Pascal-2/RSX-11 Version 2.1E for PDP-11

Installation Guide/Release Notes

October 31, 1986

This document describes the Pascal-2 Software Development System for PDP-11 computers with RSX-11.

SUPERSESSION INFORMATION:	Supersedes V2.1D.
OPERATING SYSTEM AND VERSION:	RSX-11, RSX-11M-PLUS, mRSX-11M-PLUS, P/OS.
ADDITIONAL SYSTEM REQUIREMENTS:	NOANSILIB for Micro-RSX

Oregon
Software

6915 S.W. Macadam Avenue Portland, Oregon 97219-9987

Name of Contractor and Address:

Oregon Software, Inc.
6915 S.W. Macadam Avenue
Portland, Oregon 97219
Phone: 503-245-2202

Introduction

These notes describe V2.1E of the Pascal–2 Software Development System hosted on PDP-11/RSX-11 computers. These notes are organized into four sections.

- The "Introduction" describes system requirements and provides a list of the files to be found on the distribution media.
- The "Installation" section provides step-by-step instructions for installing the compiler, support library, and utilities from the distribution media.
- The "Release Notes" provide information on the software and documentation that is specific to this release.
- The "Pascal Certification Report" provides a brief explanation of the British Standards Institution's certification program and a copy of the test results for this product.

The entire Installation section should be read before attempting to install the compiler and utilities on your system.

This document follows these style conventions:

Text: Pascal reserved words, predefined symbols, operating system commands, options and directories and user-program names are in boldface typewriter type: **forward**, **read**, **link**, **test**. Names of utility programs or system names are in upper case: PASMAT, MACRO.

Examples: All commands assume a carriage return at the end. Commands that you should enter are in underlined boldface typewriter: pas2/list.

Program Listings: The Pascal–2 compiler accepts any combination of upper-case and lower-case characters. Examples in this manual have Pascal words in lower case and have user-defined words with capitalization as needed for readability, as shown in this program fragment:

```
procedure Show;
begin
  SomeUserAction;
  writeln(Result);
end;
```

Terminology: We use standard terms as they are used in documents describing the RSX-11 operating system.

System Requirements

To execute programs compiled by Pascal–2, the PDP-11 computer must be running one of the following operating systems: RSX-11M, RSX-11M-PLUS (see "Release Notes" for specific limitations), IAS, VAX/VMS (RSX-11M Compatibility Mode), RSX emulator on RSTS/E.

On a MICRO/PDP-11 under Micro-RSX, you must use Version 1 with the Advanced Programmer's Kit, which supplies a replacement system library (NOANSLIB) containing FCS object modules required to support Pascal tasks. The compiler cannot be installed or run without this library.

Distribution Files

The distribution media contains the following files.

Contents of the Release Media

Component	File Name	Description of Contents
Compiler	PASA.OLB	Compiler library, part A
	PASB.OLB	Compiler library, part B
Command Files	PASCAL.CMD	TKB commands for building the compiler
	PASBLD.CMD	Commands for installing the compiler
	UTLBLD.CMD	Commands for installing the utilities
	PASRES.CMD	Commands for creating Pascal resident libraries
Object Libraries	PASFPP.OLB	FPP support library
	PASFIS.OLB	FIS support library
	PASEIS.OLB	EIS support library
	PASSIM.OLB	SIM support library
Command String Interpreters (CSI)	CSICON.PAS	CSI constant definitions
	CSITYP.PAS	CSI type definitions
	CSIPRO.PAS	CSI procedure definitons
	FIXARG.PAS	CSI argument parser
	FIXINC.PAS	CSI %include file handler
	FIXOUT.PAS	CSI temporary output manager
	GETCS.PAS	CSI command parser
	CNVNUM.PAS	CSI command-line number converter
Utility Programs	PROSE.PAS	Document formatter
	PASMAC.MAC	MACRO-11 interface package
	XREF.PAS	Concordance generator for Pascal programs
	PROCRE.PAS	Procedural cross-referencer
	STRING.PAS	Dynamic string package
	PASMAT.PAS	Pascal source text formatter
	LIBDEF.PAS	Library definitions
	OPERRO.PAS	Support library error routine
	UERROR.PAS	User error routine
	VERIFY.PAS	Installation verifier
	SAYERR.PAS	System error routine
	CONVRS.PAS	Conversion of Pascal–1 to Pascal–2
Overlay Descriptions	PASCAL.ODL	Overlay directives for the compiler
	PAS.ODL	Overlay directives for Pascal programs

Installation

This guide describes the way to load the RSX Pascal–2 software on your system. The procedure for building the Pascal software is similar for all RSX-based systems, but you may have to perform some steps manually on systems that do not emulate RSX indirect command files.

This guide contains a sample use of the PASBLD.CMD installation command file and the text of UTLBLD.CMD, which builds the Pascal–2 utilities.

Installation Process

Follow these steps to build the Pascal–2 system:

1. Load all modules in the distribution kit onto your system disk.
2. Task-build the Pascal–2 compiler.
3. Configure the support library to match the hardware.
4. Build the utilities (optional).

The four steps are explained in detail below. (Steps 2 and 3 are performed automatically by the PASBLD.CMD installation command file.)

Copying Modules to Disk

We recommend that you set aside a special User File Directory (UFD) to collect all the Pascal–2 modules and utilities in one place. Then log in to that account and load all of the Pascal–2 modules onto the disk. VAX users should copy the distribution kit into its own directory, for example, `[PASCAL2.V21E]`.

Because Pascal–2 runs on all of the systems previously described, we chose a tape and disk format that was compatible among all of the systems. The tapes are in DOS format, and the disks are in RT-11 format. You must use the `FLX` conversion utility to load the files onto your system. Note that on RSX-11M PLUS, DOS and RT-11 volumes must be mounted with foreign characteristics. Sample `FLX` commands are given below for the most common media on which Pascal–2 is available:

Command Line	Medium	Device
`FLX>SY:/RS=MT0:[1,1]*.*/DO`	Magnetic Tape	TS04
`FLX>SY:/RS=DX0:*.*/RT`	Floppy Disks	RX01
`FLX>SY:/RS=DK0:*.*/RT`	Disk Pack	RK05

Enter `Control-Z` to exit any RSX utility, such as `FLX` or `PIP`.

