS	CLOSE procedure
AMPXCOLS	COLS function
AMPXDBCS	Reads DBCS data into variables
AMPXDRCT	Opens a file for random access
AMPXGET	GET procedure (text files)
AMPXGETR	GET procedure (record files)
AMPXLTIO	Closes all files in a routine or dynamic variable
AMPXOPEN	Opens files opened with RESET, REWRITE, TERMIN, TERMOUT, and UPDATE
AMPXOPN1	Initializes a PCB before opening file
AMPXOPN2	Sets a PCB after opening a file
AMPXPARS	Processes file open options
AMPXPCBC	Closes a file (PCB)
AMPXPDS	Opens files opened with PDSIN and PDSOUT
AMPXPUT	PUT procedure
AMPXRCHR	Reads a CHAR
AMPXRGCHR	Reads DBCS characters
AMPXRGSTR	Reads DBCS strings
AMPXRGTXT	Reads DBCS text
AMPXRINT	Reads an INTEGER
AMPXRLIN	Reads to end-of-line (text file)
AMPXRR	Reads a REAL value
AMPXRRDY	Prepares a text file for input
AMPXRREC	Reads one record (record file)
AMPXRSTR	Reads a STRING
AMPXRTXT	Reads an array of CHAR
AMPXSEEK	SEEK procedure
AMPXSTAT	Returns a code indicating whether a file opened successfully
AMPXTIO	Terminates I/O processing
AMPXWB	Writes a BOOLEAN value
AMPXWCHR	Moves data to an I/O output buffer
AMPXWCHS	Writes a CHAR to a text file

Figure 18 (Part 1 of 2). Input/Output Routines

Procedure	Description
AMPXWGCHR	Writes DBCS characters
AMPXWGSTR	Writes DBCS strings
AMPXWGTXT	Writes DBCS text
AMPXWINT	Writes an INTEGER to a text file
AMPXWLIN	Writes an end-of-line to a text file
AMPXWR	Writes a REAL value to a text file
AMPXWRDY	Prepares a text file for output
AMPXWREC	Writes one record (record file)
AMPXWSTG	Writes a string to a text file
AMPXWTXT	Writes an array of CHAR to a text file
AMPYCLOS	System-dependent QSAM close
AMPYDFLT	Applies system-dependent defaults to a file
AMPYDRCT	System-dependent BDAM open
AMPYGET	System-dependent GET procedure
AMPYOPEN	System-dependent QSAM open
AMPYPAGE	PAGE procedure
AMPYPDS	System-dependent partitioned data set interface
AMPYPUT	System-dependent PUT procedure
AMPYSEEK	System-dependent SEEK procedure
AMPZDAMR	Issues a READ for a BDAM data set
AMPZDAMW	Issues a WRITE for a BDAM data set
AMPZDCBC	Closes an OS/VS DCB
AMPZDCBO	Opens an OS/VS DCB
AMPZFIND	Issues an OS/VS BLDL
AMPZGET	Issues a QSAM GET
AMPZPUT	Issues a QSAM PUT
AMPZPUTX	Issues a QSAM PUTX
AMPZSAMR	Issues a READ for a BSAM data set
AMPZSAMW	Issues a WRITE for a BSAM data set
AMPZSTOW	Issues an OS/VS STOW
AMPZTGET	Issues a TGET (OS/VS) or RDTERM (CMS)
AMPZTPUT	Issues a TPUT (OS/VS) or WRTERM (CMS)

Figure 18 (Part 2 of 2). Input/Output Routines

Error-Handling Routines

VS Pascal detects many kinds of errors during program execution. Upon detection of an error, the VS Pascal run-time library provides error handling.

Certain errors are considered fatal by the run-time library. Examples of these errors are operation exceptions and protection exceptions. When a fatal error occurs, VS Pascal produces a message describing the error, displays a trace-back, and terminates the program.

Other errors such as checking errors do not stop program execution. You must determine the extent to which the nonfatal errors affect program results. When a nonfatal error occurs, VS Pascal:

1. Produces a message describing the error. The message is written to the file specified by ERRFILE or, when ERRFILE is not specified, to either the terminal in VM/CMS and MVS/TSO, or to the SYSPRINT data set in MVS batch.
2. Produces a symbolic variable dump of the variables in the procedure experiencing the error when the program was compiled and link-edited with the DEBUG option, and the program was not executed with the DEBUG run-time option. The message is written to the file specified by ERRFILE or, when ERRFILE is not specified, to either the terminal in VM/CMS and MVS/TSO, or to the SYSPRINT data set in MVS batch.
3. Invokes the Interactive Debugging Tool as if a breakpoint had been encountered if the program was compiled, link-edited, and executed with the DEBUG option.

VS Pascal allows a specific number of nonfatal errors to occur before the program is terminated. This number is set by the ERRCOUNT run-time option. The default is 20.