When the distribution media are floppy disks, you must load the contents of each disk onto the system. About 3000 blocks are read in. Enough free space must be left over to task-build the Pascal–2 compiler. The task requires about 450 contiguous blocks. Much of this space can be reclaimed after the build process finishes. On multidisk systems (such as RL01-based systems) you may want to load the Pascal–2 system on an empty disk, which can then serve as a backup disk.

Note to VAX Users

The PASBLD.CMD command file, described in the next section, contains MCR-style commands, which cannot be executed from DCL. Before executing PASBLD indirectly, you must enable the MCR at login so the PASBLD commands can be properly executed.

Assuming you have just loaded the distribution kit onto your disk, log out and log back in using the /CLI and /NOCOMMAND switches along with your user name. The /CLI=MCR switch specifies the MCR as your command line interpreter, enabling the processing of MCR command files (such as PASBLD.CMD). The /NOCOMMAND switch prevents the automatic execution of your LOGIN.COM file, which probably contains DCL commands not recognized by the MCR.

For example:

```
Username: STEVE/NOCOMMAND/CLI=MCR
Password: ————————————————————— enter password normally
```

The MCR then prompts for a command. Set your default directory to the Pascal–2 release directory.

```
>SET DEFAULT [PASCAL2.V21E]
```

Now you can execute PASBLD, as shown below.

Building the Compiler and Support Library

After loading the Pascal–2 modules, you can use the PASBLD.CMD command file, supplied with the release, to select only those modules that match the hardware on your particular PDP-11 and to configure the compiler and support library for your system. Execute PASBLD as an indirect command file, as shown:

```
>@PASBLD
```

This command file guides you through the rest of the build process. On systems that do not fully emulate RSX command files, you must examine the PASBLD.CMD file and duplicate the actions it performs. (See Appendix B.)

The PASBLD command file installs the compiler in three steps:

- It configures the compiler and support library.
- If you wish, it deletes Pascal–1 modules from the support library (Pascal–1 and Pascal–2 use the same library—see "Installing Pascal–2 With Pascal–1").
- It purges any unnecessary files left over from the installation.

At each step PASBLD asks you a question or a series of questions, to which you respond with either 'Y' for yes or 'N' for no.

The first question PASBLD.CMD asks you is "Use automatic configuration for compiler and library?" Answer "yes" to this question and PASBLD automatically configures the support library for your hardware.

However, if you answer "no" to this question or if the automatic configuration fails, PASBLD then asks you a series of questions to identify the hardware on which you wish to run Pascal–2 programs.

First, you are asked whether your machine has FPP floating-point instructions. These floating-point instructions, available on many machines, are often implemented with a floating-point processor hardware option.

FPP instructions include the **ADDF**, **DIVF**, and **MODF** instructions. If your machine supports such instructions, you should answer "yes" to the FPP question; otherwise, answer "no."

Your processor may instead support the Floating Instruction Set (FIS). The FIS instructions are stack-oriented instructions providing these four basic functions: **FADD**, **FSUB**, **FMUL**, and **FDIV**. If your processor supports FIS instructions, you should answer "yes" to the FIS question; otherwise, answer "no."

If you answered "no" to the first two questions, you are then asked if your processor supports the Extended Instruction Set (EIS). Most PDP-11s now support EIS instructions, which include **MUL**, **DIV**, **XOR**, and **SOB**. Note that the Pascal–2 compiler requires EIS to run (the FPP and FIS options support EIS). If your processor supports EIS, then answer "yes" to this question; otherwise, answer "no."

Answering "no" to the first three questions causes a software floating-point simulator to be included in the Pascal–2 support library. This simulator permits you to write programs that use **real** numbers.

The answers to the above questions determine the module your hardware requires for the Pascal support library. The modules of choice are as follows:

Module	Description
PASFPP.OLB	FPP library modules
PASFIS.OLB	FIS library modules
PASEIS.OLB	EIS library modules
PASSIM.OLB	SIM library modules

Depending on the processor option for your system, the PASBLD command file renames one of these modules to PASLIB.OLB and deletes the libraries that are not used.

After the questions have been asked but before the libraries are actually renamed, PASBLD.CMD uses the Task Builder to build the Pascal–2 compiler task image. The compiler is supplied as two object module libraries, PASA.OLB and PASB.OLB. The Task Builder command file, PASCAL.CMD, is used in this step, along with PASCAL.ODL (because the compiler is overlaid).

The task-build procedure also requires the library PASEIS.OLB to build the Pascal–2 compiler because the compiler itself is a Pascal program.

After building the compiler and the libraries, the PASBLD command file asks whether you wish to delete the Pascal–1 modules from the support library. Doing so reduces the size of PASLIB.OLB. PASBLD.CMD then asks a series of questions relating to the copying and deleting of new and old compiler and library files. These questions assist in post-installation disk cleanup.

When PASBLD completes, the following modules are created:

PAS.TSK — This is the Pascal–2 compiler. It should be copied to your system account so that others may have access to it. If you use the INS command to install the compiler, you are able to invoke the compiler with the PAS command. You may also run the compiler by typing RUN PAS. VAX users should copy PAS.EXE to SYS$SYSTEM:. The command to install the compiler from your system account using INS is simply:

```
>INS $PAS
```

PASLIB.OLB — This is the Pascal support library. You should copy this file into [1,1] on your library device. This library contains modules required by all compiled Pascal–2 programs, including modules for the Pascal–2 Debugger and Profiler. The command you may use is:

```
>PIP LB:[1,1]=PASLIB.OLB
```

VAX users should log out and log back in as a DCL user before copying the previously mentioned modules to the system accounts.

After copying the above modules to the system accounts, you are able to compile, debug, profile and run Pascal–2 programs. After also copying any of the utilities, you may remove the disk on which you loaded the distribution kit.

Building the Utility Programs

The Pascal–2 distribution kit contains other utility and demonstration programs. After building the compiler and configuring the libraries, you can also build the Pascal–2 utility programs with the UTLBLD.CMD command file. Simply type the command:

```
>@UTLBLD
```

On systems that do not emulate RSX indirect command files, you must examine UTLBLD.CMD and execute the commands manually. (See Appendix C.) Also note that task-building commands in UTLBLD.CMD invoke the /FP switch. On systems that do not have floating-point hardware, you must edit UTLBLD.CMD so that you task build the utilities without the floating point option.

The utilities that UTLBLD.CMD builds are:

PASMAT: — Reformats Pascal source code to improve readability. Does syntax checking.

PROCRE: — Generates a cross-reference listing of procedure names.

PROSE: — Formats documentation and text files.

XREF: — Generates a cross-reference listing of your program. Also gives a concordance of any text file.

These utilities are described in the Utilities Guide of the *Pascal–2 User Manual*.

Verifying the Installation: The VERIFY Utility

After you have task-built the compiler and configured the support library, you should verify that you have installed the compiler correctly. The program VERIFY.PAS, supplied in the distribution kit, verifies the installation for you. Although it is designed specifically to verify the installation process, this program can be executed any time after the compiler is installed, to learn about your system's configuration.

You may decide to run the verification program for the following reasons:

- To determine the cause of obscure problems occurring when you try to use the compiler.
- To see whether your RSX system is SYSGENed correctly.
- To determine the hardware configuration for your particular system.
- To see whether you've incorrectly added floating-point hardware after configuring the RSX monitor.
- To see whether the Pascal support library is the correct version.
- To receive a demonstration of Pascal–2's error reporting capabilities. The demonstration intentionally causes an I/O error and shows the diagnostics Pascal–2 prints. For details, see "Typical Verification Session" below and the section on "Run-Time Error Reporting" in the Programmer's Guide of the *Pascal–2 User Manual*.

Running VERIFY

The UTLBLD.CMD command file automatically compiles and builds the VERIFY program. To run the program, enter the command:

```
>RUN VERIFY
```

If your system crashes at this point, you have incorrectly SYSGENed RSX for your hardware. You'll have to re-SYSGEN your system correctly before running VERIFY again. Chances are, you won't have this problem. Only a few users have reported such system crashes, which only occur when all of the following conditions exist:

1. No floating-point hardware is present;
2. During Phase I of the SYSGEN of your RSX system you answered "yes" to the question: "Floating point processor (FP-11) present?" (the correct response here is "no");
3. A program is task-built with the **/FP** switch. The **/FP** switch tells the RSX monitor to save the status of the FPP hardware when the task checkpoints.

The system crashes after the program is executed because the poorly configured RSX monitor is trying to save the contents of the nonexistent floating-point registers. The RSX monitor cannot save the registers and it traps while trying to execute a reserved instruction. If in doubt, configure Pascal to use EIS instructions.

In this situation, any task (Pascal or otherwise) that you build using the **/FP** switch may crash the system. You must re-SYSGEN your system correctly, without FPP hardware. The **AUTOCONFIGURE** option in the Version 4.0 RSX SYSGEN helps prevent this kind of problem.

The run-time error message "PASCAL — Fatal initialization error" usually means that you have configured the compiler and support library incorrectly. Reinstall the compiler with the correct hardware options for your system.

The run-time error "Not enough memory. Try making task checkpointable or expand $$HEAP" has three possible causes:

- You did not task-build VERIFY with the /CP switch, as shown above.
- Checkpointing is not enabled. Use the ACS command to allocate checkpoint space.
- The maximum task extension limit is set too low. Have the system manager reset the limit to 64K bytes using the SET/MAXEXT command because Pascal-2 tasks must be able to expand up to 64K bytes.

The run-time error "No FPP support. Re-Task Build with /FP" has two possible causes:

- You incorrectly SYSGENed your RSX system. This could occur if your processor has FPP hardware but you answered "no" to the Phase-I question "Floating point processor (FP-11) present?" A "no" answer prevents RSX from using the floating-point hardware.

Typical Verification Session

The results of the verification program are usually similar to this sample

```
>RUN VERIFY

Pascal installation verification program V2.1

Congratulations!  You have correctly configured your Pascal system.


Today is Thursday, 25-Aug-1983  12:44:25 PM

This is an RSX-11M operating system

The name of this task is "TT2"
It is running in the "GEN" partition
Your UIC is [2,32]
Your default disk is: DL0:

Error reporting demonstration

This program now intentionally causes an I/O error
to show the full error reporting facilities.

The error should be "reading past end of file".
A complete dump of the file structures will follow.
A dump of how memory is allocated will be printed, and
an error walkback identifies where the error happened.

TT2 -- I/O error at user PC= 7736
Attempt to read past end of file
I/O error code= -10. (366B) in file: DB0:TEST.TMP;0
End of file detected-------------------------- text of I/O error message

File information for file variable at: 126472B---------------- beginning of file dump
    :
    :
Memory map:------------------------------------ beginning of memory dump
    :
    :
Task size=47040. bytes

Error occurred at line 862 in procedure cause_error
Last called from line 880 in procedure error_demo
Last called from line 896 in program verify
```