VS Pascal also provides a mechanism for you to gain control when a run-time error occurs. When such an error occurs, a procedure called ONERROR is called to perform any necessary action before generating a diagnostic message. A default ONERROR routine (which does nothing) is provided in the VS Pascal library.

You can write a version of ONERROR and declare it as an EXTERNAL procedure. The procedure is invoked when an error occurs. An example of this is shown in the *VS Pascal Application Programming Guide*. the string routines listed in

Figure 19 on page 56 summarizes the error-handling routines.

Procedure	Description
AMPXCHKR	Intercepts run-time checking errors
AMPXCOER	Calls the ONERROR procedure
AMPXDIAG	Intercepts program exceptions
AMPXERR	General run-time error handler
AMPXIOER	I/O error interception routine
ONERROR	Default ONERROR procedure

Figure 19. Error Handling-Routines

Conversion Routines

There are several places where VS Pascal must perform data conversions. These conversions take place when you are doing I/O on text files and when you use the READSTR and WRITESTR routines.

Figure 20 summarizes conversion routines.

Procedure	Description
AMPXATOS	Converts a DBCS fixed string to a STRING
AMPXBTOS	Converts a BOOLEAN to a string
AMPXCTOS	Converts a CHAR to a string
AMPXGTST	Converts a GSTRING to a STRING (short string)
AMPXHTOS	Converts a GSTRING to a STRING
AMPXITOS	Converts an INTEGER to a string
AMPXKTOS	Converts a GCHAR to a STRING
AMPXRTOS	Converts a REAL to a STRING
AMPXSTGS	Converts a STRING to a GSTRING (short string)
AMPXSTOA	Converts a STRING to a DBCS fixed string
AMPXSTOC	Converts a STRING to a CHAR
AMPXSTOG	Copies part of a STRING to another STRING
AMPXSTOH	Converts a STRING to a GSTRING
AMPXSTOI	Converts a STRING to an INTEGER
AMPXSTOK	Converts a STRING to a GCHAR
AMPXSTOR	Converts a STRING to a REAL
AMPXSTOS	Appends a STRING to another STRING

Figure 20 (Part 1 of 2). Conversion Routines

Procedure	Description
AMPXSTOT	Converts a STRING to an array of CHAR
AMPXSTRP	Checks a STRING for valid DBCS characteristics and removes adjacent shift-in/shift-out pairs and DBCS nulls
AMPXTTOR	Converts an array of CHAR to a REAL
AMPXTTOS	Appends an array of CHAR to a string
AMPX$GTS	Converts a GSTRING to a STRING (long string)
AMPX$STG	Converts a STRING to a GSTRING (long string)

Figure 20 (Part 2 of 2). Conversion Routines

Mathematical Routines

The mathematical routines are implemented as VS Pascal functions. The VS Pascal compiler changes the user-provided name (such as SIN) to an internal name (such as AMPXSIN).

Figure 21 summarizes the mathematical routines.

Procedure	Description
AMPXATAN	ARCTAN function
AMPXCOS	COS function
AMPXEXP	EXP function
AMPXLN	LN function
AMPXRAND	RANDOM function
AMPXSIN	SIN function
AMPXSQRT	SQRT function

Figure 21. Mathematical Routines

String Routines

The string routines are implemented as VS Pascal functions and procedures. The VS Pascal compiler changes the user-provided names (such as SUBSTR) to an internal name (such as AMPXSUBS). Several routines are provided in two forms: long and short. The short form is always used if possible. To use the short form, the VS Pascal compiler must determine that the resulting string is less than 2048 bytes long. If the size cannot be limited by compiler analysis, the compiler uses the long form that passes the results through the heap.

In the string routines listed in Figure 22, procedure names follow these conventions:

Convention	**Means**
AMPXxxxx	Short string routine
AMPX$xxx	Long string routine
AMPXMxxx	Short mixed string routine
AMPX$Mxx	Long mixed string routine
AMPXGxxx	Short DBCS string routine
AMPX$Gxx	Long DBCS string routine

Figure 22 summarizes the string routines.