A Sample Installation

(This log file shows a sample use of the PASBLD.CMD command file in which the automatic configuration option is selected and the compiler and support library are installed and old versions are deleted.)

```
>@PASBLD
>;
>; Pascal-2 V2.1D Configuration procedure
>;
>; Consult the "Installation Guide" in the Release Notes
>;
>;
>; This command file can automatically configure the Pascal compiler
>; and support library to make the most efficient use of the hardware
>; available on your processor.
>;
>* Use automatic configuration for compiler and library? [Y/N]: Y
>TKB CONFIG/CP=PASEIS/LB:CONFIG,PASEIS/LB
>RUN CONFIG
Your system supports FPP floating point instructions.

>PIP CMPLIB.OLB/RE/NV=CMPFPP.OLB
>TKB @PASCAL
>PIP PASLIB.OLB=PASFPP.OLB/RE/NV
>LBR PROFIL=PASLIB/EX:PROFL1
>;
>; The Pascal-2 support library PASLIB.OLB contains modules to support both the
>; Pascal-1 and Pascal-2 compilers.  The modules used only by Pascal-1 can be
>; removed, and the size of PASLIB.OLB can be reduced.
>;
>* Should the Pascal-1 modules be deleted from PASLIB.OLB? [Y/N]: Y
>PIP PASLIB.OLB;0/LI

Directory DL0:[2,33]
2-NOV-83  15:22

PASLIB.OLB;4          350.     C  02-NOV-83 15:16

Total of 350./350. blocks in 1. file
```

```
>LBR PASLIB/DE:DBG:DBG1:DBG2:DPOINT:$DEBUG:PROFL1:$REG1
Modules deleted:
DBG
 :
$REG1
>LBR PASLIB/DE:D1:D2:D3:D4:D5:D6:D7:D8:D9:D10
Modules deleted:
D1
 :
D10
>LBR PASLIB.OLB/CO=PASLIB.OLB
>PIP PASLIB.OLB/TR
>PIP PASLIB.OLB;0/LI

Directory DLO:[2,33]
2-NOV-83  15:27

PASLIB.OLB;5        246.    C  02-NOV-83 15:26

Total of 246./246. blocks in 1. file

>;
>;
>; The Pascal support library is usually located in LB:[1,1]PASLIB.OLB.
>; This is not a requirement, but if you wish to use overlays you will
>; need to edit PAS.ODL if PASLIB.OLB will not reside in LB:[1,1].  The
>; Pascal compiler PAS.TSK is usually located in LB:[1,54].
>;
>* Should PAS.TSK be copied to LB:[1,54]? [Y/N]: Y
>PIP LB:[1,54]/NV=PAS.TSK
>;
>; The new compiler will be installed
>;
>REM PAS
>INS $PAS
>* Should previous versions of lb:[1,54]PAS.TSK be deleted? [Y/N]: Y
>PIP LB:[1,54]PAS.TSK/PU
>* Should PASLIB.OLB be copied to LB:[1,1]? [Y/N]: Y
>PIP LB:[1,1]/NV=PASLIB.OLB
>* Should the previous versions of LB:[1,1]PASLIB.OLB be deleted? [Y/N]: Y
>PIP LB:[1,1]PASLIB.OLB/PU
>* Should the overlay description PAS.ODL be copied to LB:[1,1]? [Y/N]: Y
>PIP LB:[1,1]/NV=PAS.ODL
>* Should the previous versions of LB:[1,1]PAS.ODL be deleted? [Y/N]: Y
>PIP LB:[1,1]PAS.ODL/PU
```

```
>;
>; Configuration is now complete.
>;
>; Many of the files used during the configuration of your compiler will not
>; be needed now that your compiler has been built.
>;
>* Should the unnecessary files be deleted? [Y/N]: Y
>PIP PASCAL.CMD;*/DE/NM,PASCAL.ODL;*,PASA.OLB;*,PASB.OLB;*
>PIP PAS.TSK;*/DE/NM
>PIP PASLIB.OLB;*/DE/NM
>PIP PASLIB.OLB/PU/NM
>PIP PAS.ODL;*/DE/NM
>PIP PASFIS.OLB;*/DE/NM
>PASEIS.OLB;*/DE/NM
>PIP PASSIM.OLB;*/DE/NM
>PIP PASBLD.CMD;*/DE/NM,CONFIG.CMD;*,CONFIG.TSK;*,CONFIG.EXE;*
>@ <EOF>
```

Utilities Installation Control File (UTLBLD.CMD)

```
;
; build Pascal-2 utilities
;
pas verify
pas pasmat
pas prose
pas procre
pas xref
;
tkb verify/cp/fp=verify,lb:[1,1]paslib/lb
tkb pasmat/cp/fp=pasmat,lb:[1,1]paslib/lb
tkb prose/cp/fp=prose,lb:[1,1]paslib/lb
tkb procre/cp/fp=procre,lb:[1,1]paslib/lb
tkb xref/cp/fp=xref,lb:[1,1]paslib/lb
;
; Pascal-2 utility construction complete.
;
```

Release Notes

The information contained in this document describes the Pascal–2 Version 2.1E release package. In these notes you will find:

- A summary of problems ("bugs") we've fixed in this release.
- Notes on changes in the documentation and on how to update the user manual.
- Miscellaneous notes of interest to Pascal–2 users.

Version 2.1E is primarily a maintenance up-date: most of the changes from Version 2.1D to 2.1E correct reported problems. In particular, two major problems have been corrected:

Packing Problems. The bugs in Pascal–2's packing of records and arrays have been corrected without any modification of the packing algorithm. Users who have programs that involve the reading and writing of packed records or arrays may continue to used their data files under 2.1E.

'BREAK' Function. The `break` procedure is now fully operational. A new library module, called OPBRK.MAC, replaces the name OPIO.MAC in the listing of library source files (page 2-102) as the source of p$62.

Problems We've Fixed

Version 2.1E corrects a number of bugs. Two particulary troublesome ones are:

- Complex or numerous definitions in a `%include` file no longer cause the compiler to generate a spurious error message "Actual parameter type doesn't match formal parameter type" in the listing file.
- Variables declared with a fixed origin of `177777B` no longer are reported as a syntax errors.

Known Problems

At the time of this release, Version 2.12.1E has the following limitations.

DCL Command-Line Interpreter

A problem may occur due to the interaction of Pascal–2 and the DCL command-line interpreter. The symptoms are: when a command containing a compiler switch is issued to the installed Pascal-2 task, this command returns an error message complaining about `Two filenames in one field....` For example, neither of the following commands would succeed because the compiler switch `nocheck` is used.

```
$PAS PROGNAM/NOCHECK
:

$PAS
file? PROGNAME/NOCHECK
```

To work around this problem, you may use the following command:

```
$RUN $PAS
PAS> PROGNAME/NOCHECK
```

Another solution is to change the name of the installed Pascal-2 task. Since the problem is caused by the presence in the DCL tables of a predefined command `PAS`, changing the task name to something else eliminates the conflict with DCL. The new task name should be three characters which do not conflict with any other DCL commands. The name change can be accomplished using the INSTAL utility, as in:

```
$INSTAL $PAS/TASK=...PC2
```

Note that this INSTAL command is performed during compiler installation.

When the compiler is invoked by another task name, such as `PC2`, no adverse interaction with DCL occurs, as unrecognized Pascal–2 commands are passed to the MCR command line interpreter. **Note:** Examples in the manual still show `PAS` as the invocation for the compiler.

Documentation Notes

The documentation for Pascal–2 Version 2.1E includes a copy of the third edition of the *Pascal–2 User Manual*.

Miscellaneous Notes

Version 2.1E requires more memory than the 2.1D version. The change which uses the memory allows certification of the compiler at ISO Level 1. If you find that compilations of large programs fail with the "Out of memory" error message, you should break the modules involved into smaller sized units and recompile them.

Note the following, additional items:

- The `ord` function returns a range of `0..255` for type `char`.
- Users may experience problems with restoring registers after calls to `non-pascal` subroutines, e.g. FORTRAN. When calling a FORTRAN procedure via a Pascal–2 `procedureI` from a Pascal–2 `procedureY`, a real variable which is local to `procedureY` gets corrupted. The variable is held in a floating-point accumulator rather than in the stack. This problem will be resolved in a future release.

Please be aware of the following documented limitations of the software:

1. PASMAC does not support structured functions.
2. The Pascal-2 Debugger cannot be used when images are task-built to use I/D space separation.
3. A call to the `eof` function may alter the results of the `Getpos` procedure.
4. Structured constants containing strings create duplicates of each string. This may cause problems for a system with limited memory or for large tasks that approach the 64K byte limit. This problem can be avoided by dropping down one level of structuring and defining the individual characters of the array as separate elements, not as strings.

Pascal Certification Report

The British Standards Institution has certified this version of the Pascal-2 compiler for compliance with ISO standard 7185. During validation process, Oregon Software submits the software to an automated suite of 740 tests. The test program automatically generates a report on the software's performance, which must be distributed with the software. The report contains four notable sections:

- Processor Identification — indicates the exact combination tested: the machine, the version and release level of the compiler, the operating system, and the conditions under which the tests were made.
- Statement of Compliance — states the level of the standard, any exceptions to compliance, non-standard extensions, and any implementation-specific features.
- Details of Tests — provides data on the type of tests performed and the results of each. This section contains tests of: conformance and deviance, error-handling, implementation-defined features, implementation-dependent features, quality, and extensions.
- Statistical Summary — gives the number of tests passed, failed, or withdrawn; the number of errors detected or not detected.

Pascal certification is relatively new. In fact, Oregon Software is the first company in the United States to successfully complete the process and has passed the largest number of compilers at one time—eleven at the first attempt. Assuming that our users are not familiar with validation reports, we've provided this summary of the report's contents. You may find a more complete explanation of the test criteria, with comments on how to interpret the data, in: *A Readers' Guide to Pascal Compiler Validation Reports* by Z. J. Ciechanowicz and R. A. Wichmann, NPL Report DITC 24/83, © 1983.

The report uses the following terms as they are defined in the standard:

processor the complete Pascal implementation needed to execute a Pascal program, including a compiler, run-time library, operating system, and underlying hardware.

implementation-defined a feature of the language that must be defined in order to adhere to the standard.

implementation-dependent a feature of the language that is not in the standard and which may not be portable.

shall indicates a requirement of the standard: the compiler must perform the specified action and/or detect deviations.

error a specific transgression of the standard that requires some course of action.

The report also uses the standard's conventions for reference. For example, the results of each test indicate the applicable subsection by number, e.g. 6.1.1-3 for the test of an error caused by accessing a field with an undefined-value. Letter-number combinations refer to the standard's definitions, as follows:

E.1 to E.16 implementation-defined features

D.1 to D.59 error numbers

F.1 to F.11 implementation-dependent features

A final note: Users may decide upon the relevance of this data. For example, the authors of *A Readers' Guide...* point out that "failure of each quality test is reported and it is up to the user to determine whether any failure indicates a weakness that is critical to his application."

PASCAL VALIDATION REPORT

PASCAL PROCESSOR IDENTIFICATION
MACHINE : Digital PDP-11
COMPILER : Pascal-2.1E, to be released spring 1986
OPERATING SYSTEM : PDP-11 RSX ANE hosted on VAX/VMS 3.2

TEST CONDITIONS
DATE : February 11, 1986
TESTER : Don Baccus, Oregon Software, Inc
TEST SUITE VERSION : 4.1
EVALUATOR : J B Souter

COMPILER OPTIONS USED DURING VALIDATION
"Standard" switch used
default values for other switches were used (range checking, index bounds checking, pointer checking, source walkback enabled).

Statement of compliance

The above processor complies with the requirements of level 1 of ISO 7185 with no exceptions.

The implementation-defined features are as follows:

E.1 The value of each char-type corresponding to each allowed string-character is the corresponding ISO character
E.2 Default subset of real number denoted by signed-real are the values representable by Digital's 32-bit floating point format, about 7 digits of accuracy. User may specify Digital's 64-bit floating point format as a compile time option, giving about 16 digits of accuracy.
E.4 The values of the char-type are the ISO character set, see ISO 646.
E.5 Lazy-I/O is implemented.
E.6 The value of maxint is 32767.
E.7 The accuracy of the approximations of the real operations is determined by which of the two formats described in E.2 is chosen and by rounding of intermediate results. The processor uses Oregon Software's proprietary implementation of standard functions on real numbers, and is limited to roughly 6 digits precision in single precision mode.
E.8 Default TotalWidth for integers is 7 characters.
E.9 Default TotalWidth for reals is 13 characters.