Procedure	**Description**
AMPX$COM	COMPRESS function (long strings)
AMPX$DEL	DELETE function (long strings)
AMPX$GCP	COMPRESS function (long GSTRINGs)
AMPX$LTR	LTRIM procedure (long strings)
AMPX$MCP	MCOMPRESS function (long mixed strings)
AMPX$MDE	MDELETE function (long mixed strings)
AMPX$MLT	MLTRIM function (long mixed strings)
AMPX$MSU	MSUBSTR function (long mixed strings)
AMPX$MTR	MTRIM function (long mixed strings)
AMPX$SUB	SUBSTR function (long strings)
AMPX$TRI	TRIM function (long strings)
AMPXCAT	Concatenates 2 to 9 strings
AMPXCOMP	COMPRESS function (short strings)
AMPXDELE	DELETE function (short strings)
AMPXGCAT	Concatenates GSTRINGs
AMPXGCOM	COMPRESS function (short GSTRINGs)
AMPXGIDX	INDEX function (short GSTRINGs)
AMPXGLPD	LPAD procedure (GSTRINGs)
AMPXGRIX	DBCS RINDEX function (short GSTRINGs)
AMPXGRPD	RPAD procedure (GSTRINGs)
AMPXINDX	INDEX procedure (strings)
AMPXLPAD	LPAD procedure (STRINGs)

Figure 22 (Part 1 of 2). String Routines

Procedure	Description
AMPXLTRI	LTRIM procedure (short strings)
AMPXMCMP	MCOMPRESS function (short mixed strings)
AMPXMDEL	MDELETE function (short mixed strings)
AMPXMIDX	MINDEX function (mixed strings)
AMPXMLEN	MLENGTH function (mixed strings)
AMPXMLTR	MLTRIM function (short mixed strings)
AMPXMRIX	MRINDEX function (mixed strings)
AMPXMSUB	MSUBSTR function (short mixed strings)
AMPXMTRI	MTRIM function (short mixed strings)
AMPXRIDX	RINDEX function (strings)
AMPXSUBS	SUBSTR function (short strings)
AMPXTRIM	TRIM function (short strings)
LPAD	LPAD procedure (used only with %INCLUDE STRING)
PICTURE	PICTURE function
RPAD	RPAD procedure (used only with %INCLUDE STRING)

Figure 22 (Part 2 of 2). String Routines

Storage Management Routines

All VS Pascal dynamic variables are allocated from pools of storage called heaps. The NEW function generates a call to the internal procedure AMPXNEW (or AMPXVNEW for pointers to variant records). This procedure allocates storage within a heap. If a heap has not yet been created, NEWHEAP obtains storage from the operating system to create a heap.

The DISPOSE procedure generates a call to the procedure AMPXDISP. This procedure deallocates the heap storage acquired by a previous call to AMPXNEW. DISPOSEHEAP frees a heap created by NEWHEAP.

The MARK procedure generates a call to the procedure AMPXMARK. This procedure creates a new subheap from which subsequent calls to AMPXNEW obtain storage.

The RELEASE procedure generates a call to the procedure AMPXRLSE. This procedure frees a subheap that was previously created via the AMPXMARK procedure. Subsequent calls to AMPXNEW obtain storage from the heap or subheap which was active before the call to AMPXMARK.

QUERYHEAP returns a pointer to the current heap.

USEHEAP changes the current heap.

Data required by the run-time environment is allocated in a separate heap controlled by AMPXINEW and AMPXIDSP. Thus the I/O control blocks and debugging tables are in a distinct area.

Figure 23 summarizes the storage management routines.

Procedure	**Description**
AMPXALOC	Basic storage allocator
AMPXDISP	DISPOSE procedure
AMPXDSPH	DISPOSEHEAP procedure
AMPXFREE	Basic storage deallocator
AMPXIDSP	Free a dynamic variable in the special VS Pascal run-time heap
AMPXINEW	Create a dynamic variable in the special VS Pascal run-time heap
AMPXMARK	MARK procedure
AMPXNEW	NEW procedure
AMPXNEWH	NEWHEAP procedure
AMPXQUEH	QUERYHEAP procedure
AMPXRLSE	RELEASE procedure
AMPXTMEM	Terminates processing for storage management
AMPXUSEH	USEHEAP procedure
AMPXVNEW	NEW procedure (when record is allocated with tags)

Figure 23. Storage Management Routines

The Debugging Library

The Interactive Debugging Tool is activated using the DEBUG compile-time option. This option sets up the debugging tables and includes the necessary Interactive Debugging Tool routines in the unit being compiled. When the load module is executed, the DEBUG run-time option activates the debugging environment.

Breakpoint Handling

The address of the statement at which a breakpoint is to be set is calculated and stored in a table. The first two bytes of the code at this address are stored in a table, and then are overwritten with an illegal opcode. A branch back to the program is also stored in the table so that execution can resume at the correct location after the breakpoint is hit. When an illegal operation occurs, the table is searched to determine if the error is a breakpoint or an actual program error. If it is a breakpoint, the debugger command prompt is issued.

When execution resumes, the original first two bytes are used to rebuild and execute the statement. The illegal opcode in the code is not replaced with the original first two bytes until the breakpoint is removed.

Appendix A. Summary of Changes

VS Pascal Release 2 provides additions and enhancements to VS Pascal Release 1 in the following areas:

- **Communication with Other Programming Languages**

  VS Pascal now:

  - *Provides better error detection in Assembler routines called by VS Pascal.* Program checks in Assembler routines coded with the PROLOG macro will be handled using the ONERROR procedure used by VS Pascal instead of causing the severe error message AMPX902S. In order for program checks to be handled by ONERROR, Assembler routines using the PROLOG macro must be reassembled.