E.10 Default TotalWidth for booleans is 5 characters.
E.11 The value of ExpDigits is 2
E.12 The exponent character is e (lower case)
E.13 Boolean values output in upper case
E.14 Page outputs form-feed
E.15 Input and Output may be bound by means of RSX commands
before execution. Other files are local to current
execution unless non-standard features are invoked.
E.16 Rewrite to standard output does not overwrite previous
output unless it has been bound to a non-record device
(mass storage) by RSX commands invoked before execution.
Reset sets file variable to next record of standard file
output (normally user's terminal) unless it was previously
bound external to the execution as previously described,
in which case file variable points to first record of the
file.
E.17 All equivalent symbols are provided.

The following errors are detected:

D.1, D.3, D.4, D.7, D.10, D.14, D.15, D.16,
D.23, D.24, D.28, D.29, D.33, D.34, D.37, D.40,
D.41, D.44, D.45, D.49, D.51, D.54, D.56, D.57,
D.59,

The processor does not contain any extensions to ISO 7185
when the "standard" compilation switch is used.

Implementation dependent features F.1-F.8, F.10 and F.11 are
treated as undetected errors, and our processor does not
guarantee any particular order of evaluation where permitted
by the standard, and may evaluate similar expressions in
different order in different contexts. Non-file variables
denoted as program parameters are not bound in any way to
the external environment of the program (F.9).

DETAILS OF TESTS

CONFORMANCE TESTS

Total number of Conformance tests = 215
Number of tests passed = 213
Number of tests withdrawn = 2

The following tests were withdrawn since they contain uninitialized
variables (correctly detected by the processor):

6.1.1-3
This test contains every valid pair of adjacent lexical
units.

6.6.3.7.1-6

This test checks that the final array dimension of a schema can be packed and also checks the bound-identifier values.

DEVIANCE TESTS

Total number of Deviance tests = 252
Number of tests which detected deviations = 252
Number of tests which did not detect deviations = 0

ERRORHANDLING TESTS

The manufacturer claimed detection of the following error numbers
D.1, D.3, D.7, D.10, D.14, D.15, D.16, D.23,
D.24, D.28, D.29, D.33, D.34, D.40, D.41, D.44,
D.45, D.49, D.51, D.54, D.56, D.57, D.59,
Total number of Errorhandling tests = 86
Number of pretests which passed = 86
Number of pretests which failed = 0
Number of tests which detected errors = 49
Number of tests which did not detect errors = 37

Details of tests in which an error condition was not detected:

6.4.3.3-10 Error Number = D.2
This test causes an error by accessing a field of an inactive variant.

6.4.3.3-11 Error Number = D.43
This program causes an error by accessing a field with an undefined-value.

6.4.3.3-12 Error Number = D.43
This program accesses a field of a variant which is not the current variant, thereby changing the selected variant, but causes an error due to the undefined-value.

6.4.3.3-13 Error Number = D.43
This program causes an error by accessing a field with an undefined-value.

6.4.6-12 Error Number = D.50
This test violates the assignment-compatibility rules for set-types.

6.4.6-13 Error Number = D.8
This test violates the assignment-compatibility rules for sets passed as parameters.

6.5.4-2 Error Number = D.4
This test causes an error since the pointer-variable

has an undefined value when it is dereferenced.

6.5.5-2 Error Number = D.6
This program causes an error to occur by changing the current file position of a file, while the buffer-variable is an actual variable parameter to a procedure.

6.5.5-3 Error Number = D.6
This test causes an error by altering the value of a file-variable when a reference to the buffer-variable exists.

6.6.2-7 Error Number = D.48
This program contains a function with an assignment to its identifier, however the assignment is never executed.

6.6.5.2-9 Error Number = D.12
This test causes an error by applying 'put' to an undefined buffer-variable.

6.6.5.2-12 Error Number = D.17
This program is in error since the value of the buffer-variable is not assignment-compatible with the variable-access, as a result of a call of read.

6.6.5.2-14 Error Number = D.18
This test causes an error by writing an expression which is not assignment-compatible with the buffer-variable, as a result of a call of write.

6.6.5.3-6 Error Number = D.5
This program causes an error to occur as a variable which is currently an actual variable parameter is referred to by the pointer parameter of dispose.

6.6.5.3-7 Error Number = D.5
This program causes an error to occur as a variable which is an element of the record-variable-list of a with-statement is referred to by the pointer parameter of dispose.

6.6.5.3-8 Error Number = D.25
This program causes an error to occur, as a variable created by the use of the long form of new is used as an operand in an expression.

6.6.5.3-9 Error Number = D.25
This program causes an error to occur, as a variable created by the long form of new is used as the variable in an assignment-statement.

6.6.5.3-10 Error Number = D.25

This program causes an error to occur, as a variable created by the long form of new is used as an actual parameter.

6.6.5.3-11 Error Number = D.43
This program is illegal since it uses the value of a pointer after a call of dispose.

6.6.5.3-13 Error Number = D.19
This test causes an error by activating a variant which is different from those specified by new(p,c1, ,cn).

6.6.5.3-14 Error Number = D.20
This test causes an error since an identifying-value which had been created using the form new(p,c1, ,cn) is consequently removed by applying dispose(p).

6.6.5.3-16 Error Number = D.21
This test causes an error by calling dispose(q,k1, ,km) with an incorrect number of parameters.

6.6.5.3-17 Error Number = D.22
This test causes an error by calling dispose(q,k1, ,km) when the variants in the variable identified by the pointer q, are different from those specified by the case-constants k1, ,km.

6.6.5.3-21 Error Number = D.4
This test causes an error by dereferencing a dangling pointer.

6.6.6.2-13 Error Number = D.32
This program causes an error to occur by a call of the function sqr such that the resulting value does not exist with an integer parameter.

6.6.6.4-5 Error Number = D.38
This program causes an error to occur as the function succ is applied to the last value of an ordinal-type.

6.6.6.4-6 Error Number = D.39
This program causes an error to occur as the function pred is applied to the first value of an ordinal-type.

6.6.6.4-7 Error Number = D.37
This test evokes an error by pushing chr past the limits of the char-type. It assumes that no char-type has more than 1000 + ord('0') values.

6.6.6.5-7 Error Number = D.42
This test causes an error by applying the eoln function to a file f while eof(f) is true.

6.7.2.2-12 Error Number = D.47

This program attempts to output a value which overflows.

6.7.2.2-13 Error Number = D.46
This program incorporates the error of j being negative in the expression i mod j.

6.7.2.2-19 Error Number = D.47
This test contains a simple-expression in which the value of a term exceeds maxint.

6.7.2.4-4 Error Number = D.50
This test checks that operations on overlapping sets are detected.

6.9.1-10 Error Number = D.55
This test causes an error by reading an integer whose value is not assignment-compatible with the type possessed by the variable-access.

6.9.3.1-2 Error Number = D.58
This test attempts to output characters whose field width parameters are non-positive.

6.9.3.1-3 Error Number = D.58
This program contains the error of FracDigits being less than one.

6.9.3.1-7 Error Number = D.58
This program contains the error of FracDigits being less than one.

IMPLEMENTATION DEFINED TESTS

Total number of Implementation Defined tests = 17
Number of tests passed = 17
Number of tests failed = 0

Details of successful tests:

6.1.9-5 Reference Number = E.17
ALTERNATE SUBSCRIPT BRACKETS IMPLEMENTED
ALTERNATE COMMENT DELIMITERS IMPLEMENTED

6.4.2.2-10 Reference Number = E.6
THE VALUE OF MAXINT IS 32767

6.4.2.2-11 Reference Number = E.2
ACCURACY OF UNSIGNED-REAL IS
8 DECIMAL PLACES

6.4.2.2-12 Reference Number = E.4
ORDINAL VALUES OF CHARACTERS

VALUE	CHAR BETWEEN DECIMAL POINTS
65	.A. UPPER CASE LETTER
66	.B. UPPER CASE LETTER
67	.C. UPPER CASE LETTER
68	.D. UPPER CASE LETTER
69	.E. UPPER CASE LETTER
70	.F. UPPER CASE LETTER
71	.G. UPPER CASE LETTER
72	.H. UPPER CASE LETTER
73	.I. UPPER CASE LETTER
74	.J. UPPER CASE LETTER
75	.K. UPPER CASE LETTER
76	.L. UPPER CASE LETTER
77	.M. UPPER CASE LETTER
78	.N. UPPER CASE LETTER
79	.O. UPPER CASE LETTER
80	.P. UPPER CASE LETTER
81	.Q. UPPER CASE LETTER
82	.R. UPPER CASE LETTER
83	.S. UPPER CASE LETTER
84	.T. UPPER CASE LETTER
85	.U. UPPER CASE LETTER
86	.V. UPPER CASE LETTER
87	.W. UPPER CASE LETTER
88	.X. UPPER CASE LETTER
89	.Y. UPPER CASE LETTER
90	.Z. UPPER CASE LETTER
48	.0. DIGIT
49	.1. DIGIT
50	.2. DIGIT
51	.3. DIGIT
52	.4. DIGIT
53	.5. DIGIT
54	.6. DIGIT
55	.7. DIGIT
56	.8. DIGIT
57	.9. DIGIT
43	.+. PLUS
45	.-. MINUS
42	.*. MULTIPLY
47	./. DIVIDE
61	.=. EQUALS
60	.<. LESS THAN
62	.>. GREATER THAN
46	... DECIMAL POINT
44	.,. COMMA
58	.:. COLON
59	.;. SEMICOLON
94	.^. UP-ARROW OR COMMERCIAL AT
40	.(. OPEN ROUND BRACKET
41	.). CLOSED ROUND BRACKET
32	. . SPACE
39	.'. APOSTROPHE
97	.a. LOWER CASE LETTER

```
        98          .b. LOWER CASE LETTER
        99          .c. LOWER CASE LETTER
       100          .d. LOWER CASE LETTER
       101          .e. LOWER CASE LETTER
       102          .f. LOWER CASE LETTER
       103          .g. LOWER CASE LETTER
       104          .h. LOWER CASE LETTER
       105          .i. LOWER CASE LETTER
       106          .j. LOWER CASE LETTER
       107          .k. LOWER CASE LETTER
       108          .l. LOWER CASE LETTER
       109          .m. LOWER CASE LETTER
       110          .n. LOWER CASE LETTER
       111          .o. LOWER CASE LETTER
       112          .p. LOWER CASE LETTER
       113          .q. LOWER CASE LETTER
       114          .r. LOWER CASE LETTER
       115          .s. LOWER CASE LETTER
       116          .t. LOWER CASE LETTER
       117          .u. LOWER CASE LETTER
       118          .v. LOWER CASE LETTER
       119          .w. LOWER CASE LETTER
       120          .x. LOWER CASE LETTER
       121          .y. LOWER CASE LETTER
       122          .z. LOWER CASE LETTER

 6.6.6.2-11      Reference Number = E.2
       BETA =    2
          T =   24
        RND =    1
       NGRD =    0
     MACHEP =  -24
     NEGEP  =  -24
       IEXP =    8
     MINEXP = -126
     MAXEXP =  124
        EPS = 5.960464e-08
     EPSNEG = 5.960464e-08
       XMIN = 1.175494e-38
       XMAX = 2.126765e+37

 6.7.2.2-17      Reference Number = E.7
     ACCURACY OF REAL OPERATIONS IS
     ABOUT  7 DECIMAL PLACES

 6.9.3.1-1       Reference Number = E.9
     DEFAULT OUTPUT WIDTH FOR REALS
     TOTALWIDTH DEFAULT VALUE =   13 CHARACTERS

 6.9.3.1-8       Reference Number = E.10
     DEFAULT OUTPUT WIDTH FOR BOOLEANS
     TOTALWIDTH DEFAULT VALUE =    5 CHARACTERS

 6.9.3.1-9       Reference Number = E.8
```

```
        DEFAULT OUTPUT WIDTH FOR INTEGERS
        TOTALWIDTH DEFAULT VALUE =    7 CHARACTERS

    6.9.3.4.1-1     Reference Number = E.11
        NUMBER OF DIGITS IN AN EXPONENT
        EXPDIGITS IS    2

    6.9.3.4.1-2     Reference Number = E.12
        IMPLEMENTATION DEFINED EXPONENT CHARACTER
        IS e
        LOWER CASE

    6.9.3.5-1       Reference Number = E.13
        CASE OF BOOLEAN VALUES
        TRUE, FALSE
        UUUU, UUUUU

    6.10-9          Reference Number = E.16
        FIRST MESSAGE: IF THIS IS THE FIRST LINE OF OUTPUT,
        THEN REWRITE DOES NOT OVERWRITE PREVIOUS OUTPUT.
        SECOND MESSAGE: IF THIS IS THE FIRST LINE OF OUTPUT,
        THEN REWRITE CAUSES PREVIOUS OUTPUT TO BE OVERWRITTEN.

    6.4.2.2-13      Reference Number = E.4
        RANGE OF ORDINAL VALUES OF CHAR-TYPE IS
        FROM        0 TO      255

    6.1.9-6         Reference Number = E.17
        EQUIVALENT SYMBOL TO UP-ARROW IS IMPLEMENTED


IMPLEMENTATION DEPENDENT TESTS

Total number of Implementation Dependent tests =  16
Number of tests which produced informative output =  16
Number of tests which did not produce informative output =   0

Details of tests which produced informative output:

    6.5.3.2-6       Reference Number = F.1
        EVALUATION ORDER OF V(.A,B,C.) IS ABC

    6.6.5.2-16      Reference Number = F.10
        NUMBER OF EVALUATIONS OF F IN READ(F,A,B,C) IS  1

    6.6.5.2-17      Reference Number = F.10
        NUMBER OF EVALUATIONS OF F IN WRITE(F,A,B,C) IS  1

    6.6.5.4-8       Reference Number = F.11
        ORDER OF EVALUATION OF PACK(A,I,Z) IS AZI

    6.6.5.4-9       Reference Number = F.11
        ORDER OF EVALUATION OF UNPACK(Z,A,I) IS ZAI
```

```
 6.7.1-11        Reference Number = F.2
     ORDER OF EVALUATION OF (. A, B, C .) IS ABC

 6.7.1-12        Reference Number = F.3
     ORDER OF EVALUATION OF (. A..B .) IS AB

 6.7.1-14        Reference Number = F.3
     ORDER OF EVALUATION OF (. A .. B .) IS AB

 6.7.2.3-3       Reference Number = F.4
     TEST OF SHORT CIRCUIT EVALUATION OF (A AND B)
     BOTH EXPRESSIONS EVALUATED

 6.7.2.3-4       Reference Number = F.4
     TEST OF SHORT CIRCUIT EVALUATION OF (A OR B)
     BOTH EXPRESSIONS EVALUATED

 6.7.3-2         Reference Number = F.5
     ORDER OF EVALUATION OF F(F(A,B),F(C,D)) IS ABCD

 6.8.2.2-1       Reference Number = F.6
     TEST OF BINDING ORDER (A[I] := EXPRESSION)
     SELECTION THEN EVALUATION

 6.8.2.2-2       Reference Number = F.6
     TEST OF BINDING ORDER (P^ := EXPRESSION)
     SELECTION THEN EVALUATION

 6.8.2.3-2       Reference Number = F.7
     ACTUAL PARAMETERS EVALUATED IN FORWARD ORDER

 6.9.5-3         Reference Number = F.8
     SEQUENCE CORRESPONDING TO PAGE
     CHAR, ORD(C)=       12


QUALITY TESTS

Total number of Quality tests =  64
Number of tests passed =  48
Number of tests failed =  16

Details of failed tests:

   1.2-3
      This program is a test on the speed of procedure calls.

   6.1.5-10
      This program checks that there are no small limits on
      the number of real literals allowed by a processor.

   6.4.3.4-8
      This test contains some very complicated
      set-constructors, in particular it contains 'set of
```