- **Transient Run Time**

  Users now have the option to:

  - *Create modules that are self-contained, or that can dynamically access run-time routines.* The transient run-time library will help free up resources in large-scale, modular systems that must serve multiple users. Transient run time reduces the size of load modules and makes it unnecessary for each user to maintain a copy of the run-time library.

- **Compiler Features**

  Users now have the option to:

  - *Compile only selected portions of a source program*. This "conditional compilation" feature can simplify debugging and help support multiple operating environments.
  - *Place headers in generated code*. Headers include the name of the compiled routine, the compiler name, and the date and time of compilation. Users can also insert a customized header after the compiler header.

- **Compile-Time Limits**

  Users can now write and debug larger and more complex programs. Each compilable program can have up to:

  - *999 %INCLUDE directives* (previous limit: 255)
  - *8192 TYPE declarations* (previous limit: 255)
  - *32678 chararcters in identifier names in a routine* (previous limit: 8192)
  - *1024 fields per record* (previous limit: 255)

- **Debugging**

  Users can now:

  - *Specify how many instances of a breakpoint can occur before program execution halts.* Previously, execution halted at every occurrence of a breakpoint. Programmers now have the flexibility to bypass a specified number of executions of a repeated statement.
  - *Display the statistics kept by the* COUNT *run-time option at any time during a debugging session.*

- **National Language Support**

  Release 2 also:

  - *Allows customization of character translation and uppercase tables at installation*. This eases compiler recognotion of tokens and characters due to different national programming standards, and allows creation of uppercase rules.
  - *Provides three languages from which sites choose a default language during installation*. Both at run time and compile time, users can override the default language with another language. Currently, VS Pascal provides mixed-case English, uppercase English, and Japanese.

Bibliography

VS Pascal Publications

These books provide additional information about VS Pascal.

Evaluation

- *VS Pascal General Information*, GC26-4318, provides an overview of VS Pascal.
- *VS Pascal Licensed Program Specifications*, GC26-4317, contains warranty information for VS Pascal.

Application Programming

- *VS Pascal Application Programming Guide*, SC26-4319, explains how to compile, execute, and debug VS Pascal programs.
- *VS Pascal Language Reference*, SC26-4320, provides a detailed explanation of the VS Pascal programming language and its syntax.
- *VS Pascal Reference Summary*, SX26-3760, provides quick-reference charts of VS Pascal language rules and processing/debugging options.

Installation

- *VS Pascal Installation and Customization for VM*, SC26-4342, explains how to install VS Pascal under VM/SP and VM/XA.
- *VS Pascal Installation and Customization for MVS*, SC26-4321, explains how to install VS Pascal under MVS/SP, MVS/XA and MVS/ESA.

Related Publications

- *OS/VS Message Library: VS2 System Messages*, GC38-1002
- *MVS/Extended Architecture Message Library: System Messages Volume 1*, GC28-1376
- *MVS/Extended Architecture Message Library: System Messages Volume 2*, GC28-1377
- *MVS/Enterprise System Architecture Message Library: System Messages Volume 1*, GC28-1812
- *MVS/Enterprise System Architecture Message Library: System Messages Volume 2*, GC28-1813
- *Virtual Machine/System Product System Messages and Codes*, SC19-6204
- *VM/XA System Product Systems Messages and Codes Reference*, SC23-0376.
- *Field Engineering Programming Systems General Information*, G229-2228

Index

Reader's Comment Form

VS Pascal
Diagnosis Guide and Reference

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: ____________________

Comments (please include specific chapter and page references) :

Note: Staples can cause problems with automatic 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 ____________________ Date ____________________

Company ____________________ Phone No. (______) ____________________

Address ____________________

____

Thank you for your cooperation. No postage is 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 them directly to the address in the Edition Notice on the back of the title page.)

LY27-9525-1

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 MAIL PERMIT NO. 40 ARMONK, NY

POSTAGE WILL BE PAID BY ADDRESSEE

IBM Corporation
Programming Publishing
P.O. Box 49023
San Jose, CA 95161-9023

Fold and tape **Please do not staple** **Fold and tape**

IBM®

IBM®

Program Number
5668-767
5668-717

File Number
S370-37

The VS Pascal Library

Application Programming Guide	SC26-4319
Diagnosis Guide and Reference	LY27-9525
General Information	GC26-4318
Installation and Customization for MVS	SC26-4321
Installation and Customization for VM	SC26-4342
Language Reference	SC26-4230

Supplementary Publications

Licensed Program Specifications	CG264317
Reference Summary	SX26-3760

LY27-9525-1