-10..10'. Processors will rarely pass this test.

6.5.1-2
This test checks that 300 identifiers are allowed in a variable-declaration-part.

6.5.3.2-3
This program checks that array indices can be nested 10 deep.

6.6.1-8
This test checks that procedures may be nested to 15 levels.

6.6.6.2-8
This test checks the implementation of the exp function.

6.6.6.2-9
This test checks the implementation of the sin and cos functions.

6.6.6.2-10
This test checks the implementation of the ln function.

6.7.1-3
This program checks that deeply-nested expressions are permitted.

6.7.1-15
This test constructs a null set by using a member-designator of the form maxint..-maxint.

6.8.3.2-2
This program checks that a procedure may have 300 statements.

6.8.3.4-2
This program checks that deeply-nested if-statements are permitted.

6.8.3.9-20
This program checks that for-statements can be nested 15 deep.

6.8.3.10-7
This test checks that with-statements may be nested to 15 levels.

6.9-3
This test writes a line of length 255 to a textfile.

Details of successful tests:

6.1.3-3
NUMBER OF SIGNIFICANT CHARACTERS >= 20

6.7.2.2-14
REAL DIVISION IS SUPPORTED IN SENSE OF W S BROWN

6.9.1-6
READ OF REALS SUPPORTED IN SENSE OF W S BROWN

6.9.1-8
READ/WRITE OF REALS IS APPROXIMATE

6.9.3.4.2-2
WRITE OF REALS IS APPROXIMATE

EXTENSION TESTS

Total number of Extension tests = 4
All Extension tests have been rejected

STATISTICAL SUMMARY OF THE VALIDATION

			Total
Conformance	213/2	PASS/WITHDRAWN	215
Deviance	252/0	PASS/FAIL	252
Pretests	86/0	PASS/FAIL	86
Error Handling	49/37	DETECTED/NOT DETECTED	86
Implementation Defined	17/0	PASS/FAIL	17
Implementation Dependent	0/16	DETECTED/NOT DETECTED	16
Quality	48/16	QUALITY/FAIL	64
Extension	4/0	DETECTED/NOT DETECTED	4
		Grand Total	740